Anomaly Detection in Unmanned Vehicles: Thesis

Eliahu Khalastchi
Advisor: Gal A. Kaminka
The MAVERICK Group
Department of Computer Science
Bar Ilan University
Ramat Gan, Israel

2010

Contents

Abstract

Autonomy implies 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.

Anomaly detection can also be applied to medical monitors; alarms will be raised whenever the values of the measurements of the patient are anomalous with respect to the patient's current condition. This anomaly detection is particularly useful for medical devices that monitor patients in recovery after a surgery, where the assigned nurses or physicians are not near to watch the monitor.

Model-based diagnosis and fault-detection systems have been proposed to recognize failures. However, these rely on the capabilities of the underlying model, which necessarily abstracts away from the physical reality of the robot.

We present two novel, model-free, domain independent approaches for detecting anomalies in unmanned autonomous vehicles, based on their sensor readings (internal and external). Both approaches use the familiar Mahalanobis Distance for the online anomaly detection. The first approach uses an *offline* training process. With this approach, we show the importance of a training process, which enables the Mahalanobis Distance to detect anomalies successfully. The second approach uses an *online* training process, in a way that is light-weight, and is able to take into account a large number of monitored sensors and internal measurements. These properties make the approach a "plug & play" anomaly detection mechanism for different robotic platforms. We demonstrate a specialization of the 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, a high-fidelity flight simulator, and an electrical power system. We find that the online Mahalanobis distance technique, presented here, is superior to previous methods.

Chapter 1

Introduction

1.1 Anomaly Detection

The use of unmanned vehicles and autonomous robots is appealing for tasks which are dangerous or dull, such as patrolling [1], aerial search [12], rescue [4] and mapping [29]. 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* [6]. 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.

Our method can also be applied to medical medical monitors, where anomalous values of monitored attributes (such as heart rate) can be detected. This anomaly detection is particularly useful for medical devices that monitor patients in recovery after a surgery, where the assigned nurses or physicians are not near to watch the monitor, and should attend the patient if something goes wrong.

In this thesis, we focus on anomaly detection methods for robots. We present two novel approaches for detecting anomalies in the behavior of UVs, using the Mahalanobis distance [19]. 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 shown in this thesis, its use with the high-dimensional sensor data produced by robots is not trivial, and requires determining correlated dimensions.

The first approach consists of a pre-processing phase which finds dependencies between different internal sensors on the vehicles. This *dependency detection* (DD) phase uses an efficient search method—developed for data-mining applications and described in [21]—to identify sub-groups of variables that are statistically dependent (i.e., their values changes together in predictable ways). The results of this phase are therefore several distinct groups of variables—each of much smaller number of dimensions. The second phase, taking place during the execution of the mission, uses the Mahalanobis distance to identify anomalous values in each of the smaller-dimensional groups of variables.

We provide results of extensive experiments conducted using data from commercial UAVs, and in laboratory mobile ground robots. In the experiments, we investigate the efficacy of this approach in detecting anomalies in the UVs' behavior. We also demonstrate the critical role of the first phase.

The second approach uses an online training process that is light-weight, and enables the Mahalanobis Distance to take into account a large number of monitored sensors and internal measurements, and detect anomalies with high precision. While our first approach 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 four 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 that was subjected to physical faults.
- The *Flightgear* flight simulator, which is widely used for research [10, 14, 24] 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) lab at the NASA Ames Research Center [15].

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 specialization for robots also improves competing anomaly detection techniques, and is thus independent of the use of the Mahalanobis distance.

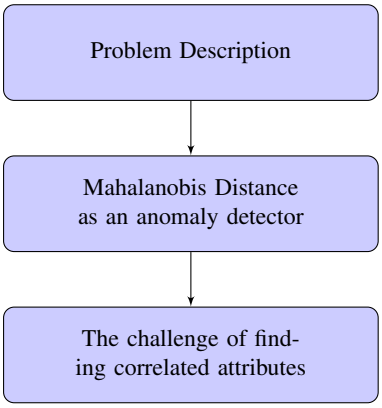
1.2 Thesis Layout

Chapter 2: *Background*

In Chapter 2 we provide the background for the *Anomaly Detection Problem*. We describe the related work, and how our work differs from previous methods.

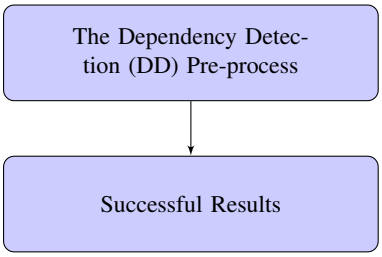
Chapter 3: *Mahalanobis Distance as an Anomaly Detecotr*

In Chapter 3 we describe the problem of anomaly detection in the domain of *Unmanned Vehicles*, how the input is provided and what is expected in return. We describe how the Mahalanobis Distance can be utilized as an anomaly detector that meets the requirements of the problem description, and how it can precisely detect any of the three common types of anomalies. We argue that it is possible only if correlated attributes are used. We then describe and demonstrate the challenge of finding correlated attributes, and why it is not a trivial task.

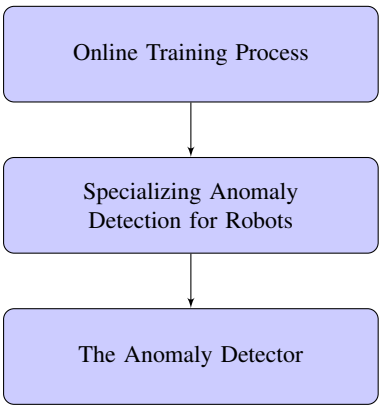


Chapter 4: *Offline Training using Statistical Dependency Detection*

In Chapter 4 we show the importance of a training process. We describe our first approach. We show that only by applying the Mahalanobis Distance on correlated attributes, successful results can be achieved. We show how the *Multi-Stream Dependency Detection (MSDD)* algorithm can be used *offline* for this goal.

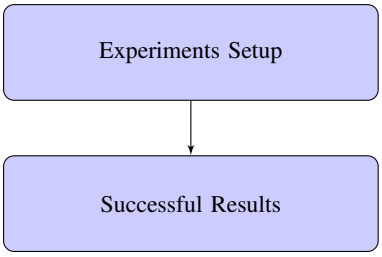


Chapter 5: *Online Anomaly Detection for Robots*



Having showed that applying correlated attributes is critical for the use of Mahalanobis Distance as an anomaly detector, we tried to create a simpler yet effective training process, that can be applied *online*. In Chapter 5 we describe our second approach. We begin by outlining the approach and describing the *online* training process. We continue with the description of how we specialized the anomaly detection for robots. Then, we describe the anomaly detector.

Chapter 6: *Evaluation of the Approach*



In Chapter 6 we evaluate our approach. We describe the four test domains, and a competitive approach to ours. We show the importance of each feature of our approach, and how it is superior to previous approaches.

1.3 List of Publications

This work led to two papers:

1. Raz Lin, Eliahu Khalastchi, and Gal A. Kaminka. *Detecting Anomalies in Unmanned Vehicles using the Mahalanobis Distance*. In International Conference on Robotics and Automation (ICRA) 2010.
2. Eliahu Khalastchi, Meir Kalech, Gal A. Kaminka, and Raz Lin. *Online Anomaly Detection in Unmanned Vehicles*. In Autonomous Agents and Multiagent Systems (AAMAS) 2011.

Chapter 2

Background

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 [6]. These anomalies are usually characterized as being *contextual* or *collective* anomalies, which are very illusive [6]. Past data is available for learning. However, usually only nominal data samples are available [6].

Anomaly detection has generated substantial research over past years. Applications include intrusion and fraud detection, medical applications, robot behavior novelty detection, etc. (see [6] for a comprehensive survey).

Machine learning methods are usually employed to model what constitutes a nominal behavior and deriving from the representation of the nominal behavior the abnormal behavior. For example, Ahmed *et al.* [2, 3] investigate the use of two distinct machine learning approaches, namely the block-based One-Class Neighbor Machine and the recursive Kernel-based Online Anomaly Detection algorithms, to detect network anomaly. Yet, as often happens in machine learning techniques, their models are constrained and cannot be easily adapted to other domains. Besides the fact that it is sensitive to different thresholds, to enable its use in different domains many parameters must be fine tuned. We, on the other hand, demonstrate that our approach can be easily adapted to different domains, while preserving the 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* [6]. Any test instance that falls outside the learnt boundary is considered as anomalous. Support Vector Machines (SVMs) [26], as other machine learning techniques, need additional computation to calculate this boundary in the one-class setting [20, 23]. Kernels, such as *radial basis function (RBF) kernel*, can be used to learn complex regions [6]. However, as we show in the results section, 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 were available for the training. Our proposed unsupervised method detected these anomalies.

Lotze *et al.* [18] studied the problem of detecting anomalies in sequence of real-time data of patience and diseases. Our first approach uses a Dependency Detection method, which is based on the work on multi-stream dependency-detection, described in [21]. However, we use only a subset of the results generated, and thus can potentially alleviate the computational load when applying the Mahalanobis Distance on the selected correlated attributes.

Recently, Daigle *et al.* [8] proposed an event based approach for diagnosis parametric faults in continuous systems. Their approach is based on a qualitative abstraction of deviations from the nominal behavior. Yet, in contrast to our proposed approach, their approach is aimed to diagnose an isolating single fault. Moreover, 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.

Another approach for anomaly detection is based on model based reasoning (e.g., [13, 25]). Yet, this requires to have a model of the robot and its interactions with the environment. Such a model is expensive and complex to build.

The large amount of data of monitored UVs, 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., [17, 22]). Therefore, methods that produce an anomaly score for each given point, can use calculations that consider the points' density, such as Mahalanobis Distance [17] or K-Nearest Neighbor (KNN) [22]. We use 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 [6]. Laurikkala *et al.* [16] proposed the use of Mahalanobis Distance to reduce the multivariate observations to univariate scalars. Brotherton and Mackey [5] 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. We address this challenge here.

To distinguish the inherent noisy data from anomalies, Kalman filters are usually applied (e.g., [7, 11, 27]). 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 [7] 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.

Chapter 3

Mahalanobis Distance as an Anomaly Detector

In this chapter we describe how the Mahalanobis Distance can be used as an anomaly detector in the domain of unmanned vehicles. We start with the problem description of anomaly detection in this domain. We describe how anomalies can be detected by the Mahalanobis Distance, in a way that fits the description of the problem. We describe the three common categories of anomalies and how each can be detected. However, we argue that anomalies can be detected successfully by Mahalanobis Distance, only if correlated dimensions are used. We demonstrate the challenge of finding correlated dimensions.

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 Mahalanobis Distance as an Anomaly Detector

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 [19]. 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.

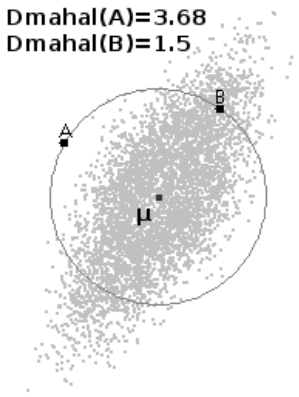


Figure 3.1: Euclidean vs. Mahalanobis Distance.

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:

$$D_{mahal}(\vec{i_t}, H) = \sqrt{(\vec{i_t} - \vec{\mu})S^{-1}(\vec{i_t}^T - \vec{\mu}^T)}$$

An example is depicted in Figure 3.1. 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 [6]:

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.

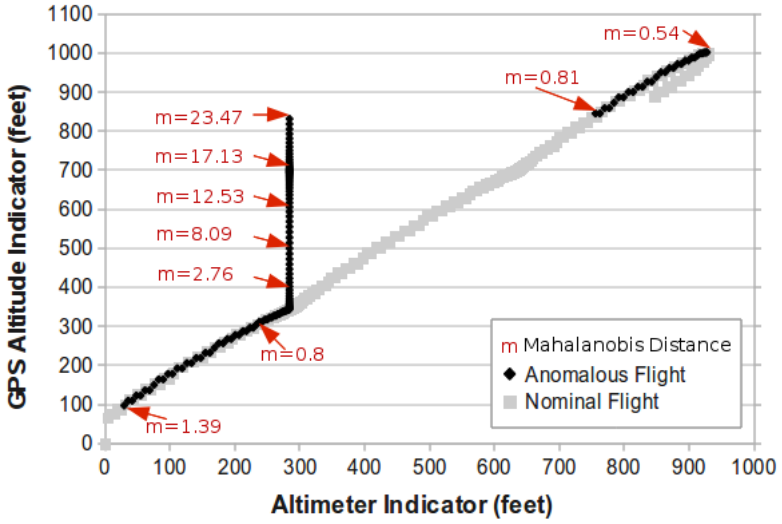


Figure 3.2: Example of Mahalanobis Distance as an Anomaly Detector

Figure 3.2 presents an example of the Mahalanobis Distance as an anomaly detector. The data was taken from simulated flights of *FlightGear fight simulator* [9]. Recall the running example, 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 suppose to change together. However, when one of the attributes is stuck, even on a legal value, it is a *collective anomaly* with the respect to the other attribute. These two attributes are depicted as the two dimensions in Figure 3.2. The square (gray) points present an observation of the values of these attributes, taken from a nominal flight. The diamond (black) points present the values of an anomalous flight. For a period of time, the *GPS Altitude* attribut's values kept on rising while the *Altimeter* attribut's values stayed the same. The Mahalanobis Distance of several points from the anomalous flight with respect to the nominal observation, is depicted in Figure 3.2 as m. Before the anomaly occurs, the Mahalanobis Distance is 0.8 standard deviations. During the anomaly time, as the *GPS Altitude*'s values keep on rising while the *Altimeter*'s values stay the same, the representing point is being located farther away from a dense area, rising the Mahalanobis Distance up to 23.47 standard deviations. After the anomaly time, the *Altimeter*'s values are nominal again, placing the representing point back in a dense area, decreasing the Mahalanobis Distance to 0.81.

Using the Mahalanobis Distance as an anomaly detector is prone to errors without guidance. In this thesis we show 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* that selects correlated dimensions, prior to the usage of the Mahalanobis Distance. The selection of correlated dimensions also acts as a dimension reduction, which is essential for the Mahalanobis Distance calculation to be feasible online.

3.3 The Challenge of Finding Correlated Attributes

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 (see Figure 3.3). 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 is getting more vertical. This, in turn, makes the *elevation* value to correlate to the *heading* value, rather than the height (see Figure 3.4). In the same manner, the *rudder* is correlated the the aircraft's *yaw*, which usually effects the heading. However as the aircraft is being rolled, the *yaw* axis is getting more horizontal. This, in turn, makes the *rudder* value to correlate to the *altitude* value, rather than the heading.

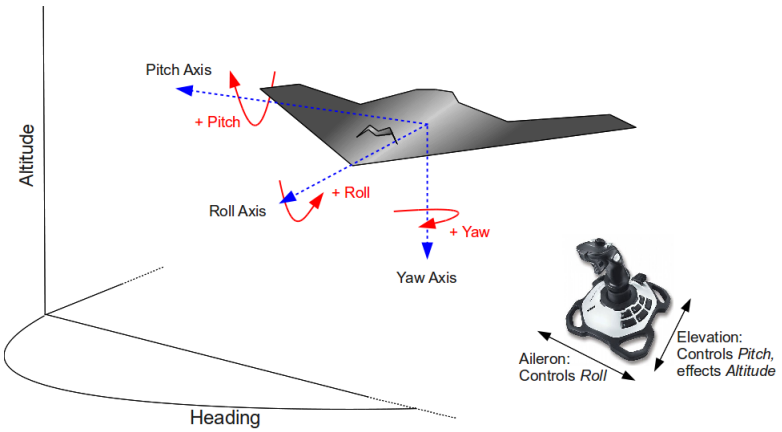


Figure 3.3: The $Altitude$ is affected by the $Elevation$

In the next two chapters, we describe the work of a *training process*. In Chapter 4, we describe our first approach, where we used the *MultiStream Dependency Detection (MSDD)* algorithm [21] as an *offline* training process. We used this preprocess to detect correlated attributes

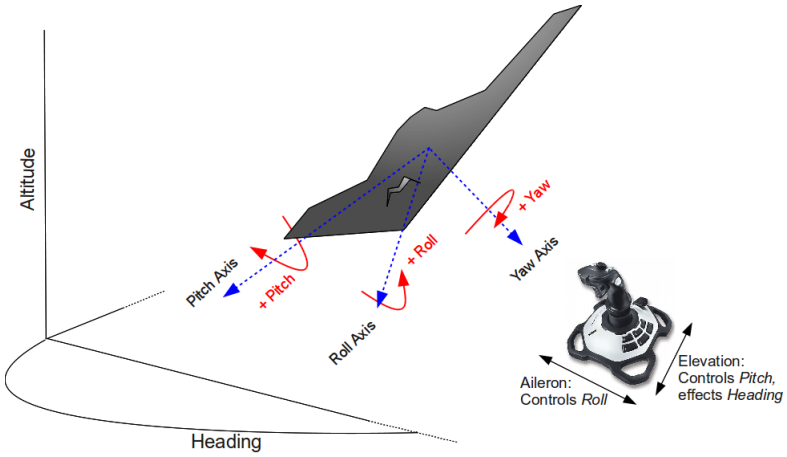


Figure 3.4: The $Heading$ is affected by the $Elevation$

prior to the online anomaly detection process. The dimensions of $\vec{i_t}$ were reduced to match only the correlated attributes. Then, Mahalanobis Distance is used to compare $\vec{i_t}$ to H.

In Chapter 5, we describe our second approach, where an *online* process that *finds* and *groups* correlated attributes is used, 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.

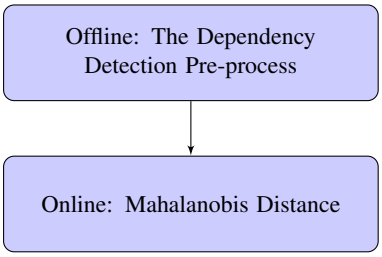
Chapter 4

Offline Training using Statistical Dependency Detection

In this chapter, we describe an approach to find correlated attributes with an offline training process. We show that applying the Mahalanobis Distance on correlated attributes achieves successful results. We show how the *Multi-Stream Dependency Detection (MSDD)* algorithm can be used *offline* for this aim.

The outline of the approach

The Dependency Detection training process runs offline on H (a recored of past nominal data). H is reduced to the learned correlated dimensions. Then online, every input vector $\vec{i_t}$ is reduced to these correlated dimensions and compared to H using Mahalanobis Distance. An anomaly score is returned.



4.1 The Dependency Detection (DD) Pre-process

We introduce a pre-processing mechanism that uses statistical dependency-detection methods to determine possible sub-groups of attributes which are statistically inter-dependent. These sub-groups are then used in the second (online) phase to form the basis for the Mahalanobis distance measurements. Thus, instead of using the Mahalanobis outlier detector on the entire input vector, we break the task into a set of outlier detectors, each focused on parts of the input vector, each using its own nominal distribution H, and each operating in a small-dimensional space (in our experiments, typically 2–3 attributes).

In this work we build on earlier work by Oates *et al.* [21], which have developed efficient data-mining algorithm called Multi-Stream Dependency Detection (MSDD). The algorithm finds statistically significant patterns of the type $A_x B_y \rightarrow C_z D_q$, which should be understood as follows: In an input vector $\vec{v}$, if the value A appears in attribute x, and the value B appears in

attribute y, then the value C will likely appear in attribute z and the value D will likely appear in attribute q. In other words, MSDD is able to determine that attribute values are dependent on each other. The statistical strength of the patterns are measured by the G statistic [30][1]. MSDD uses an efficient heuristic search which guarantees finding the complete set of patterns, without examining the entire combinatorial search space.

MSDD is both too crude and too good for our needs. On one hand, MSDD has the capability for finding such patterns even when they are spread over time (i.e., to find patterns of the form "if attribute x has value A, then in two ticks, we expect attribute y to have value B"), and can thus produce finer-grained information than what is needed for the Mahalanobis distance, which only requires knowledge of general dependencies between variables.Rather than simply outputting a single pattern for each set of dependent attributes, MSDD very often detects redundant dependencies, by finding different variations on the same basic dependence:

$$\begin{aligned} A_x B_y &\rightarrow C_z D_q \\ C_z A_x B_y &\rightarrow D_q \\ C_z D_q &\rightarrow A_x B_y \\ &\dots \end{aligned}$$

We therefore used a modified version of MSDD—called simply Dependency Detection (DD)—in which redundant patterns of the type above are merged together to form groups of dependent attributes (in this case, the set would be x, y, z, q). This modified version, in effect, tells us what attributes are correlated, and this, in turn, allows to run the Mahalanobis distance only on the dependent attributes, thus significantly reducing the dimensionality of the space. Note that often more than one group of dependent attributes would be identified, in which case multiple Mahalanobis outlier detectors should be used, one for each group.

An example of how the whole approach works:

consider a robot with two sonar range detectors which are physically located near one another. The robot can also return the its current speed using odometry or GPS. In a nominal input, in most observations, the values of the attributes of the two sonars are changing together. Thus, the DD process will report an almost constant correlation between these two attributes. However, a correlation between the speed attribute to the sonars attributes, will be significantly less constant; only certain values of the speed attribute might correlate to certain values of the sonars attributes. Thus, only the attributes of the two sonars are manually selected into a correlated set. In the same manner, an additional correlated set might contain the odometery speed attribute and the GPS speed attribute; but no correlated set will contain both speed and sonar attributes.

Online, the attributes of each correlated set are considered as dimensions. The current input vector $\vec{i_t}$ will be broken into multidimensional points. In this example, $\vec{i_t}$ will be broken into a 2D point containing the current values of the sonars attributes, and a 2D point containing the current values of the odometry and GPS speed attributes. These points are compared using the Mahalanobis Distance to the nominal points with the same dimension extracted from the nominal data H.

If no anomaly occurs then the current points will fit the distribution of the nominal points. Thus a low Mahalanobis Distance will be returned. However, if a correlated set was to contain a speed attribute and a sonar attribute, then the current point, even if it contains perfectly legal values, might not fit the distribution of the nominal points, since the occurrence of these two

[1] Similar in principle to the χ^2 statistic.

values together was not observed, due to their being uncorrelated. A high Mahalanobis Distance will be returned, raising a false alarm.

Consider the following anomalies:

- One of the sonars is stuck on a legal value while the other sonar is working properly. The current 2D point will not fit the distribution of the nominal points, since the occurrence of these two values together was not observed, due to the fact that in most observations the two sonars had similar values. Thus, a high Mahalanobis Distance will be returned, justifiably raising an alarm.
- The robot entered a slippery surface. The lack of friction causes the wheels to spin in place. The odometry speed reports a high value while the GPS speed reports a low value. The 2D point will not fit the nominal distribution where the values of the two dimensions were similar. A high Mahalanobis Distance will be returned justifiably raising an alarm.

4.2 Experiments

To evaluate the efficacy of this approach for detecting anomalies in UVs we conducted several sets of experiments. The different experiments demonstrate the strength of the combination between the pre-processing dependency detection mechanism and the online anomaly detection mechanism, as well as the generality of the approach. We begin by describing the experiment setup, and then continue to describe the different experiments and results.

4.2.1 Experiment Setup

We chose two different unmanned vehicles to demonstrate the generality of our approach. The first set of data came from actual commercial unmanned aerial vehicles. The UAV is equipped with several sensors and actuators, as well as a communication system. The communication system transmits the information, along with monitoring information, to the ground station.

The information which is measured by the UAV sensors and is relevant for the anomaly detection process includes more than 50 attributes. The different attributes can be categorized to different families: *air data* (includes telemetry data that the UAV measures), *inertial data* (includes information about the inertial navigation system (INS)), *engine data* (includes 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 is measured by the sensors either in a 1Hz or 10Hz frequencies, yet the whole data is downloaded from the UAV at a frequency of 10Hz.

The second set of experiments was conducted on a commercial vacuum-cleaning mobile robot (the Friendly Robotics RV-400, used in our lab and fitted with our own control software. see Figure 4.1).

Figure 4.1: An RV-400 robot.

The RV-400 robot is equipped with many less sensors and actuators than the UAV. It has 22 attributes measured by ten sonar sensors which measure ranges, 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 is recorded in a 10Hz frequency.

In the course of evaluations, we utilized data from several nominal runs of the robots, as well as from failure runs. We refer to these runs in the discussion of the experiments below.

For the UAV we the following errors were recorded:

- *Descend*: In this error, one of the sensors is malfunctioning and thus causes the sensor's input to decrease rapidly from a valid input to a constant value of zero.
- *Constant*: In this error, one of the sensors is malfunctioning and reports a constant value for a period.

For the UGV the following errors were recorded:

- *Weight Drag Halt*: In this error, the robot was attached to a cart via a fishing string which was loose. Then, the robot started its movement away from the cart, causing the string to stretch, until it was completely stretched. This caused the robot to completely halt.
- *Direction Deviation*: In this error, 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 seconds). It also changed its heading, etc.

For each UV we had a nominal behavior file which was used for two purposes. First, it was used in the pre-processing phase to obtain the strongest dependent attributes for the domain of the UV. Then, it was used in the online anomaly detection process for finding deviations in the behavior of the vehicle from the nominal behavior recorded in that file (i.e., as the basis for the nominal set m). The experiment data sets are summarize in Table 4.1.

4.2.2 Successfully Detecting Anomalies

As we mentioned, the anomaly detection process requires that we first find the attributes that are considered correlated. To this end, we ran the pre-processing mechanism described in Section 4.1 on a nominal data file, Nominal UAV A (Nominal UGV A for the UGV domain). Out of the 56 (22 for the UGV) different attributes that are measured several attributes were found to be significantly strongly correlated (based on the G statistic). We chose to use 2 of the correlated attributes in our anomaly detection process (both in the UAV and the UGV domains). It is notable to mention that in the case of the UGV domain, the MSDD pre-processing mechanism returned somewhat unsurprising correlation between the odometry sensors, yet it also returned a surprising correlation between two sonars on-board the robot. Later we found this dependency highly useful for detecting the anomalies in the UGV experiments.

In the process of detecting the anomalies we need to determine the threshold above which an anomaly is flagged. To this end we first run the Mahalanobis distance algorithm on the nominal file and create a histogram of the standard deviations that are the output of the algorithm. The threshold is then determined in such that at least 93% of the measurements are below it. For the UAV and UGV domains, this generates a threshold of 15 and 0.081 standard deviation units, respectively. We begin by describing the results on the UAV domain and finish with the description of the UGV experiments.

Detecting Anomalies in UAVs

Figure 4.2 shows the results of the Mahalanobis distance algorithm when applied on the Descend UAV C data. Disregarding the end of the flight, in which the behavior of the UAV changes, we

Data Type	Description
Nominal UAV A	Contains nominal flight behavior. This file was also used for the pre-process phase and the comparison of other UAVs' behavior.
Nominal UAV B	Contains an additional nominal flight behavior.
Descend UAV C	Contains an error in a sensor, which rapidly decreases its value until a constant zero. The error is between time units 15,990 and 16,054.
Constant UAV D	Contains an error in a sensor, which value is stuck constant. The error is between time units 8,105 and 8205.
Nominal UGV A	Contains nominal driving behavior. This file was also used for the pre-process phase and the comparison of other UGVs' behavior.
Weight Drag Halt UGV	Contains an error in the nominal driving behavior: The UGV attempts to drag a heavy load, which causes to comes to a complete halt at time unit 100.
Direction Deviation UGV	Contains an error in the nominal driving behavior: The UGV has an object stuck in one of its wheels, causing it to bounce every 5 seconds.

Table 4.1: Description of experiment data.

can see from the figure that in the exact times of the error, the output of the Mahalanobis distance is significantly higher than the threshold. Out of the 64 time units of the error, a total of 59 (92%) were above the 15 threshold.

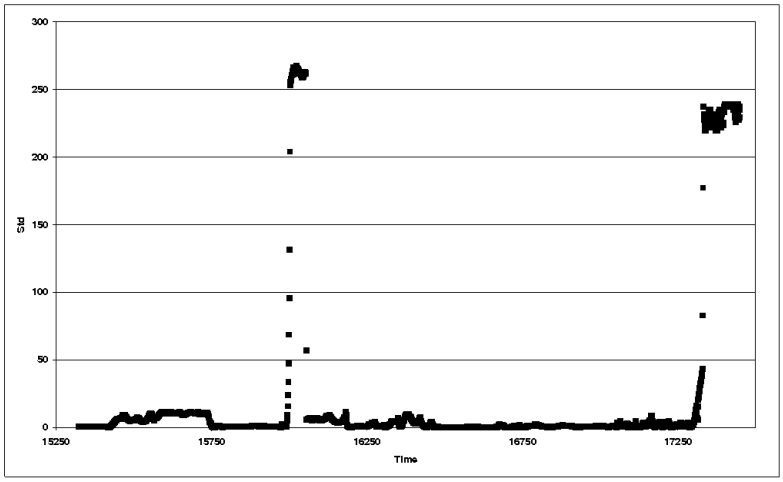


Figure 4.2: Descend UAV C: Mahalanobis distance (in std units) as a function of flight time (1/100 of a minute).

Figure 4.3 shows the results of the Mahalanobis distance algorithm when applied on the Constant UAV D data. Again, we disregard the start and end of the flight time periods (we discuss them later). Unfortunately, though, the algorithm found no evidence of deviations from the nominal behavior in this case. The explanation for this is the fact that "freezing" a sensor on a constant value does not cause deviations from nominal behavior, since the value is legit, and thus the Mahalanobis distance cannot detect these kinds of errors.

Trying to overcome this issue we ran an additional experiment. In this experiment we took the differential of the data per each attribute, and now ran the anomaly detection mechanism to find whether there are deviations from the nominal behavior of the UAV based on the differential of the data. Figure 4.4 now shows the results of this experiment (note that for display purposes we omitted the start and end periods of the flight from the figure's scale). Now we can see that the algorithm was indeed able to find a deviation from the nominal behavior at the end of the error period, just before the sensor re-started reporting normal behavior.

Encouraged by these results, we moved to apply this technique to the UGV experiment data. The results of the approach on the UGV domain is given below.

Detecting Anomalies in UGVs

Figure 4.5 shows the results of the Mahalanobis distance algorithm when used with the UGV Weight Drag data, causing the UGV to halt . In Figure 4.5 we can see that the approach accurately detected the stop movement of the UGV around time unit 100.

Finally, Figure 4.6 depicts the results of the Mahalanobis distance algorithm when applied on the Direction Deviation UGV anomaly. We can see that approximately every 5 seconds the

standard deviation units leap to a value larger than 0.08. An operator at the control station watching this data online (or rather, being notified as the measures pass the threshold) would have been able to detect that there is some malfunction with the robot, which is taking place every 5 seconds.

4.3 The Importance of DD Pre-Processing

The Mahalanobis distance cannot stand on its own to detect anomalies. Its success in detecting anomalies above lies in the fact that the dependency detection pre-processing mechanism was invoked prior to the anomaly detection algorithm, and chose specific attributes on which to focus the Mahalanobis distance measure. Here, we demonstrate the importance of invoking this mechanism.

First, let us examine the run-time of using the Mahalanobis distance as the number of attributes increases. Figure 4.7 demonstrates the importance of narrowing down the input to the Mahalanobis distance algorithm. The figure shows the run-time (in minutes) of the algorithm as a function of the number of attributes in each stream of data it uses to detect the deviation from the nominal behavior. The results demonstrate that the algorithm's run-time increases quickly as a function of the number of attributes. This is a part of the motivation for allowing the DD process to select a smaller set of attributes.

However, it is not simply a case of reducing the number of attributes. To demonstrate the importance of running the Mahalanobis distance on dependent attributes we ran the following experiment.

Here, we built on a predefined knowledge of the UAV domain and chose several attributes which are independent of each other (we verified also that they do not appear in the results of the DD process). We then applied the Mahalanobis outlier detector based on these attributes, to see if we could detect the failures using these attributes instead of those selected by the DD process. We hypothesized that both on the nominal files and the simulated error files the results would generate high rates of false alarms (detecting anomalies even though there is none), making the algorithm useless.

We started by running the algorithm on two different data files which describe a nominal behavior (Nominal UAV A and Nominal UAV B). Then, we applied the same mechanism on a data file which simulated errors in predefined times (Descend UAV C). Figure 4.8 show the percentage of false alarms detected when running the Mahalanobis distance on uncorrelated attributes as compared to running it on correlated attributes.

As we hypothesized, we can see that the approach does not scale well if the input is not fine tuned. While the rates of false alarms when applying the algorithm on dependent attributes is relatively low (0.71%, 0.17% and 0.10% for Nominal UAV A, Nominal UAV B and Descend UAV C, respectively), the rates increase significantly when applied on uncorrelated attributes (2.06%, 34.29% and 60.17% for Nominal UAV A, Nominal UAV B and Descend UAV C, respectively). That is, the algorithm "found" that the nominal flights actually deviated from the nominal behavior, which, of course, was not the case.

As we argued in Section 4.1, a small number of attributes is also important because it facilitates increased accuracy. Figure 4.9 demonstrates that the number of false alarms dramatically increases (compared to the DD-based runs) if too many attributes are used (3.97%, 1.19% and 21.81% for Nominal UAV A, Nominal UAV B and Descend UAV C, respectively, when four attributes are used—compare to 0.71%, 0.17% and 0.10% when using the two strongly-dependent attributes). From the figure we can see the difference in the false alarm ratio when only two of the strongest correlated attributes are used as compared to using four strongest attributes. Thus,

using the DD algorithm to find the K strongest correlated attributes can also allow minimizing false alarms in the anomaly detection process.

4.4 Discussion

We have showed that apart from having to reduce dimensions when using Mahalanobis Distance, the dimensions that are left should be correlated. In this first approach [17], we demonstrated how using an offline mechanism as the Multi-Stream Dependency Detection (MSDD) [21] 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, we manually defined the correlated attributes for our experiments. We have experimented with two domains of physical data of real unmanned vehicles; showing the success of this domain independent approach in the real world.

However, the main drawback of using the MSDD method is that it consumes many resources during its offline training phase. For example, training on the UGV data set required 3 days of computation time, on a PC with 3GB memory and 2 cores. Thus, in the next chapter, 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.

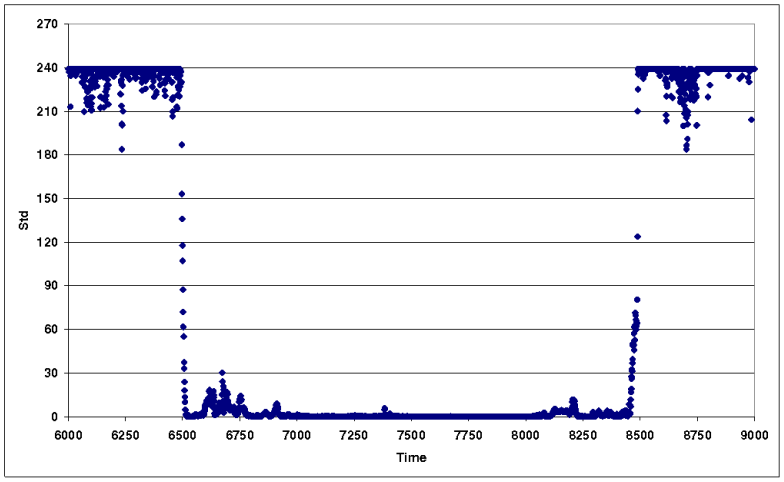


Figure 4.3: Constant UAV D: Mahalanobis distance (in std units) as a function of flight time (1/100 of a minute).

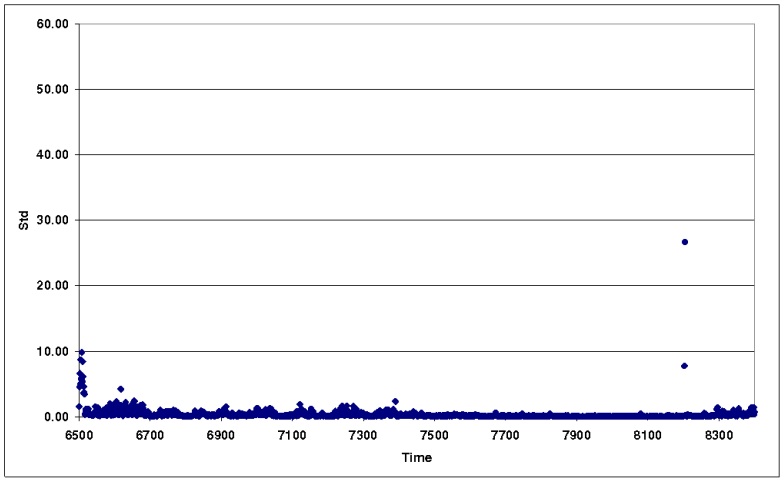


Figure 4.4: Constant UAV D, analysing differential data: Mahalanobis distance (in std units) as a function of flight time (1/100 of a minute).

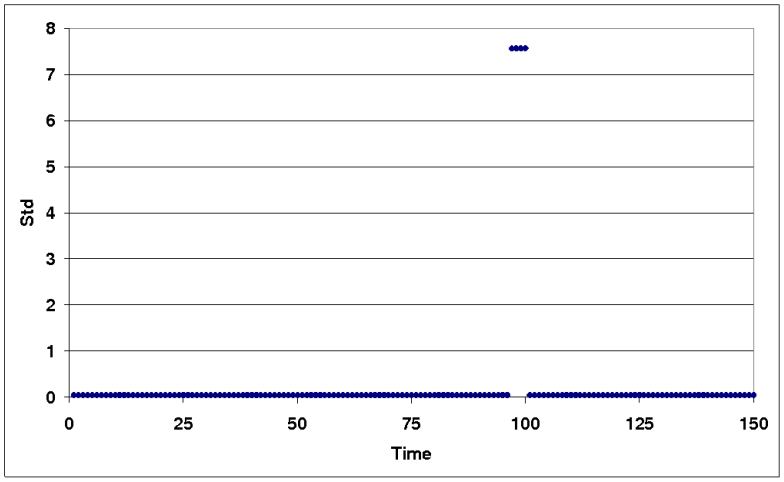


Figure 4.5: Drag Weight Halt UGV: Mahalanobis distance (in std units) as a function of movement time (1/10 of a second).

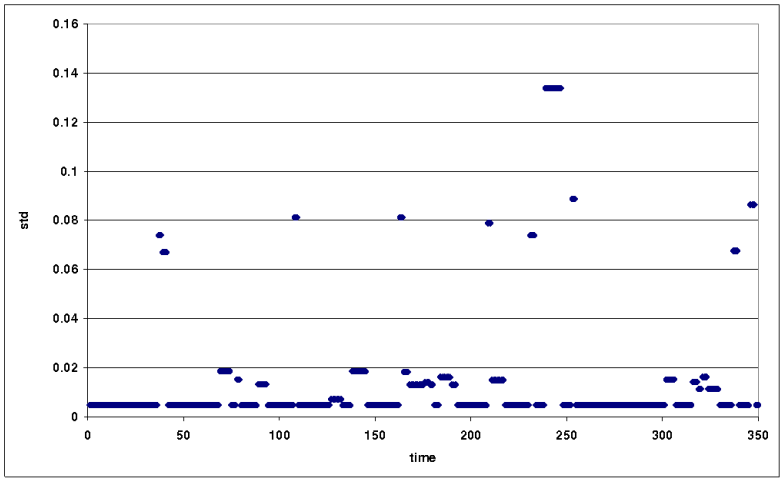


Figure 4.6: Direction Deviation UGV: Mahalanobis distance (in std units) as a function of movement time (1/10 of a second).

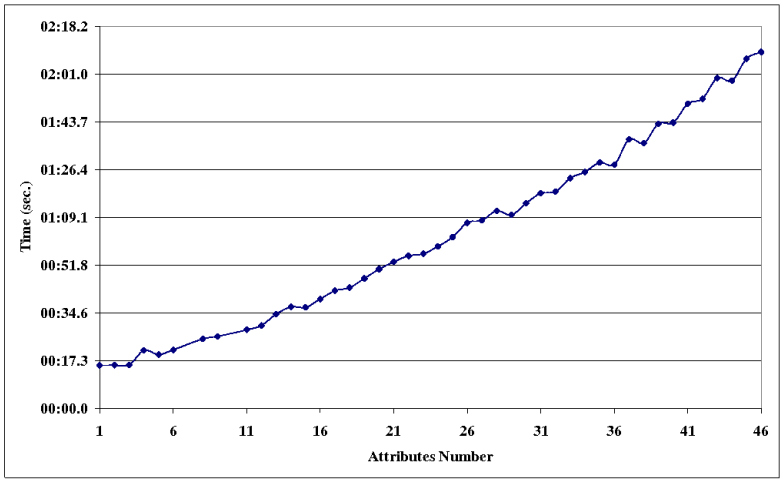


Figure 4.7: Mahalanobis distance's run-time (in minutes and seconds) as a function of attributes number.

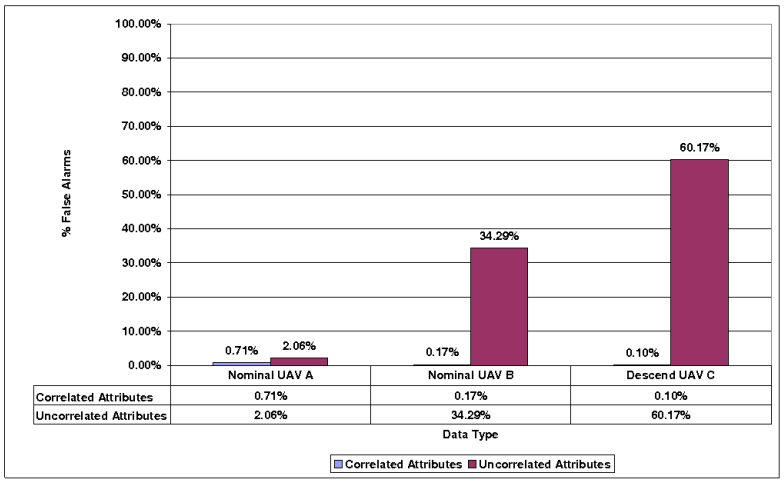


Figure 4.8: False alarm rates when applying the Mahalanobis distance on correlated and uncorrelated attributes.

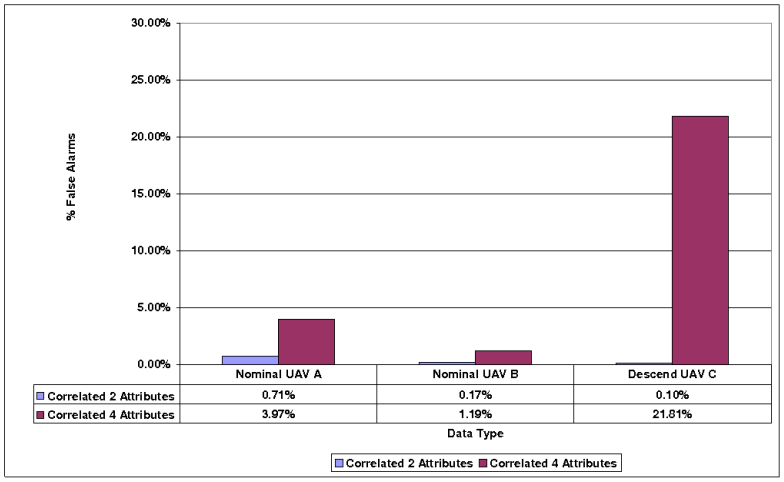


Figure 4.9: False alarm rates when applying the Mahalanobis distance on two and four correlated attributes.

Chapter 5

Online Anomaly Detection for Robots

In the last chapter we showed that a training process that selects correlated attributes for the use of the Mahalanobis Distance is essential for detecting anomalies successfully, and enables the Mahalanobis Distance to be executed online due to the dimension reduction. However, the training process in this approach is carried out offline because of its computational requirement, while we believe that online training will be a much more efficient approach, for the following reasons:

- The correlations between attributes are dynamically changed over time, as we demonstrate in section 5.3.
- This approach is (1) model free (2) domain independent (3) unsupervised and (4) completely online, making it a "plug & play" anomaly detection mechanism for different robotic platforms.
- The replacement of the MSDD algorithm with a simpler training process, allows a (computationally weak) robot to execute the approach online locally, rather than remotely, making it more autonomous. The MSDD algorithm could take much time to finish while consuming vital computational resources. Thus, it cannot be executed online, and cannot be executed on a robot. However, a simpler training process, which has a *negligible execution time*, can be executed online on a robot.

We begin by outlining our approach. Then 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.

5.1 The Outline of the Approach

We use the online input vectors to maintain the input history, which is regarded as nominal data (described in section 5.2). An online training process (described in section 5.3) uses this nominal data to return sets of correlated attributes, and a threshold per each set. The online training uses the *Pearson correlation coefficient* calculation to determine which attributes are correlated. Then, correlated attributes are grouped into sets.

Each set of correlated attributes defines a multidimensional space. For each set, multidimensional points are extracted from the nominal data; each point presents an observation of the values of the attributes in the set. These points define the nominal distribution for each multidimensional space. A threshold is calculated per each set as the highest Mahalanobis Distance of all the points in the distribution of the multidimensional space defined by the set. The highest Mahalanobis Distance belongs to the least likely observed point in the distribution. Since this point is regarded as nominal, any higher Mahalanobis Distance indicates an even less likely observation, in other words, an anomaly. Thus, the highest Mahalanobis Distance acts as a threshold.

The anomaly detector (described in section 5.5) breaks the current input vector into multidimensional points. Each point presents the values of the attributes of a different correlated set. Then, each point is compared to the nominal points of the same dimensions with Mahalanobis Distance. If the returned Mahalanobis Distance is higher then the set's threshold then an anomaly is declared.

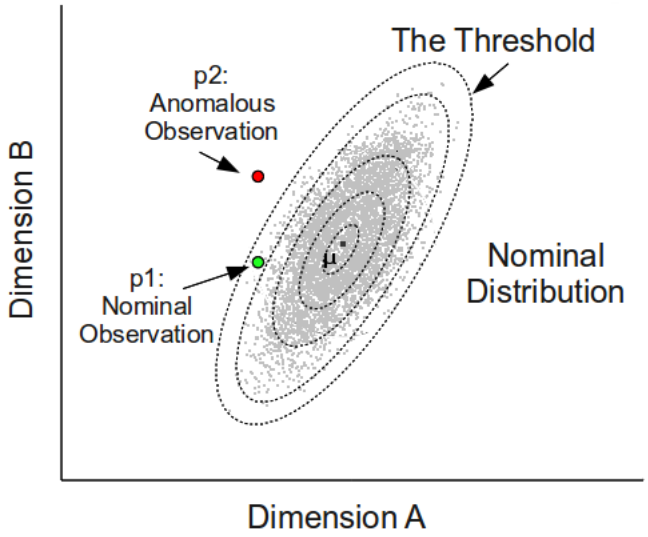


Figure 5.1: an Example of Anomalous and nominal Observations

Figure 5.1 demonstrates the work of the anomaly detector on a correlated set of two attributes. The gray (2D) points presents 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 Distance than the boundary. The outer ring is the highest Mahalanobis Distance of all the nominal points. Thus, it acts as a threshold. Points $p1$ and $p2$ have the same value in dimension A, but different values in dimension B. The Mahalanobis Distance of $p1$ is lower than the threshold. Thus it is considered as a nominal observation. The Mahalanobis Distance of $p2$ is higher than the threshold. Thus, $p2$ will cause a declaration of an anomaly.

Another aspect of the approach is the use of filtered differential data rather than raw data (described in section 5.4). The differential data indicates a change or effect, while the raw data indicates a state. The change is more fitting to the domain of robots since they act and affect the same environment that they sense and affected by. These effects are expressed in the

differential data of the robot's readings. Thus, it makes sense to use differential data in this domain. However, differential data is susceptible to noise and the high frequency of the data, which characterize this domain. Thus, a filter is used to normalize the data.

5.2 The Sliding Window technique

We utilize a *sliding window* technique [6] to maintain H, the data history, online. The sliding window (see Figure 5.2) 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* [19].

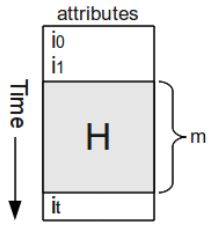


Figure 5.2:
Illustration of the sliding window.

There are several advantages for using a sliding window approach for the history 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, it is less prone to "black swan" events [28] that can lead to higher rates of false alarms. Lastly, older data is ignored and thus does not interfere with the comparison of new data. Since we consider the history H as nominal, and since H will include $\vec{i_t}$ in the next time step, it is essential that anomalies are flagged instantly. Allowing the addition of an anomalous $\vec{i_t}$ into H, reduces false positives, as it raises the thresholds described in section 5.5

5.3 Online Training

The example described in section 3.3 (the change of correlations due to axis change of the rolled UAV) 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 5.3 shows a visualization of a correlation matrix sized 71 $\times$ 71, were each 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 5.3 displays three snapshots taken from different time periods of a simulated flight, where 71 attributes were monitored. The correlation change is apparent.

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 procedure is a correlation detector (see Alg. 1) that is based on *Pearson correlation coefficient* calculation. Formally, the Pearson

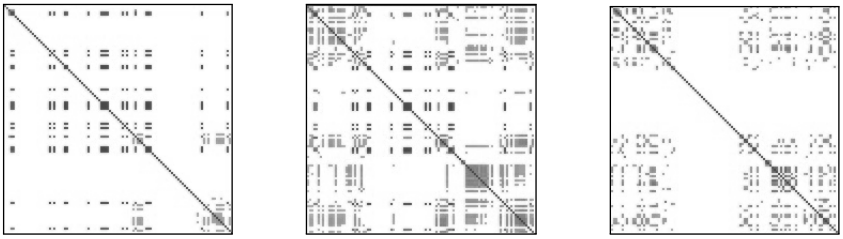


Figure 5.3: Visualization of correlation change during a flight

correlation coefficient ρ between given two vectors $\vec{X}$ and $\vec{Y}$ with averages $\bar{x}$ and $\bar{y}$, is defined as:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \tag{5.1}$$

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)

Input: H - the data history made of the last m online inputs.
Output: CS - a collection of sets of correlated attributes.

```
1: for each a_i ∈ A do
2:     CS_i ← φ
3:     for each a_j ∈ A do
4:         if |ρ_{i,j}(H_i^T, H_j^T)| > ct then
5:             add a_j to CS_i
6:     add CS_i to CS
7: 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 the ct, the 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 lowe the ct, the more attributes are considered correlated, thereby increasing the dimensions, and also increasing the likelihood of false positives, as less correlated attributes are selected. Therefore, ct should be chosen offline by repeatedly running the anomaly detection algorithm on a given data known to be nominal (e.g., a flight that had no known faults), each time increasing the value of ct until no anomalies are returned.

The second procedure sets a threshold value per each correlated set (see Alg. 2). These thresholds are later used by the *Anomaly Detector* (see Alg. 3) to declare an anomaly if the anomaly score of a given input crossed a threshold value. Each $threshold_a \in TS$ is set by

the algorithm to be the highest Mahalanobis Distance of points with dimensions relating the attributes in CS_a extracted from H. The algorithm simply calculates, for each correlated set, the Mahalanobis Distance of each point in the distribution and the highest result is set as the threshold for the correlated set. Since every point in H is considered nominal, then any higher Mahalanobis Distance indicates an anomaly.

Algorithm 2 Threshold_Setter(H, CS)

Input: H - the data history made of the last m online inputs.
Input: CS - a collection of sets of correlated attributes.
Output: TS - a collection of thresholds, one per each correlated set.

```
1: for each CS_i ∈ CS do
2:    threshold_i ← 0
3:    P_i ← points with dimensions relating to CS_i's attributes extracted from H
4:    for each p_j ∈ P_i do
5:       if threshold_i < D_mahal(p_j, P_i) then
6:          threshold_i ← D_mahal(p_j, P_i)
7:    add threshold_i to TS
8: return TS
```

One very important optimization, when using the threshold finder, is not to search for the farthest point for each correlated set with each input given, otherwise a lot of unnecessary calls for Mahalanobis Distance will be made, making it impractical. let $p_{i,max}$ be the farthest point in P_i; it sets the threshold for CS_i. With each new input, new points are added to P_i, and old ones are removed, keeping the size of P_i to m. If the anomaly detector did not find an anomaly, then this means that the new point added is closer than $p_{i,max}$ to the nominal points, so there is no need to find a new threshold yet, not until $p_{i,max}$ is removed from P_i. If the anomaly detector did find an anomaly, then the next point added to P_i is $p_{i,max}$ and its distance, already calculated, is the new threshold. This means that in the worst case (computational-wise), when there are no anomalies, a search for $p_{i,max}$ is made once every m times, making it feasible to be used on-line.

5.4 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.

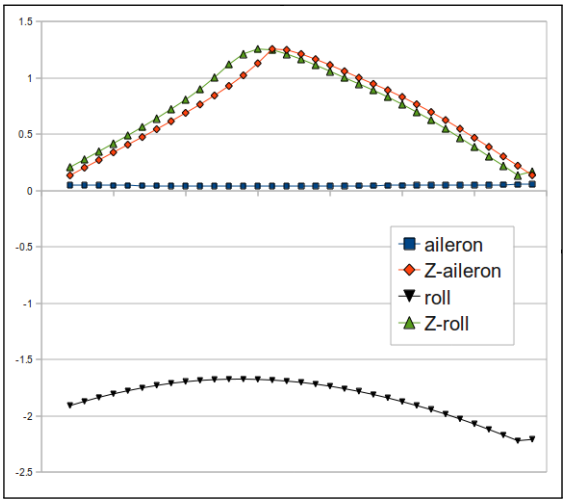


Figure 5.4: Illustration of the Z-transformation.

Figure 5.4 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 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 5.4 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 6.2

5.5 The Anomaly Detector

Algorithm 3 lists how the anomaly detector works. Each input vector that is obtained online, $\vec{i_t}$, is transformed to $Z_{\triangle}(\vec{i_t})$ (line 1). The sliding window H is updated (line 2). The *online trainer* process retrieves the *sets of correlated attributes* and their *thresholds* (line 3). For each correlated set, only the relating dimensions are considered when we compare the point extracted from $\vec{i_t}$ (line 8) to the points with the same dimensions in H (line 7). These points are compared using Mahalanobis Distance (line 9). If the distance is larger than the correlated sets' threshold, then an anomaly is declared (line 10).

Algorithm 3 Anomaly_Detector($\vec{i_t}$)

Input: $\vec{i_t}$ - the current input vector received online.

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

Chapter 6

Evaluation

First, we describe the experiments setup; the test domains and anomalies, the different anomaly detectors that emphasize the need of each one of our approach's features, and how the scoring is done. Then, we evaluate the influence of each feature of our approach, and show the advantages of our algorithm over other anomaly detection approaches.

6.1 Experiments Setup

We use four domains to test our approach, described in Table 6.1.

Domain	UAV	UGV	FlightGear	EPS
data	real	real	simulated	real
anomalies	simulated	real	simulated	real + simulated
scenarios	2	2	15	16
scenario duration (sec)	2100	96	660	120 to 300
attributes	55	25	23	81
frequency	4Hz	10Hz	4Hz	2Hz
anomalies per scenario	1	1	4 to 6	1 to 3
anomaly duration (sec)	100, 64	30	35	until the end of the input

Table 6.1: 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. 6.1). 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

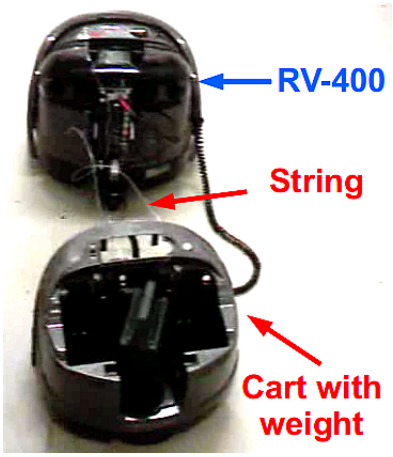


Figure 6.1: RV-400 tangled with a string connected to a heavy cart.

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 6.2: 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. 6.2). *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." [9] 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." [9]

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 [6] with respect to the UAV's behavior, and a collective anomaly [6] with respect to the measurements of different instruments such as the *GPS airspeed, altitude indicators* and the *Horizontal Situation Indicator*.

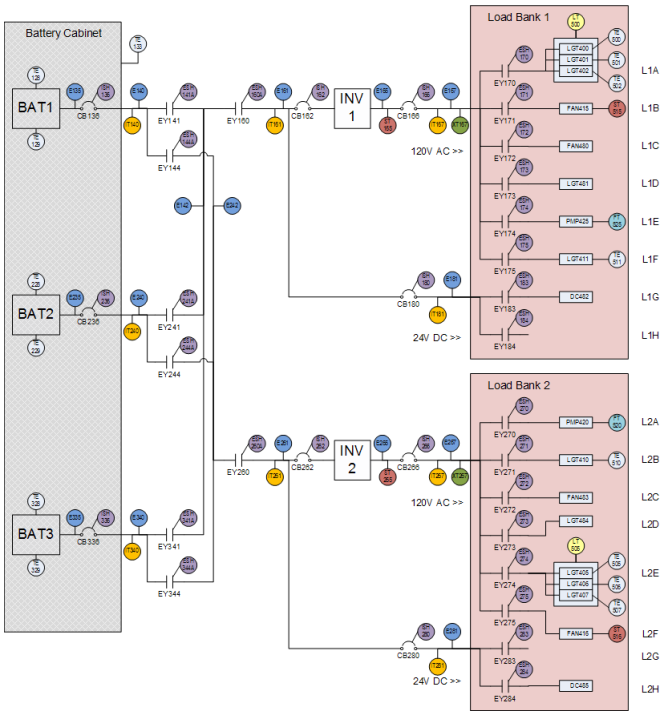


Figure 6.3: The Electrical Power System diagram.

Usually, the measured attributes of an entire robotic system are 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 the approach on a complex subsystem which is more fine grained, and where all the components effect each other, we used a fourth domain. The fourth test domain is an Electrical Power System (EPS), which simulates the functions of a typical aerospace vehicle power system (see Fig. 6.3). The data set was generated from an EPS in the Advanced Diagnostics and Prognostics Testbed (ADAPT) lab at the NASA Ames Research Center [15]. 81 attributes monitored in 2Hz; 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 [15].

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 pro-*

cess 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 6.2. The bold line is our recommended approach when using Z_Δ as the $filter$.

Name	Nominal Data	Training	Alg.
(CD,none,$filter$)	complete past data	none	4
(SW,none,$filter$)	sliding window	none	5
(CD,Tcd,$filter$)	complete past data	offline	6
(SW,Tcd,$filter$)	sliding window	offline	7
(SW,Tsw,filter)	**sliding window**	**online**	8

Table 6.2: Tested Anomaly Detectors.

The $filter$ can be *raw,* Δ*, Zraw,* Z_Δ as described in Section 5.4. CD denotes the use of a Complete record of past Data. SW denotes the use of a Sliding Window.

Here is a detailed description of each one of the algorithms we evaluated:

Algorithm 4 (CD,none,$filter$)($\vec{i_t}$)

Input: $\vec{i_t}$ - the current input vector.

1: **offline:**
2: $H \leftarrow filter(H)$
3: add A to CS
4: $threshold_H \leftarrow$ Threshold_Setter(H, CS)
5:
6: **online:**
7: $\vec{i_t} \leftarrow filter(\vec{i_t})$
8: **if** $threshold_H < D_{mahal}(\vec{i_t}, H)$ **then**
9: declare "Anomaly".

(CD,none,$filter$) operation: H is filtered offline (line 2). Since no training is done, only one "correlated" set is used (line 3), it contains all the attributes, regardless to their being correlated or not. The threshold of H is set to be the highest Mahalanobis Distance of all the points within H (line 4). $\vec{i_t}$ is filtered and compared to H online. If the threshold is crossed, an anomaly is declared.

Algorithm 5 (SW,none,$filter$)($\vec{i_t}$)

Input: $\vec{i_t}$ - the current input vector.

1: **online:**
2: $\vec{i_t} \leftarrow filter(\vec{i_t})$
3: $H \leftarrow \{\vec{i}_{t-m-1}, \ldots, \vec{i}_{t-1}\}$
4: $threshold_H \leftarrow$ Threshold_Setter($H, \{A\}$)
5: **if** $threshold_H < D_{mahal}(\vec{i_t}, H)$ **then**
6: declare "Anomaly".

(SW,none,$filter$) operation: the input vector $\vec{i_t}$ is filtered (line 2). H is set to be the last m inputs - a sliding window (line 3). The threshold is set to be the highest Mahalanobis Distance of all the points within H (line 4). Since no training is done, all the dimensions are used. $\vec{i_t}$ is compared to H (line 5). If the threshold is crossed, an anomaly is declared.

Algorithm 6 (CD,T_{CD}, $filter$)($\vec{i_t}$)

Input: $\vec{i_t}$ - the current input vector.

1: **offline:**
2: $H \leftarrow filter(H)$
3: $CS \leftarrow$ Correlation_Detector(H)
4: $TS \leftarrow$ Threshold_Setter(H, CS)
5:
6: **online:**
7: $\vec{i_t} \leftarrow filter(\vec{i_t})$
8: **for** each a ($0 \leq a \leq |CS|$) **do**
9: Let CS_a be the a'th set of correlated attributes in CS
10: Let $threshold_a$ be the a'th threshold, associated with CS_a
11: $P_H \leftarrow$ points with dimensions relating to CS_a's attributes extracted from H
12: $p_{new} \leftarrow$ point with dimensions relating to CS_a's attributes extracted from $\vec{i_t}$
13: **if** $threshold_a < D_{mahal}(p_{new}, P_H)$ **then**
14: declare "Anomaly".

(CD,T_{CD}, $filter$) operation: H is filtered offline (line 2), then a training process takes place. The Correlation_Detector uses the data in H to return the sets of correlated attributes (line 3). A threshold, per each correlated set of attributes, is set to be the highest Mahalanobis Distance of all the points within H with the dimensions relating to the attributes in the correlated set (line 4). The input vector $\vec{i_t}$ is filtered online (line 7). For each correlated set, points with the relating dimensions are extracted from $\vec{i_t}$ and H (lines 11,12), and compared with Mahalanobis Distance (line 13). If a threshold of a correlated set is crossed, then an anomaly is declared (line 14).

Algorithm 7 (SW,$T_{CD}, filter$)($\vec{i_t}$)

Input: $\vec{i_t}$ - the current input vector.

1: **offline:**
2: $H \leftarrow filter(H)$
3: $CS \leftarrow$ Correlation_Detector(H)
4:
5: **online:**
6: $\vec{i_t} \leftarrow filter(\vec{i_t})$
7: $H \leftarrow \{\vec{i}_{t-m-1}, \ldots, \vec{i}_{t-1}\}$
8: $TS \leftarrow$ Threshold_Setter(H, CS)
9: **for** each a $(0 \leq a \leq |CS|)$ **do**
10: Let CS_a be the a'th set of correlated attributes in CS
11: Let $threshold_a$ be the a'th threshold, associated with CS_a
12: $P_H \leftarrow$ points with dimensions relating to CS_a's attributes extracted from H
13: $p_{new} \leftarrow$ point with dimensions relating to CS_a's attributes extracted from $\vec{i_t}$
14: **if** $threshold_a < D_{mahal}(p_{new}, P_H)$ **then**
15: declare "Anomaly".

(SW,$T_{CD}, filter$) operation: H is filtered offline (line 2), then the Correlation_Detector uses the data in H to return the sets of correlated attributes (line 3). The input vector $\vec{i_t}$ is filtered online (line 6) and H is set to be the last m inputs - a sliding window (line 7). A threshold, per each correlated set of attributes, is set to be the highest Mahalanobis Distance of all the points within H with the dimensions relating to the attributes in the correlated set (line 8). For each correlated set, points with the relating dimensions are extracted from $\vec{i_t}$ and H (lines 12,13), and compared with Mahalanobis Distance (line 14). If the threshold of the correlated set is crossed, then an anomaly is declared (line 15).

Algorithm 8 (SW,$T_{SW}, filter$)($\vec{i_t}$)

Input: $\vec{i_t}$ - the current input vector.

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

(SW,$T_{SW}, filter$) is our proposed approach when the filter is Z_Δ (see Alg. 3 in section 5.5). Everything is done online. H is set to be the last m inputs - a sliding window (line 2), the online training uses the data in H to return the sets of the correlated attributes and their thresholds (line 3). For each correlated set, points with the relating dimensions are extracted from $\vec{i_t}$ and H (lines

7,8), and compared with Mahalanobis Distance (line 9). If the threshold of the correlated set is crossed, then an anomaly is declared (line 10).

(CD,Tsw,$filter$) is *not* displayed in table 6.2. 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, it is not compared to the other anomaly detectors displayed in table 6.2.

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 6.3 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 6.3: 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.

6.2 Results

Figures 6.4 and 6.5 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.

We begin with the first anomaly detector, (CD,none). Both Figures 6.4 and 6.5 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 6.5. However, the threshold is not crossed even when anomalies occur, resulting in a very low detection rate as Figure 6.4 shows. The reason is the absence of training. The Mahalanobis

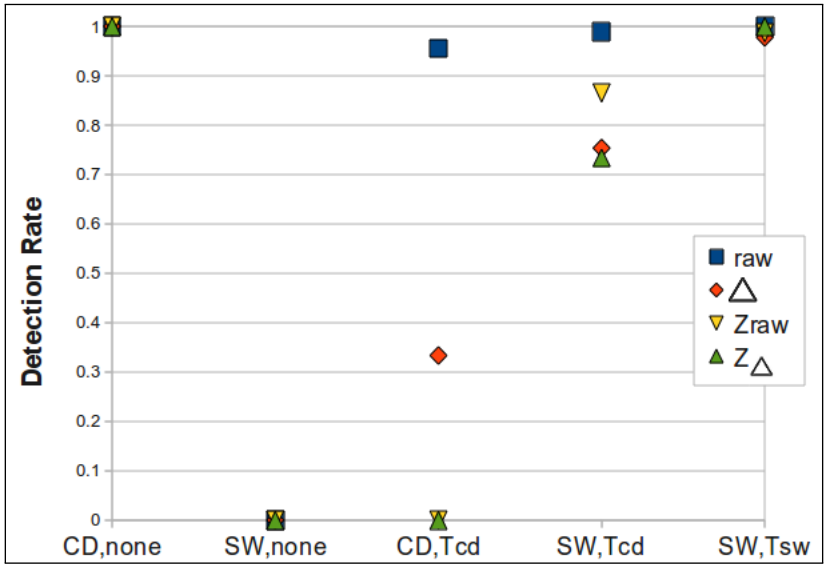


Figure 6.4: Detection rate. (Higher is better)

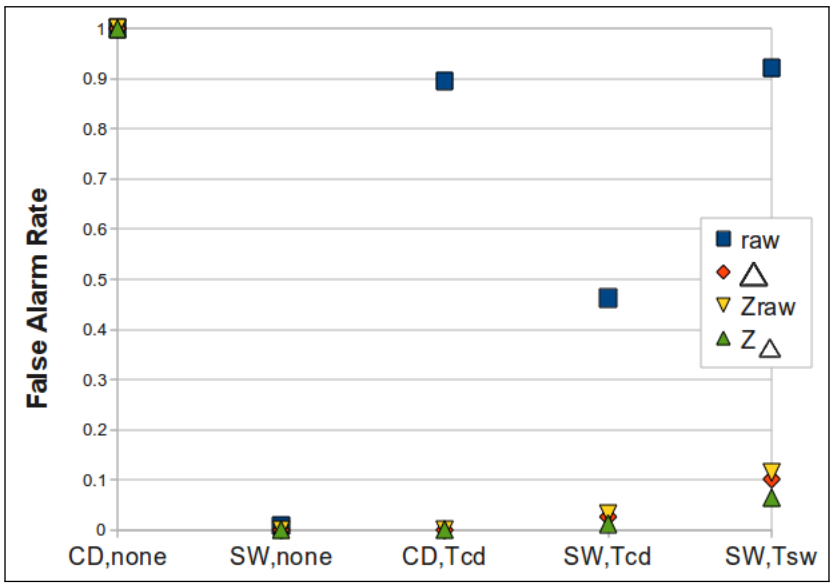


Figure 6.5: False alarm Rate. (Lower is better)

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 6.4 and 6.5. 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_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 to nearly 0. (SW,Tsw,Z_Δ), which is the approach we described in section 5.5, 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 6.4

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 6.4: Feature Contributions

Figure 6.6 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 6.6 illustrates that when the features of our approach are applied, they allow the results to approximate the perfect classifier.

Figure 6.7 shows the detection rates and false alarm rates of (TW,Tsw,Z_Δ) in the classifier space, when we increase the correlation threshold $ct \in \{0..1\}$ in the online trainer described in section 5.3. Note that the X axis scales differently than in Figure 6.6, 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 [26]. However, the SVM algorithm classifies every input as nominal, including all anomalies, resulting in a detection rate of 0 as Figure 6.8 shows. Samples of both categories are provided to the SVM, and it is an offline process, yet, the contextual and collective anomalies are undetected.

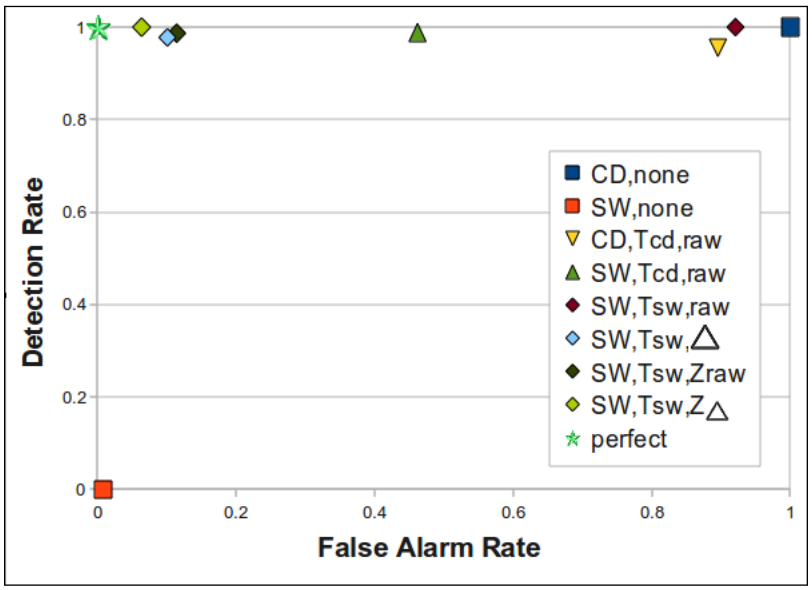


Figure 6.6: The classifier plane.

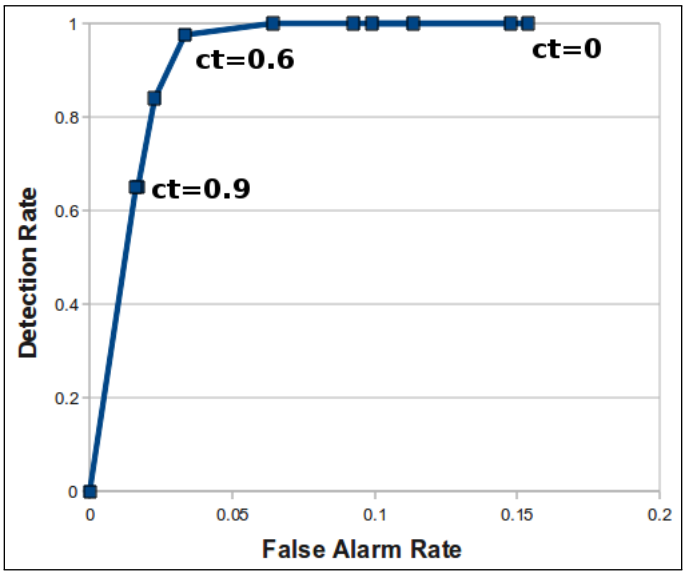


Figure 6.7: The influence of the correlation threshold.

This goes to show how illusive these anomalies are, which were undetected by a successful and well-known classifier, even under unrealistic favoring conditions.

We also examine the quality of (SW,Tsw,$Z_{\triangle}$) in the context of other anomaly detectors.

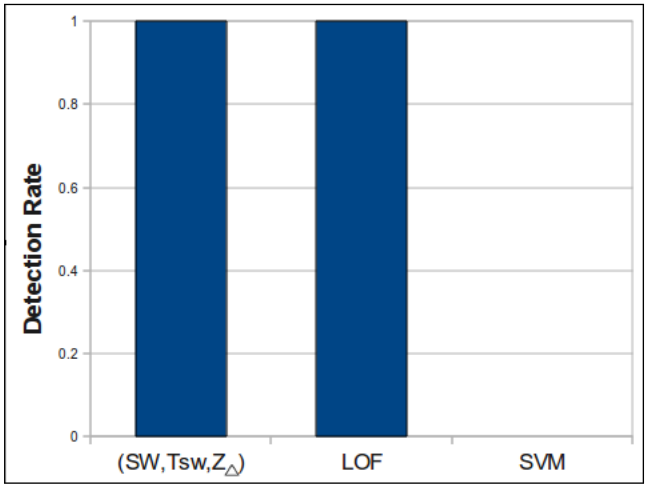


Figure 6.8: FlightGear Domain Detection Rate

We compared it to the *incremental LOF algorithm* [22]. 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 [22]. Figure 6.8 shows a detection rate of 1 to (SW,Tsw,$Z_{\triangle}$) 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 6.9 to 6.13 show for every tested domain the false alarm rate of

1. (SW,Tsw,$Z_{\triangle}$)
2. optimized (SW,Tsw,$Z_{\triangle}$) denoted as OPT(SW,Tsw,$Z_{\triangle}$)
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,$Z_{\triangle}$) 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,$Z_{\triangle}$) to OPT(LOF) does indicate which approach is better, since both are optimized. The comparison between OPT(SW,Tsw,$Z_{\triangle}$) to (SW,Tsw,$Z_{\triangle}$) indicates how better (SW,Tsw,$Z_{\triangle}$) can theoretically get.

In all the domains the OPT(SW,Tsw,$Z_{\triangle}$) had the lowest false alarm rate. Naturally, OPT(SW,Tsw,$Z_{\triangle}$) has a lower false alarm rate than (SW,Tsw,$Z_{\triangle}$), 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 robotic domains, the highest false alarm rate of (SW,Tsw,$Z_{\triangle}$) occurred in the UAV's second flight, as Figure 6.11 show (little above 0.09). In this flight, the fault

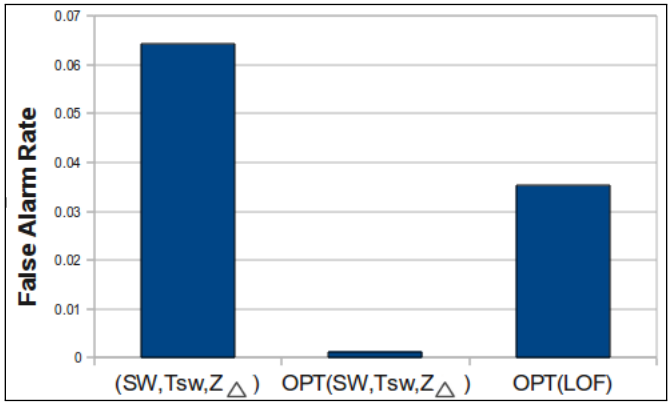


Figure 6.9: FlightGear domain.

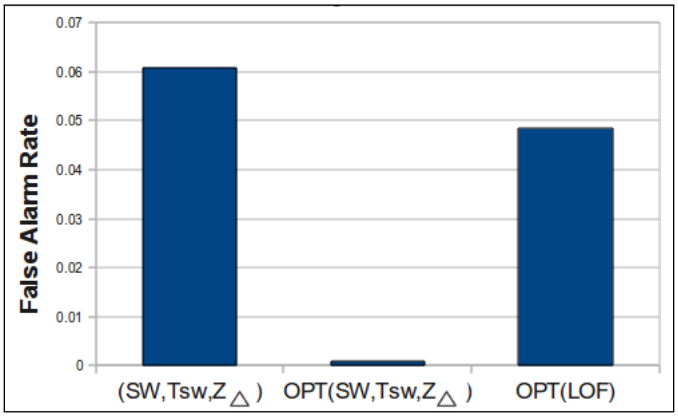


Figure 6.10: UAV first flight.

occurred in an attribute that is not very correlated 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 6.12 shows 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 [6]. 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 6.14 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

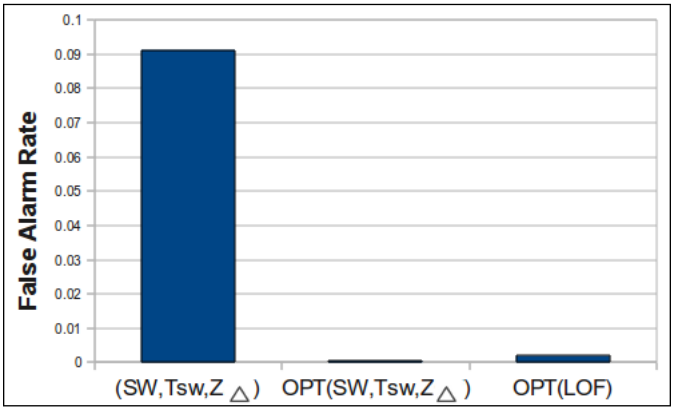


Figure 6.11: UAV second flight.

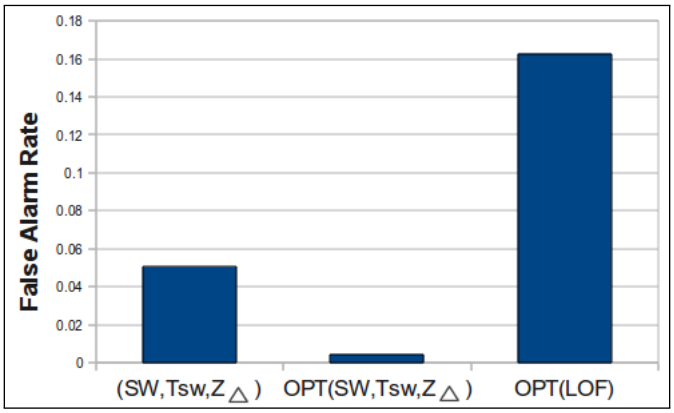


Figure 6.12: UGV domain.

optimized false alarm rate.

In this thesis, we presented two approaches. The first approach uses an offline training procedure, utilizing a complex and computationally heavy dependency detection algorithm. The second approach uses an online training procedure utilizing the quick *Pearson correlation* calculation. Now, we want to compare the two approaches, and show that the second approach is more effective.

We begin by run-time comparison. In the first approach, the run-time of the online anomaly detection phase is predetermined by the offline training phase. However, in the second approach, the run-time is influenced by parameters which are set by the user. The second approach has additional computation due to its online training. However, the Mahalanobis Distance is used to compare the input vector to a "window" of the last m inputs. Thus, the run-time is influenced by the size of m. In the first approach, where the training is done offline, the Mahalanobis Distance is used to compare the input vector to a complete recored of past data, which can be significantly larger than the online window. Thus, despite the additional computational time of the online training, the second approach's run-time can be set to be equal or even smaller than the online

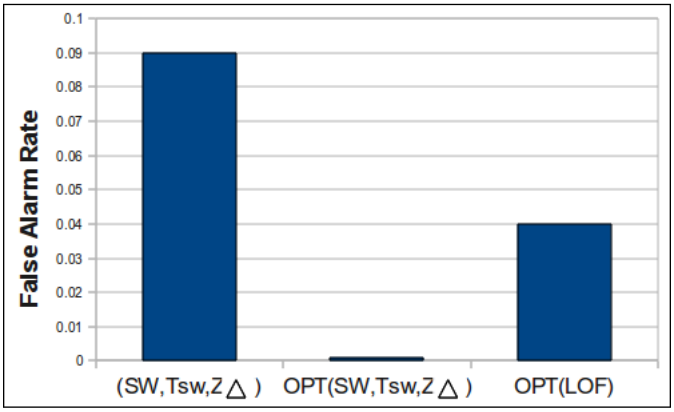


Figure 6.13: EPS domain.

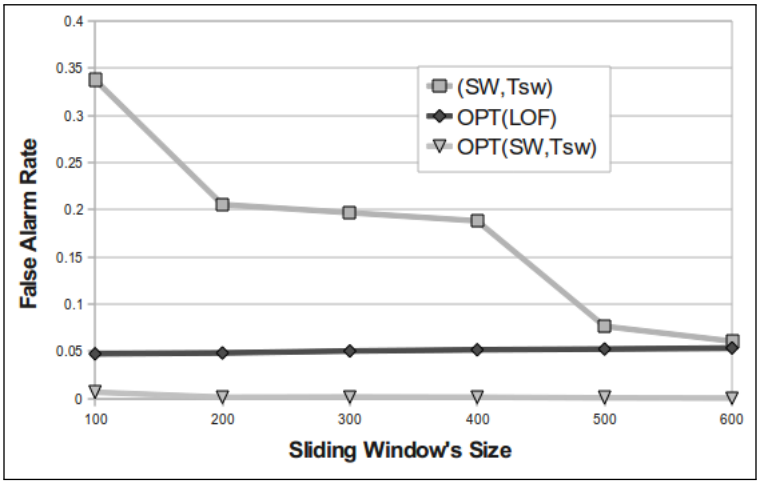


Figure 6.14: Sliding Window's changing size.

run-time of the first approach (which is usually the case).

While the second approach has no offline phase, the first 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 *days* and even more. The run-time of the online training of the second approach, scales to the number of attributes being measured; its 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 training to be slower than the frequency of the input, then the data can be sampled in a lower frequency.

The most interesting comparison, obviously, is the false alarm rates of each approach. Each approach has advantages and disadvantages that effects the results, as summarized in table 6.5

Figure 6.15 summarize the results of the proposed approach (second approach) compared against the first approach. Since in the first approach only returns an anomaly score, optimized

	offline training approach	**online training approach**
Advantages	a better algorithm to find correlations.	new data is compared to current data. the data is filtered. automatic selection of correlated attributes.
Disadvantages	attributes are *manually* selected into correlated sets. new data is compared to past data. the data is not filtered.	a simpler algorithm to find correlations.

Table 6.5: Feature Contributions

thresholds were selected by the OPT algorithm to flag anomalies. Thus the optimal results are displayed. Except for the UGV domain, where there is a slight difference in the false alarm rates, in all the others the second approach outperforms the first. In the FlightGear and EPS domains the first approach failed. every input was declared as anomalous. This goes to show the problem of manually selecting correlated attributes out of the MSDD output.

	Optimized first approach		**Optimized second approach**	
domain	detection rate	false alarm rate	detection rate	false alarm rate
UAV	1	0.0044	1	0.0014
UGV	1	0	1	0.005
FlightGear	1	1	1	0.0013
EPS	1	1	1	0.001

Figure 6.15: Comparison of first and second approach

To know the influence of our appraoch's features, upon another density based technique, we also implemented the *incremental LOF algorithm* with our features. Figure 6.16 shows the decrease of false positives, of $OPT(LOF)$, when the filters were used, averaged over the 15 flights of the *FlightGear*'s domain. While the raw data produced an average of 63 false positives per flight, *diff* produced only a half, *Zraw* and *Zdiff* produced an average of 4.53, 4.5 false positives per flight respectively. The last results are very close to the optimal results of our approach which had only 3 false positives. This goes to show the special need of the robots domain, in a differential filtered data.

We also tested our approach, with the replacement of the Mahalanobis Distance calculation with the incremental LOF calculation. This means that an online trainer grouped correlated attributes into sets, and the *incremental LOF algorithm* was applied to each correlated set, while maintaining points from the sliding window. However, this was proved to have a very long computational time, which is not feasible online.

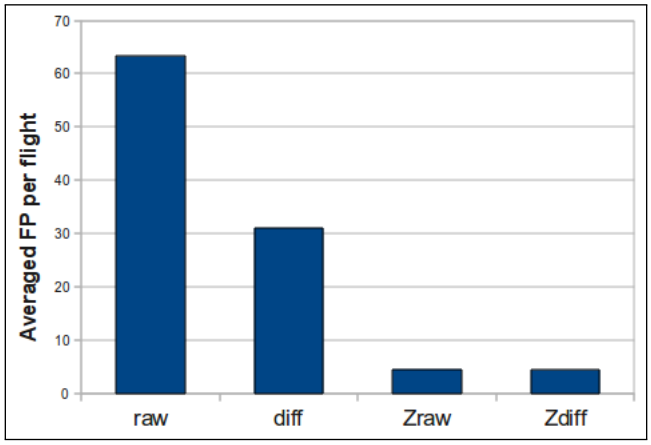


Figure 6.16: False positives of inc. *LOF*

Chapter 7

Conclusions and Future Work

In this thesis we presented two novel approaches for detecting anomalies in unmanned autonomous vehicles. Both approaches use the Mahalanobis Distance to detect anomalies and thus benefit from its model-free nature. The experimentation with different domains of both simulated and physical nature, showed the approaches' domain independent quality and the ability to succeed in the real world. While the first approach used an offline training procedure, the second approach used an online training procedure that made the approach also unsupervised, and completely online anomaly detector for robots. Therefore, the second approach has the qualities of a "plug & play" mechanism for different robotic platforms. We showed how essential a training process is to the successful online anomaly detection. Moreover, the experiments with the second approach also showed the benefits of:

- The comparison of the current data input to the data of the sliding window.
- The finding of dynamic correlations between attributes (online).
- The filtering of the data.

We showed that this approach is superior to previous approaches such as the incremental LOF algorithm, and that the approach succeeds where other well-known classifiers, such as SVM, have failed even under unrealistic favoring conditions. We compared our two approaches, and found that the online approach is more effective than the first. Finally, we showed that the filtering can improve other anomaly detection techniques, thereby showing its independent contribution.

The next step after anomaly detection towards diagnosis, is "Anomaly Isolation". We think that once an anomaly was declared, the most likely attributes that are expressing the anomaly can be easily found by a process of elimination. If the removal of a dimension lowers the Mahalanobis Distance then the attribute relating this dimension is probably responsible for the anomaly. Once the isolated symptoms are collected, a process of diagnosis can begin.

The large difference between the results of the optimized thresholds to the results of the calculated thresholds, shows that the thresholds chosen by our technique can be chosen better. We think that better learning algorithms can find closer thresholds to the optimized ones.

This work focused on detecting anomalies on a single robot. We think that our approach can work just as well in the data of multiple robots. For example, a few UAV's are flying in a formation, the data of the altitude of each aircraft is added to a multidimensional point. When one aircraft breaks the formation unexpectedly, the Mahalanobis Distance will raise indicating an anomaly.

Bibliography

[1] Noa Agmon, Sarit Kraus, and Gal A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2339–2345, 2008.

[2] T. Ahmed, M. Coates, and A. Lakhina. Multivariate online anomaly detection using kernel recursive least squares. In *IEEE INFOCOM*, pages 625–633, 2007.

[3] Tarem Ahmed, Boris Oreshkin, and Mark Coates. Machine learning approaches to network anomaly detection. In *the Second Workshop on Tackling Computer Systems Problems with Machine Learning*, 2007.

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

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

[6] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *the Association for Computing Machinery, Computing Surveys*, 41(3):1–58, 2009.

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

[8] M. Daigle, X. Koutsoukos, and G. Biswas. A qualitative event-based approach to continuous systems diagnosis. *IEEE Transactions on Control Systems Technology*, 17(4):780–793, 2009.

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

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

[11] Puneet Goel, Göksel Dedeoglu, Stergio I. Roumeliotis, and Gaurav S. Sukhatme. Fault-detection and identification in a mobile robot using multiple model estimateion and neural network. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[12] Michael A. Goodrich, Bryan S. Morse, Damon Gerhardt, Joseph L. Cooper, Morgan Quigley, Julie A. Adams, and Curtis Humphrey. Supporting wilderness search and rescue using a camera-equipped mini UAV. *Journal of Field Robotics*, pages 89–110, 2008.

[13] Michael Hofbaur, Johannes Köb, Gerald Steinbauer, and Franz Wotawa. Improving robustness of mobile robots using model-based reasoning. *Journal of Intelligent and Robotic Systems*, 48(1):37–54, 2007.

[14] R. Murphy J. Craighead and B. Goldiez J. Burke. A survey of commercial open source unmanned vehicle simulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 852–857, 2007.

[15] ADAPT lab. Website, 2007. `https://c3.ndc.nasa.gov/dl/data/adapt-an-electrical-power-system-testbed/`.

[16] Jorma Laurikkala, Martti Juhola, and Erna Kentala. Informal identification of outliers in medical data. In *Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology*. 2000.

[17] Raz Lin, Eliyahu Khalastchi, and Gal A. Kaminka. Detecting anomalies in unmanned vehicles using the mahalanobis distance. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3038–3044, 2010.

[18] Thomas Lotze, Galit Shmueli, Sean Murphy, and Howard Burkom. A wavelet-based anomaly detector for early detection of disease outbreaks. In *International Conference on Machine Learning*, 2006.

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

[20] Larry M. Manevitz and Malik Yousef. One-class svms for document classification. *Journal of Machine Learning Research*, 2:139–154, March 2002.

[21] Tim Oates, Matthew D. Schmill, Dawn E. Gregory, and Paul R. Cohen. Learning from data: Artificial intelligence and statistics. chapter Detecting Complex Dependencies in Categorical Data, pages 185–195. Springer Verlag, 1995.

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

[23] Gunnar Rätsch, Sebastian Mika, Bernhard Schölkopf, and Klaus-Robert Müller. Constructing boosting algorithms from svms: An application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1184–1199, 2002.

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

[25] Gerald Steinbauer, Martin Mörth, and Franz Wotawa. Real-time diagnosis and repair of faults of robot control software. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 13–23. 2006.

[26] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer-Verlag, 2008.

[27] Paul Sundvall and Patric Jensfelt. Fault detection for mobile robots using redundant positioning systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3781–3786, 2006.

[28] Nassim Nicholas Taleb. *The Black Swan: The Impact of the Highly Improbable*. Random House, 2007.

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

[30] T. D. Wickens. *Multiway Contingency Tables Analysis for the Social Sciences*. Lawrence Erlbaum Associates, 1989.