Online Anomaly Detection in Unmanned Vehicles

Eliahu Khalastchi[1], Gal A. Kaminka[2], Meir Kalech[1], Raz Lin[2]
[1]DT Labs, Information Systems Engineering, Ben-Gurion University
Beer Sheva, Israel 84105
eli.kh81@gmail.com, kalech@bgu.ac.il

[2]The MAVERICK Group, Department of Computer Science, Bar-Ilan University
Ramat-Gan, Israel 52900
{galk,linraz}@cs.biu.ac.il

ABSTRACT

Autonomy requires robustness. The use of unmanned (autonomous) vehicles is appealing for tasks which are dangerous or dull. However, increased reliance on autonomous robots increases reliance on their robustness. Even with validated software, physical faults can cause the controlling software to perceive the environment incorrectly, and thus to make decisions that lead to task failure. We present an online anomaly detection method for robots, that is light-weight, and is able to take into account a large number of monitored sensors and internal measurements, with high precision. We demonstrate a specialization of the familiar Mahalanobis Distance for robot use, and also show how it can be used even with very large dimensions, by online selection of correlated measurements for its use. We empirically evaluate these contributions in different domains: commercial Unmanned Aerial Vehicles (UAVs), a vacuum-cleaning robot, and a high-fidelity flight simulator. We find that the online Mahalanobis distance technique, presented here, is superior to previous methods.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Experimentation

Keywords

anomaly detection, Mahalanobis Distance , uncertainty, machine learning, robotics

1. INTRODUCTION

The use of unmanned vehicles and autonomous robots is appealing for tasks which are dangerous or dull, such as surveillance and patrolling [1], aerial search [9], rescue [2] and mapping [19]. However, increased reliance on autonomous robots increases our reliance on their robustness. Even with validated software, physical faults in sensors and actuators can cause the controlling software to perceive the environment incorrectly, and thus to make decisions that lead to task failure.

This type of fault, where a sensor reading can be valid, but invalid given some operational or sensory context, is called *contextual failure* [4]. 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.

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 (or alert a human operator). To be useful, such a system has to be computationally light (so that it does not create a computational load on the robot, which itself can cause failures), and detect faults with high degree of both precision and recall. A too-high rate of false positives will lead operators to ignoring the system; a too-low rate makes it ineffective. Moreover, the faults must be detected quickly after their occurrence, so that they can be dealt before they become catastrophic.

In this paper, we focus on online anomaly detection methods for robots. We present methods that are light-weight, and are able to take into account a large number of monitored sensors and internal measurements, with high precision. We make two contributions. First, we argue that in monitoring robots and agents, anomaly detection is improved by considering not the raw sensor readings, but their differential. This is because robots act in the same environment in which they sense, and their actions are expected to bring about changes to the environment (and thus change to their sensor readings). Second, we demonstrate the online use of the Mahalanobis distance—a statistical measure of distance between a sample point and a multi-dimensional distribution—to detect anomalies. The use of Mahalanobis distance is not new in anomaly detection; however, as previous work has shown [12] its use with the high-dimensional sensor data produced by robots is not trivial, and requires determining correlated dimensions. While previous work relied on offline training, to do this, we introduce the use of the lightweight Pearson correlation measure to do this. Taken together, the two contributions lead to an anomaly detection method specialized for robots (or agents), and operating completely on-line.

To evaluate these contributions, we conduct experiments in three different domains: We utilize actual flight-data from commercial Unmanned Aerial Vehicles (UAVs), in which simulated faults were injected by the manufacturer; data from the RV-400 vacuum cleaning robot; and the *Flightgear* flight simulator, which is widely used for research [10, 16, 7]. In all, we experiment with variant algorithms, and demonstrate that the online Mahalanobis distance technique, presented here, is superior to previous methods. The experiments also show that the use of the differential sensor readings improve on competing anomaly detection techniques, and is thus independent of the use of the Mahalanobis distance.

2. RELATED WORK

Anomaly detection has generated substantial research over past

Cite as: Online Anomaly Detection in Unmanned Vehicles, Eliahu Khalastchi, Gal A. Kaminka, Meir Kalech and Raz Lin, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

years. Applications include intrusion and fraud detection, medical applications, robot behavior novelty detection, etc. (see [4] for a comprehensive survey). We focus on anomaly detection in Unmanned (Autonomous) Vehicles (UVs). This domain is characterized by a *large amount of data* from many sensors and measurements, that is typically *noisy* and streamed online, and requires an anomaly to be *discovered quickly*, to prevent threats to the safety of the robot [4].

The large amount of data is produced from a large number of system components comprising of actuators, internal and external sensors, odometry and telemetry, that are each 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 (e.g., [12, 15]). Therefore, methods that produce an anomaly score for each given point, can use calculations that consider the points' density, such as Mahalanobis Distance [12] or K-Nearest Neighbor (KNN) [15]. We repeat such a method here.

When large amounts of data are available, distributions can be calculated, hence, statistical approaches for anomaly detection are considered. These approaches usually assume that the data is generated from a particular distribution, which is not the case for high dimensional real data sets [4]. Laurikkala *et al.* [11] proposed the use of Mahalanobis Distance to reduce the multivariate observations to univariate scalars. Brotherton and Mackey [3] use the Mahalanobis Distance as the key factor for determining whether signals measured from an aircraft are of nominal or anomalous behavior. However, they are limited in the number of dimensions across which they can use the distance, due to run-time issues.

Apart from having to reduce dimensions when using Mahalanobis Distance, the dimensions that are left should be correlated. Recently, Lin *et al.* [12] demonstrated how using an offline mechanism as the Multi-Stream Dependency Detection (MSDD) [14] can assist in finding correlated attributes in the given data and enable use of Mahalanobis Distance as an anomaly detection procedure. The MSDD algorithm finds correlation between attributes based on their values. Based on the results of the MSDD process, they manually defined the correlated attributes for their experiments. However, the main drawback of using the MSDD method is that it consumes many resources and can only be used with offline training. Thus, we propose using a much simpler algorithm, that groups correlated attributes using *Pearson correlation coefficient* calculation. This calculation is both light and fast and therefore can be used online, even on a computationally weak robot.

To distinguish the inherent noisy data from anomalies, Kalman filters are usually applied (e.g., [8, 18, 5]). Since simple Kalman filters usually produce a large number of false positives, additional computation is used to determine an anomaly. For example, Cork and Walker [5] present a non-linear model, which, together with Kalman filters, tries to compensate for malfunctioning sensors of UAVs. We use a much simpler filter that significantly improved the results of our approach. The filter normalizes values using a Z score transformation.

3. ONLINE ANOMALY DETECTION FOR ROBOTS

We begin by describing the problem and outlining our approach. We describe the online training procedure, and the specialization for anomaly detection on robots. Finally, we describe when our approach should flag anomalies and describe our algorithm in detail.

3.1 Problem Description

We deal with the problem of online anomaly detection. Let $A = \{a_1, \ldots, a_n\}$ be the set of attributes that are monitored. Monitored attributes can be collected by internal or external sensors (e.g., $odometry$, $telemetry$, $speed$, $heading$, GPS_x, GPS_y, etc.). The data is sampled every t milliseconds. 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. With each $\vec{i_t}$ given, a decision needs to be made instantly whether or not $\vec{i_t}$ is anomalous.

Past data H (assumed to be nominal) is also accessible. H is an $m \times n$ matrix where the columns denotes the 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, $H = \{\vec{i}_{t-m-1}, \ldots, \vec{i}_{t-1}\}$.

We demonstrate the problem using a running example. Consider a UAV with its actuators that collects and monitors n attributes, such as: air-speed, heading, altitude, roll pitch and yaw, and other telemetry and sensors data. The actuators provides input in a given frequency (usually with 10Hz frequency), when suddenly a fault occurs; for instance, the altimeter is stuck on a valid value, while the GPS's indicated that the altitude keeps on rising. Another example could be that the UAV's stick is moved left or right but the UAV is not responsive, due to icy wings. This is expressed in the unchanging values of the roll and heading. Our goal is to detect these failures, by flagging them as anomalies.

3.2 Online Detection

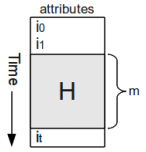

Figure 1: Illustration of the sliding window.

We utilize a *sliding window* technique [4] to maintain H, the data history, online. The sliding window (see Figure 1) is a dynamic window of predefined size m which governs the size of history taken into account in our algorithm. Thus, every time a new input $\vec{i_t}$ is received, H is updated as $H \leftarrow \{\vec{i}_{t-m-1}, \ldots, \vec{i}_{t-1}\}$ the last m online inputs. The data in H is always assumed to be nominal and is used in the *online training* process. Based on H we evaluate the anomaly score for the current input $\vec{i_t}$ using the *Mahalanobis Distance* [13].

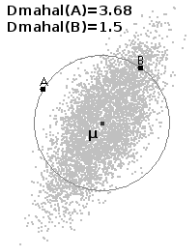

Figure 2: Euclidean vs. Mahalanobis Distance.

Mahalanobis Distance is an n dimensional Z-score. It calculates the distance between an n dimensional point to a group of others, in units of standard deviations [13]. In contrast to the common n dimensional Euclidean Distance, Mahalanobis Distance also considers the points' distribution. Therefore, if the group of points represents an observation, then the Mahalanobis Distance indicates whether a new point is an outlier compared to the 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.

An example is depicted in Figure 2. We can see in the figure 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.

Thanks to the nature of the Mahalanobis Distance, we can utilize it for anomaly detection in our environment. Each of the n attributes of the domain correlates to a dimension. An input vector $\vec{i_t}$ is the n dimensional point, that is measured by Mahalanobis Distance against H. The Mahalanobis Distance is then used to indicate whether each new input point $\vec{i_t}$ is an outlier with respect to H.

Using the Mahalanobis Distance, we can easily detect the three common categories of anomalies [4]:

1. *Point anomalies*: illegal data instances, corresponding to illegal values in $\vec{i_t}$.
2. *Contextual anomalies*, that is, data instances that are only illegal with respect to specific context but not otherwise. In our approach, the context is provided by the changing data of the sliding window.
3. *Collective anomalies*, which are related data instances that are legal apart, but illegal when they occur together. This is met with the multi-dimensionality of the points being measured by the Mahalanobis Distance .

An anomaly of any type, can cause the representative point to be apart from the nominal points, in the relating dimension, thus placing it outside of a dense area, and leading to a large Mahalanobis Distance and eventually raising an alarm.

Formally, the Mahalanobis Distance is calculated as follows. Recall that $\vec{i_t} = (i_{t,1}, i_{t,2}, \ldots, i_{t,n})$ is the vector of the current input of the n attributes being monitored, and $H = m \times n$ matrix is the group of these attributes' nominal values. We define the mean of H by $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$, and S is 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 as an anomaly detector is prone to errors without guidance. Recently, Lin *et al.* [12] showed that the success of Mahalanobis Distance as an anomaly detector depends on whether the dimensions inspected are correlated or not. When the dimensions are indeed correlated, a larger Mahalanobis Distance can better indicate *point*, *contextual* or *collective* anomalies. However, the same effect occurs when uncorrelated dimensions are selected. When the dimensions are not correlated, it is more probable that a given nominal input point will differ from the observed nominal points in those dimensions, exactly as in contextual anomaly. This can cause the return of large Mahalanobis Distance and the generating of false alarms.

Therefore, it is imperative to use a *training process* prior to the usage of the Mahalanobis Distance. This process will *find* and *group* correlated attributes, after which Mahalanobis Distance can be *applied per each correlated set of attributes*. Instead of regarding $\vec{i_t}$ as one n dimensional point and use one measurement of Mahalanobis Distance against H, we apply several measurements, one per each correlated set. In the next subsection we describe the work of the training process and how it is applied online.

3.3 Online Training

Finding correlated attributes automatically is a difficult task. Some attributes may be constantly correlated to more than one attribute, while other attribute's values can be dynamically correlated to other attributes based on the characteristics of the data. For example, the *elevation* value of an aircraft's stick is correlated to the aircraft's *pitch* and to the change of height, measured in the differences of the values of the *altitude* attribute. However, this is only true depending 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 is getting more vertical. This, in turn, makes the *elevation* value to correlate to the *heading* value, rather than the height. This example demonstrates how correlation between attributes can change during execution time. Thus, it is apparent that an *online* training is needed to find dynamic correlations between the attributes.

Figure 3 shows a visualization of a correlation matrix, were each $\text{cell}_{i,j}$ depicts the correlation strength between attributes a_i, a_j. The stronger the correlation, the darker the color of the cell. Figure 3 displays three snapshots taken from different time periods of a simulated flight, where 71 attributes were monitored. The correlation change is apparent.

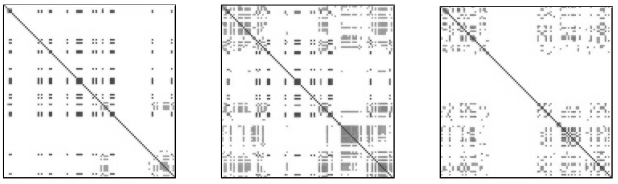

Figure 3: Visualization of correlation change during a flight

We use a fast online trainer, denoted as Online_Trainer(H). Based on the data of the sliding window H, the online trainer returns n sets of dynamically correlated attributes, denoted as $CS = \{CS_1, CS_2, \ldots, CS_n\}$, and a threshold per each set, denoted as $TS = \{threshold_1, \ldots, threshold_n\}$.

The online trainer executes two procedures. The first is a correlation detector (see Alg. 1) that is based on *Pearson correlation coefficient* calculation. Formally, the Pearson correlation coefficient ρ between given two vectors $\vec{X}$ and $\vec{Y}$ with averages $\bar{x}$ and $\bar{y}$, is defined as:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \quad (1)$$

ρ ranges between $[-1, 1]$, where 1 represents a strong positive correlation, and -1 represents a strong negative correlation. Values closer to 0 indicate no correlation.

Algorithm 1 Correlation_Detector(H)

```
for each a_i ∈ A do
    CS_i ← φ
    for each a_j ∈ A do
        if |ρ_{i,j}(H_i^T, H_j^T)| > ct then
            add a_j to CS_i
    add CS_i to CS
return CS
```

Algorithm 1 returns the n sets of correlated attributes, one per each attribute $a_i \in A$. Each CS_i contains the indices of the other attributes that are correlated to a_i. The 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 H_i^T,H_j^T. We then apply the *Pearson correlation* on them denoted as $\rho_{i,j}$. If the absolute result $|\rho_{i,j}|$ is larger than a correlation threshold parameter $ct \in \{0..1\}$, then the attributes are declared correlated and a_j is added to CS_i.

The ct parameter governs the size of the correlated attributes set. On the one hand, the higher it is, less attributes are deemed correlated, thereby decreasing the dimensions and the total amount of calculations. However, this might also prevent attributes from being deemed correlated and affect the flagging of anomalies. On the other hand, the smaller the ct more attributes are considered correlated, thereby increasing the dimensions, and also increasing

the likelihood of false positives, as less correlated attributes are selected.

The second procedure sets a threshold value per each correlated set. These thresholds are later used by the *Anomaly Detector* (see Alg. 2) to declare an anomaly if the anomaly score of a given input crossed a threshold value. Each $threshold_a \in TS$ is set to be the highest Mahalanobis Distance of points with dimensions relating the attributes in CS_a extracted from H. Since every point in H is considered nominal, then any higher Mahalanobis Distance indicates an anomaly.

3.4 Specializing Anomaly Detection for Robots

Monitoring in the domains of autonomous robots is unique and have special characteristics. The main difference emerges from the fact that we are required to monitor using the data obtained from sensors that are used in the control loop to affect the environment. In other words, the expectations to see changes in the environment are a function of the actions selected by the agent.

Therefore, it makes sense to monitor the change 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 to the lose 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 $\triangle(\vec{i_t}) = \vec{i_t} - \vec{i_{t-1}}$.

To eliminate false positives caused by the uncertainty inherent in the sensors' readings, and also to facilitate the reasoning about the relative values of attributes, 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 using 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 then 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, $Z_{raw}(\vec{i_t}) = \{Z(i_{t,1}, H_1^T), \ldots, Z(i_{t,n}, H_n^T)\}$. We also define this transformation on the differential data as $Z_{\triangle}(\vec{i_t}) = Z_{raw}(\triangle(\vec{i_t}))$.

Two aspects emphasize the need to use filters. First, the live feed of data is noisy. Had we used only the last two samples, the noise could have significantly damaged the quality of the differential data. Second, the data feed is received with high frequency. When the frequency of the incoming data is grater than the speed of the change in an attribute, the differential values might equal zero. Therefore, a filter that slows the change in that data, and takes into account its continuity, must be applied. In our simulations we experimented with two types of filters that use the aforementioned Z-transformations, Z_{raw} and $Z_{\triangle}$.

When an actuator is idle, its Z-values are all 0s, since each incoming raw value is the same as 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, up 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, also to be affected by that effect.

Figure 4 illustrates the Z-transformation technique. The data is taken from a segment of a simulated flight. The figure presents values of attributes (Y Axis) through time (X axis). The *aileron* attribute stores the left and right movement of the UAV's stick. These

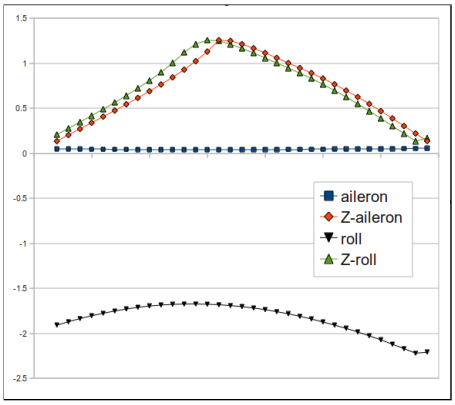

Figure 4: Illustration of the Z-transformation.

movements controls 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 is shown in Figure 4 as the square points, which remains almost constant. Yet, the *roll*'s raw data, marked as an upside triangle, differs significantly from the *aileron*'s data. However, they share a similar ripple effect, illustrated by their Z-transformation values, shown in the triangle points and the diamond points. Thus, our *Pearson* calculation technique can find this correlation quite easily. Other attributes that otherwise could be mistakenly considered correlated when using just the raw data or $\triangle$ technique, will not be considered as such when using the Z-transformation technique, unless they both share a similar ripple effect. This could explain the fact that the $Z_{\triangle}$ technique was proven to be the best one that minimizes the number of false positives as described in Section 4.2

3.5 The Anomaly Detector

Algorithm 2 lists how the anomaly detector works. Each input vector that is obtained online, $\vec{i_t}$, is transformed to $Z_{\triangle}(\vec{i_t})$. The sliding window H is updated. The *online trainer* process retrieves the *sets of correlated attributes* and their *thresholds*. For each correlated set, only the relating dimensions are considered when we compare the point extracted from $\vec{i_t}$ to the points with the same dimensions in H. These points are compared using Mahalanobis Distance. If the distance is larger than the correlated sets' threshold, then an anomaly is declared.

Algorithm 2 Anomaly_Detector($\vec{i_t}$)

$\vec{i_t} \leftarrow Z_{\triangle}(\vec{i_t})$
$H \leftarrow \{\vec{i}_{t-m-1}, \ldots, \vec{i}_{t-1}\}$
$CS, TS \leftarrow$ Online_Trainer(H)
for each a $(0 \leq a \leq |CS|)$ **do**
  Let CS_a be the a'th set of correlated attributes in CS
  Let $threshold_a$ be the a'th threshold, associated with CS_a
  $P_H \leftarrow$ points with dimensions relating to CS_a's attributes extracted from H
  $p_{new} \leftarrow$ point with dimensions relating to CS_a's attributes extracted from $\vec{i_t}$
  if $threshold_a < D_{mahal}(p_{new}, P_H)$ **then**
    declare "Anomaly".

4. EVALUATION

First, we describe the experiments setup; the test domains and anomalies, the different anomaly detectors that emphasize that need of each of our approach's features, and how the scoring is done. Then, we evaluate the influence of each feature of our ap-

proach, and we show how it outperforms other anomaly detection approaches.

4.1 Experiments Setup

We use three domains to test our approach, described in Table 1.

Domain	UAV	UGV	FlightGear
data	real	real	simulated
anomalies	simulated	real	simulated
scenarios	2	2	15
scenario duration (sec)	2100	96	660
attributes	55	25	23
frequency	4Hz	10Hz	4Hz
anomalies per scenario	1	1	4 to 6
anomaly duration (sec)	64, 100	30	35

Table 1: Tested domains and their characteristics.

The first is a commercial *UAV* (Unmanned Aerial Vehicles). The data of two real flights, with simulated faults, was provided by the manufacture. The fault of the first flight is a gradually decreasing value of one attribute. The fault of the second flight is an attribute that froze on a legal value. This fault is specially challenging, because it is associated with an attribute that is not correlated to any others, making it very difficult for our approach to detect the anomaly.

The second domain is a *UGV*. We used a laboratory robot, the *RV400* (see Fig. 5). This robot is equipped with ten sonars, four bumpers and odometry measures. We tested two scenarios. In each scenario the robot went straight, yet it was tangled with a string that was connected to a cart with weight. The extra weight causes the robot to slow down in the first scenario, and completely stop in the second scenario. These scenarios demonstrate anomalies that are a result of the physical objects which are not sensed by the robot. Therefore, the robot's operating program is unaware of these objects as well, leaving the situation unhandled. This domain also presents the challenge of having little data (only 96 seconds of data).

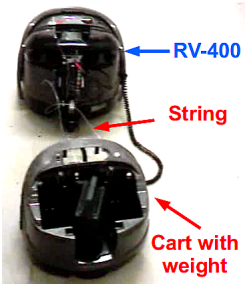

Figure 5: RV-400 tangled with a string connected to a heavy cart.

Figure 6: FlightGear flight simulator.

To further test our approach, on more types of faults and on various conditions, we used a third domain, the *FlightGear* flight simulator (see Fig. 6). *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."[6] 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."[6]

In the *FlightGear* simulation, we programmed an autonomous UAV to fly according to the following behaviors: a take-off, an altitude maintenance, a turn, and eventually a landing. During a flight, 4 to 6 faults were injected into three different components; the *airspeed-indicator, altimeter* and the *magnetic compass*. The faults and their time of injection, were both randomly selected. Each fault could be a contextual anomaly [4] with respect to the UAV's behavior, and a collective anomaly [4] with respect to the measurements of different instruments such as the *GPS airspeed, altitude indicators* and the *Horizontal Situation Indicator*.

Our approach is based on three key features, compared to previous work. 1) a comparison to a *sliding window*, rather than a complete record of past data. 2) the use of an *online training process* to find correlated attributes. 3) the use of *differential filtered data*. To show the independent contribution of each feature we tested the following online anomaly detectors that are described by three parameters (*Nominal Data, Training, Filter*), as summarized in Table 2. The bold line is our recommended approach when using Z_Δ as the $filter$.

Name	Nominal Data	Training
(CD,none,$filter$)	complete past data	none
(SW,none,$filter$)	sliding window	none
(CD,Tcd,$filter$)	complete past data	offline
(SW,Tcd,$filter$)	sliding window	offline
(SW,Tsw,filter)	**sliding window**	**online**

Table 2: Tested Anomaly Detectors.

The $filter$ can be *raw,* Δ, *Zraw,* Z_Δ as described in Section 3.4. CD denotes the use of a Complete record of past Data. SW denotes the use of a Sliding Window. (SW,Tsw,Z_Δ) is our proposed anomaly detector described in section 3.5. (SW,Tcd,$filter$) uses almost the same technique; the thresholds are calculated on the data of the sliding window. However the training is done first, offline, on a complete record of past data. With (CD,Tcd,$filter$), the data of the sliding window is replaced with the data of the complete past record. With (SW,none,$filter$) no training is done, meaning all the dimensions are used at once to compare $\vec{i_t}$ to the data of the sliding window. (CD,none,$filter$) uses all the dimensions to compare $\vec{i_t}$ to the data of a complete past record.

(CD,Tsw,$filter$) is *not* displayed in table 2. This anomaly detector executes the training process on the sliding window, thus, thresholds are calculated online each time different correlated sets are returned. However, the comparison of the online input is made against a complete record of past data, thus, thresholds are calculated on 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, hence, its is not compared to the other anomaly detectors displayed in table 2.

We evaluated the 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.

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

Table 3: Scoring an anomaly detector.

For each algorithm, we calculated the detection rate = $\frac{tp}{tp+fn}$

and the false alarm rate = $\frac{fp}{fp+tn}$. An efficient classifier should maximize the detection rate and minimize the false alarm rate. The perfect classifier has a detection rate of 1, and a false alarm rate of 0.

4.2 Results

Figures 7 and 8 present the detection rate and the false alarm rate respectively of 15 flights in the *FlightGear* simulator. We present the influence of the different filters on the different algorithms. 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.

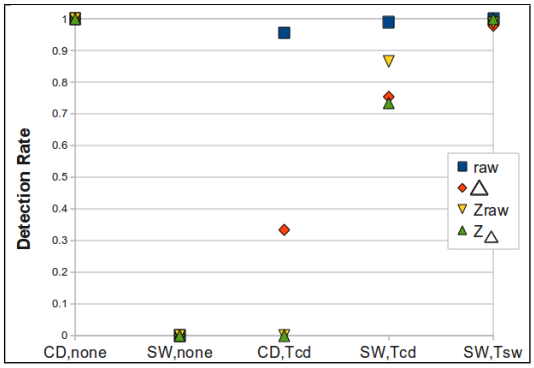

Figure 7: Detection rate. (Higher is better)

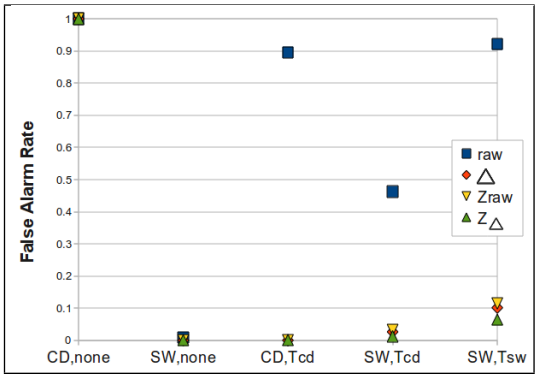

Figure 8: False alarm Rate. (Lower is better)

We begin with the first anomaly detector, (CD,none). Both Figures 7 and 8 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 accounted for the fact that the comparison is made to a complete record of *past* data. Since the new point is sampled from a *different* flight, it is very unlikely for it be observed in the past data, resulting with a higher Mahalanobis Distance than the threshold, and the declaration of an anomaly.

The next anomaly detector we examine is (SW,none). In this detector, the comparison is made to the sliding window. Since data is collected in a high frequency, the values of $\vec{i_t}$ and the values of each vector in H, are very similar. Therefore the Mahalanobis Distance of $\vec{i_t}$ is not very different than the Mahalanobis Distance of any vector in H. Thus the threshold is very rarely crossed. This explains the very low false alarm rate for this algorithm in Figure 8. However, the threshold is not crossed even when anomalies occur, resulting in a very low detection rate as Figure 7 shows. The reason is the absence of training. The Mahalanobis Distance of a contextual or collective anomaly, is not higher than Mahalanobis Distances of points with uncorrelated dimensions in H. The anomalies are not conspicuous enough.

The next two anomaly detectors, introduce the use of offline training. The first (CD,Tcd), uses a complete record of past data, while the second (SW,Tcd) uses a sliding window. However in both anomaly detectors the training is done offline, on a complete record of past data. When no filter is used, (CD,Tcd) declares an anomaly most of the times, this is illustrated in the square dot in Figures 7 and 8. When filters are used, more false negatives occur, expressed in the almost 0 false alarm rates and the decreasing of the detection rate. However, when a sliding windows is used, even with no filters, (SW,Tcd) got 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) lower the false alarm rate to almost 0, but this time, 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) it shows the crucial need of training.

The final anomaly detector is (SW,Tsw) which differs from (SW,Tcd) by the training mechanism. (SW,Tsw) applies an online training on the sliding window. This allows achieving a very high detection rate. Each filter used allows increasing the detection rate closer to 1, until $Z_\triangle$ 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 to nearly 0. (SW,Tsw,$Z_\triangle$), which is the approach we described in section 3.5, achieves a detection rate of 1, and a low false alarm rate of 0.064.

The results show the main contributions of each feature, summarized in table 4

feature	contribution	reason
sliding window	decreases FP	similarity of $\vec{i_t}$ to H.
training	increases TP	correlated dimensions $\rightarrow$ more conspicuous anomalies.
online training	increases TP	correspondence to dynamic correlation changes.
filters	decreases FP increases TP	better correlations are found.

Table 4: Feature Contributions

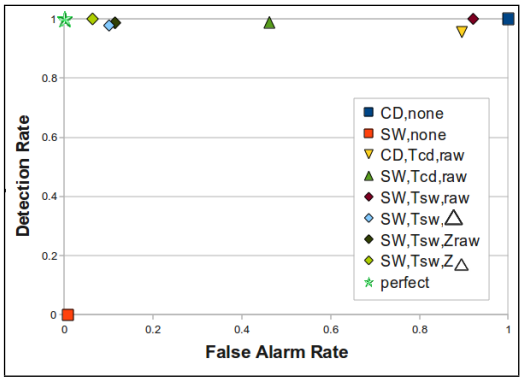

Figure 9: The classifier plane.

Figure 9 describes the entire space of classifiers: 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), that is, it has no false positives, and detects all the anomalies. Figure 9 illustrates that when the features of our approach are applied, they allow the results to approximate the perfect classifier.

Figure 10 shows the detection rates and false alarm rates of (TW,Tsw,$Z_\triangle$) in the classifier space, when we increase the correlation threshold $ct \in \{0..1\}$ in the online trainer described in

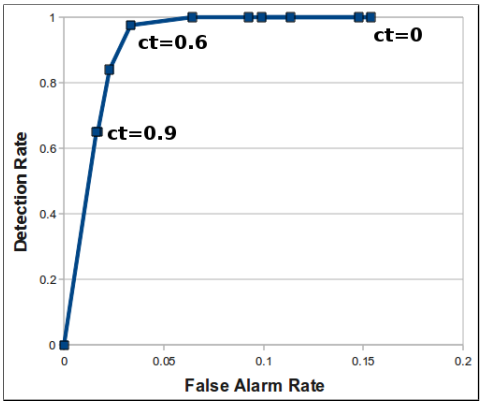

Figure 10: The influence of the correlation threshold.

section 3.3. Note that the X axis scales differently than in Figure 9, it ranges between $[0, 0.2]$ in order to zoom in on the effect.

When ct equals 0 all the attributes are selected for each correlated set, resulting with false alarms. As ct increases, less 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. (TW,Tsw,Z_Δ) averaged a detection rate of 1, and a false alarm rate of 0.064. As ct increases above that peak, less attributes that are crucial for the detection of an anomaly are selected, thereby increasing the false negatives, which in return lowers the detection rate. When ct reaches 1, no attributes are selected, resulting a constant false negative.

To further test our approach, we compare it with other methods. Support Vector Machines (SVM) are considered very successful classifiers when examples of all categories are provided [17]. However, the SVM algorithm classifies every input as nominal, including all anomalies, resulting in a detection rate of 0 as Figure 11 shows. Samples of both categories are provided to the SVM, and it is an offline process, yet, the contextual and collective anomalies are undetected. This goes to show how illusive these anomalies are, which were undetected by a successful and well-known classifier, even under unrealistic favoring conditions.

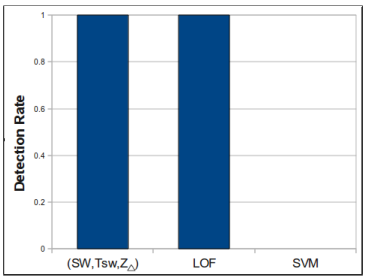

Figure 11: FlightGear Domain Detection Rate

We also examine the quality of (SW,Tsw,Z_Δ) in the context of other anomaly detectors. We compared it to the *incremental LOF algorithm* [15]. As in our approach, the *incremental LOF* returns a density based anomaly score in an online fashion. The *incremental LOF* uses K nearest neighbor technique to compare the density of the input's "neighborhood" against the average density of the nominal observations [15]. Figure 11 shows a detection rate of 1 to (SW,Tsw,Z_Δ) and the *incremental LOF algorithm*, making it a better competitive approach to ours than the SVM.

Since the *incremental LOF* returns an anomaly score rather than an anomaly label, we compared the two approaches using an offline *optimizer algorithm* that gets the anomaly *scores* returned by an anomaly detector, and the anomaly times, and returns the optimal thresholds, which in retrospect, the anomaly detector would have *labeled* the anomalies, in a way that all anomalies would have been detected with a minimum of false positives.

Figures 12 to 15 show for every tested domain the false alarm rate of

1. (SW,Tsw,Z_Δ)
2. optimized (SW,Tsw,Z_Δ) denoted as OPT(SW,Tsw,Z_Δ)
3. optimized *incremental LOF* denoted as OPT(LOF)

The results of the detection rate for these anomaly detectors is 1 in every tested domain, just like the perfect classifier; all anomalies are detected. Thus, the false alarm rate presented, also expresses the distance to the perfect classifier, where 0 is perfect.

The comparison between (SW,Tsw,Z_Δ) to OPT(LOF) *does not* indicate which approach is better in anomaly detection, since the *incremental LOF* is optimized, meaning, the best *theoretical* results it can get are displayed. However the comparison between OPT(SW,Tsw,Z_Δ) to OPT(LOF) does indicate which approach is better, since both are optimized. The comparison between OPT(SW,Tsw,Z_Δ) to (SW,Tsw,Z_Δ) indicates how better (SW,Tsw,Z_Δ) can theoretically get.

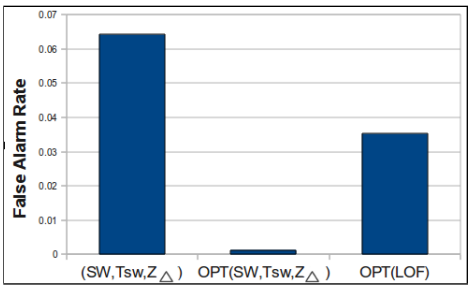

Figure 12: FlightGear domain.

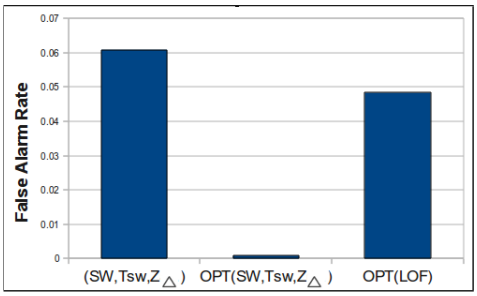

Figure 13: UAV first flight.

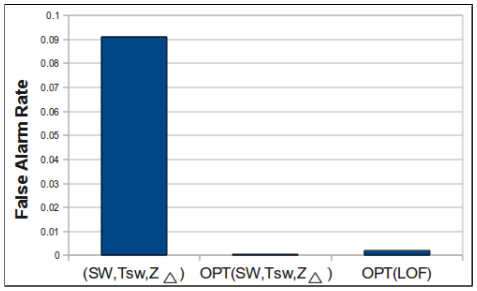

Figure 14: UAV second flight.

In all the domains the OPT(SW,Tsw,Z_Δ) had the lowest false alarm rate. Naturally, OPT(SW,Tsw,Z_Δ) has a lower false alarm rate than (SW,Tsw,Z_Δ), But more significantly, it had a lower false alarm rate than OPT(LOF), making our approach a better anomaly detector than the *incremental LOF algorithm*. Of all the tested domains, the highest false alarm rate of (SW,Tsw,Z_Δ) occurred in the UAV's second flight, as Figure 14 show (little above 0.09). In this flight, the fault occurred in an attribute that is not very correlated

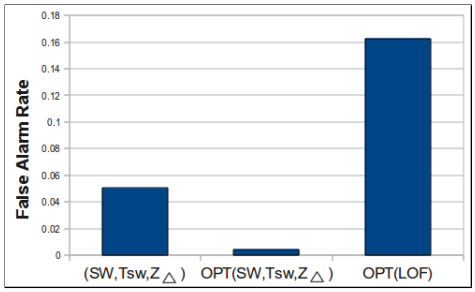

Figure 15: UGV domain.

to any other. Thus, the correlation threshold (ct) had to be lowered. This allowed the existence of a correlated set that includes the faulty attribute as well as other attributes. This led to the detection of the anomaly. However the addition of uncorrelated attributes increased the false alarm rate as well.

Figure 15 show a surprising result. Even though the results of the *incremental LOF* are optimized, (SW,Tsw,$Z_{\triangle}$), which is not optimized, had a lower false alarm rate. This is explained by the fact that in the UGV domain, there was very little data. KNN approaches usually fail when nominal or anomalous instances do not have enough close neighbors [4]. This domain simply did not provide the LOF calculation enough data to accurately detect anomalies. However, the Mahalanobis Distance uses all the points in the distribution, enough data to properly detect the anomalies.

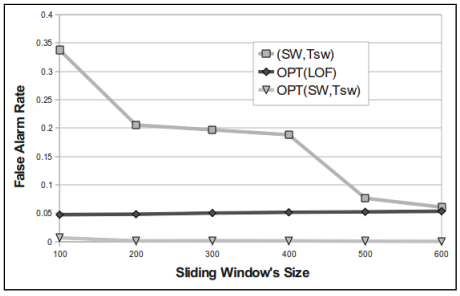

Figure 16: Sliding Window's changing size.

Figure 16 shows the false alarm rate influenced by the increase of the sliding window's size. While Mahalanobis Distance uses the distribution of all the points in the sliding window, the KNN uses only a neighborhood within the window, thus unaffected by its size. Therefore, there exists a size upon which our approach's *real* false alarm rate, meets the *incremental LOF*'s *optimized* false alarm rate.

5. SUMMARY AND FUTURE WORK

We showed an unsupervised, model free, online anomaly detector for robots, that shows a great potential in detecting anomalies while minimizing false alarms. Moreover, the features of the sliding window, the online training and the filtered differential data, made the difference between having an unusable anomaly detector, and an anomaly detector that is better than current existing methods, when applied to robots. However we also showed that with different thresholds, even better results could be obtained. Therefore, in our future work we shall try to select thresholds in a more clever way. Raising an alarm is just the first step towards autonomous self-correcting robots. The next step before diagnosing the cause of the fault, is isolating it. By process of eliminating dimensions, the anomaly, or fault, could be isolated, thus helping a diagnosis process.

Acknowledgments. This research was supported in part by ISF grant #1357/07. As always, thanks to K. Ushi and K. Raviti.

6. REFERENCES

[1] N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *ICRA*, pages 2339–2345, 2008.

[2] A. Birk and S. Carpin. Rescue robotics - a crucial milestone on the road to autonomous systems. *Advanced Robotics Journal*, 20(5), 2006.

[3] T. Brotherton and R. Mackey. Anomaly detector fusion processing for advanced military aircraft. In *IEEE Proceedings on Aerospace Conference*, pages 3125–3137, 2001.

[4] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3):1–58, 2009.

[5] L. Cork and R. Walker. Sensor fault detection for UAVs using a nonlinear dynamic model and the IMM-UKF algorithm. *IDC*, pages 230–235, 2007.

[6] FlightGear. Website, 2010. `http://www.flightgear.org/introduction.html`.

[7] FlightGear in Research. Website, 2010. `http://www.flightgear.org/Projects/`.

[8] P. Goel, G. Dedeoglu, S. I. Roumeliotis, and G. S. Sukhatme. Fault-detection and identification in a mobile robot using multiple model estimateion and neural network. In *ICRA*, 2000.

[9] M. A. Goodrich, B. S. Morse, D. Gerhardt, J. L. Cooper, M. Quigley, J. A. Adams, and C. Humphrey. Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, pages 89–110, 2008.

[10] R. M. J. Craighead and B. G. J. Burke. A survey of commercial open source unmanned vehicle simulators. In *ICRA*, pages 852–857, 2007.

[11] J. Laurikkala, M. Juhola, and E. Kentala. Informal identification of outliers in medical data. In *Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology.* 2000.

[12] R. Lin, E. Khalastchi, and G. A. Kaminka. Detecting anomalies in unmanned vehicles using the mahalanobis distance. In *ICRA*, pages 3038–3044, 2010.

[13] P. C. Mahalanobis. On the generalized distance in statistics. In *Proceedings of the National Institute of Science*, pages 49–55, 1936.

[14] T. Oates, M. D. Schmill, D. E. Gregory, and P. R. Cohen. *Learning from Data: Artificial Intelligence and Statistics*, chapter Detecting Complex Dependencies in Categorical Data, pages 185–195. Springer Verlag, 1995.

[15] D. Pokrajac. Incremental local outlier detection for data streams. In *IEEE Symposium on Computational Intelligence and Data Mining.*, 2007.

[16] E. F. Sorton and S. Hammaker. Simulated flight testing of an autonomous unmanned aerial vehicle using flight-gear. AIAA 2005-7083, Institute for Scientific Research, Fairmont, West Virginia, USA, 2005.

[17] I. Steinwart and A. Christmann. *Support Vector Machines*. Springer-Verlag, 2008.

[18] P. Sundvall and P. Jensfelt. Fault detection for mobile robots using redundant positioning systems. In *ICRA*, pages 3781–3786, 2006.

[19] S. Thrun. Robotic mapping: A survey. In *Exploring Artificial Intelligence in the New Millenium*, pages 1–35. Morgan Kaufmann, 2003.