Dyd^2: Dynamic Double anomaly Detection

Adrien Dorise, Louise Travé-Massuyès, Audine Subias, Corinne Alonso

DyD^2: Dynamic Double anomaly Detection

Application to on-board space radiation faults

Adrien Dorise[*,**], **Louise Travé-Massuyès**[*], **Audine Subias**[*], **Corinne Alonso**[*]

[*] *LAAS-CNRS, Université de Toulouse, CNRS, INSA, UPS, Toulouse, France (adorise,louise,subias,alonsoc@laas.fr)*
[**] *CNES, 18 avenue Edouard Belin 31401 Toulouse, France*

Abstract: Anomaly detection is a crucial aspect of embedded applications. However, limited computational power, evolving environments, and lack of training data are difficulties that can limit anomaly detection algorithms. One class classification algorithms are often used for this task to circumvent the need of anomalous data in the training set. This paper presents a new machine learning algorithm for anomaly detection called Dynamic Double anomaly Detection DyD^{2}that is suited to evolving environments and on-board requirements. The contributions made by DyD^2 are thoroughly presented and an experimental evaluation is set up to compare DyD^2 to state-of-the-art algorithms.

Keywords: Anomaly detection, Machine learning, One-class classification, Embedded applications, Aerospace engineering, Space Radiations

1. INTRODUCTION

In the last decades, computing power has increased exponentially. Following Moore's law, node technology has met an incredible improvement in recent years. From 90nm in 2003, Intel recently announced a new 20A technology coming in 2024, enabling industries to craft materials at the atomic level (Pat Gelsinger (2021), Etiemble (2018)). These breakthroughs led to algorithm performance enhancement, and machine learning is no exception. More complex and computationally expensive algorithms are developed in order to address various issues. However, not all fields have access to significant computation power. For instance, due to harsh radiative conditions, the space industry depends on hardened and reliable components at the cost of performance. Embedded applications have to balance power efficiency and performance (Banbury et al. (2020)).

Moreover, detecting and reacting to any fault that could endanger the mission is essential for embedded systems evolving in real-time. Thus, anomaly detection takes a significant role in any embedded applications. Machine learning algorithms are widely used as anomaly detection tools, and training sets are used to find patterns that do not conform to expected behavior (Chandola et al. (2009)). In this regard, data quantity and quality are closely related to anomaly detection performances. Many detection methods require examples of both normal and anomalous data during the training phase to perform the detection. However, in some cases, it can be complex to access anomalous data. In this context, multi-class detection cannot be performed, and other solutions must be explored. *One Class Classification (OCC)* is a particular case of classification where only a single class is observed during training (Perera et al. (2021); Khan and Madden (2014)). OCC has become very popular in anomaly detection as it is possible to train the algorithm using only normal data (Fuertes et al. (2016); Chen et al. (2017)).

⋆ The thesis is co-funded by CNES and Région Occitanie. This project is supported by ANITI through the French "Investing for the Future – PIA3" program under the Grant agreement noANR-19-PI3A-0004.

In an ideal world, training data would be representative of the data encountered during online stages, resulting in the descriptive statistics of both sets being similar. However, embedded systems can be subject to multiple variations because they are evolving in changing environments. Environmental modifications, ageing, or unexpected situations can create a deviation between the trained model and online data. Thus, the model shall adapt to its new environment. In this regard, algorithms have been developed to perform anomaly detection in evolving environments (Barbosa Roa et al. (2019)).

This paper proposes a new anomaly detection method based on one class classification called Dynamic Double anomaly Detection (DyD^2), whose novelty is to be suitable to embedded systems evolving through time. First, a description of the algorithm is proposed in section 2. Then a case study is described in section 3. Finally, the results and a comparison with state-of-the-art OCC methods are proposed in section 4.

2. THE DyD^{2}ALGORITHM

2.1 Principles of DyD^2

The DyD^2 algorithm is a dynamic double anomaly detection method designed to fulfil on-board requirements, particularly low computational cost. The goal is to efficiently detect several types of anomalies in data streams composed of multiple time series. The general idea is to first train a model offline with normal data. Then the data stream is checked against the model during the online detection phase. Taking inspiration from many clustering techniques for data streams (Barbosa Roa et al. (2019); Hyde et al. (2017)), the model is composed of μ-clustersthat group together data points according to a distance-based criterion. In DyD^2, this principle is used for the two detection phases of the algorithm with different features to characterise data points:

- The first detection phase aims at detecting critical and heavily out of distribution anomalies. It is why it

takes raw measured quantities as features. Such features do not require preprocessing, hence promoting speed. These features are called *outer features*, the model learned through this phase is called the *outer map*, and the detected anomalies are qualified as *outer anomalies*.
- The second detection phase aims at detecting subtle anomalies that require in depth analysis. Relevant features are extracted in well-chosen time windows. These features are called *inner features*, the model learned during this phase is called the *inner map*, and the detected anomalies are qualified as *inner anomalies*. This phase is slower than the first phase and assumes that this type of anomaly is not time-critical.

Anomaly detection is approached as a *one-class classification* problem: only normal data are mandatory to train the outer and inner detection maps. Consequently, it is easier to learn a model. However, the downside is that the user must ensure that no anomalies are present in the training set, as faulty behaviour could be learned, leading to false-negative results.

The DyD^2 algorithm manages objects called *samples* and *μ-clusters:*

Definition 1. (Sample) A sample S is characterised by a couple (F_S, t_S) where F_S is a feature vector and a t_S is a date.

DyD^2 makes use of two types of samples. The *point sample* S_p is a vector of features coming from time series values of a given time t. The *window sample* S_w is a vector of features extracted from a window of time series values starting of a given time t.

Definition 2. (μ-cluster) A μ-cluster μCl_k is defined by a characteristic vector CF_k of the following form:

$$CF_k = (n_k, C_k, tc_k, tu_k) \tag{1}$$

where $n_k \in \mathbb{N}$ is the number of samples in the μ-cluster, $C_k \in \mathbb{R}^+$ is a vector containing the coordinates of the μ-cluster center, $tc_k \in \mathbb{R}^+$ the creation time of the μ-cluster, $tu_k \in \mathbb{R}^+$ is the time of the last update of the μ-cluster.

The initialisation of a μ-cluster is performed using a sample S. The feature vector of S is used as coordinates for the μ-cluster center C_k. Also, the date t of S is used for both tc_k and tu_k.

Definition 3. (Detection map) A detection map M is composed of a set of learned μ-clusters of size $s_M \in (R^+)^m$, where $m \in \mathbb{N}$ is the dimension of the considered space. A detection map models normal behaviour.

DyD^2 makes use of two detection maps. The *outer map*, denoted M_{out}, is created using point samples S_p. The *inner map*, denoted M_{in}, works with window samples S_w. Detection maps are dynamic objects in the sense that μ-cluster positions in the dimensional space defined by a set of features are adjusted depending on incoming data. Therefore, detection maps are hence able to follow data evolution that must not be considered anomalous (ageing or environmental modifications), hence allowing dynamic detection.

DyD^2 is developed as an on-board application, taking into consideration computing limitations. Low memory requirements and fast response time are key elements in on-board applications. The use of μ-clusters avoids the need to save all incoming data points by grouping them. Doing so, it is possible to work on significantly fewer objects, thus saving computation time and memory space.

Moreover, prior to the two detection phases, streaming data is processed with a change point detection method that localizes potential anomalies in time series. This analysis is critical to DyD^2 efficiency because it allows not to consider each sample as a possible anomaly and significantly improves the algorithm reaction time.

An essential notion in DyD^2 is *reachability* (Barbosa Roa et al. (2019)). A μ-cluster μCl_k is reachable by a sample S if S is located inside the volume of μCl_k defined by its size s_k:

$$distance(C_k, S) \preceq \frac{s_k}{2} \tag{2}$$

where $\preceq$ is the dimension-by-dimension $\leq$ relation.

By extension, a map M is reachable by a sample S if at least one μ-cluster of M is reachable by S. The Manhattan metric is used for the distance function as it performs better for high dimensions than the Euclidean distance (Aggarwal et al. (2001)).

In the next two subsections, a thorough description of DyD^2 is done according to the flow chart of Fig. 1. The different steps are identified by circled numbers referenced in the corresponding sections and paragraphs.

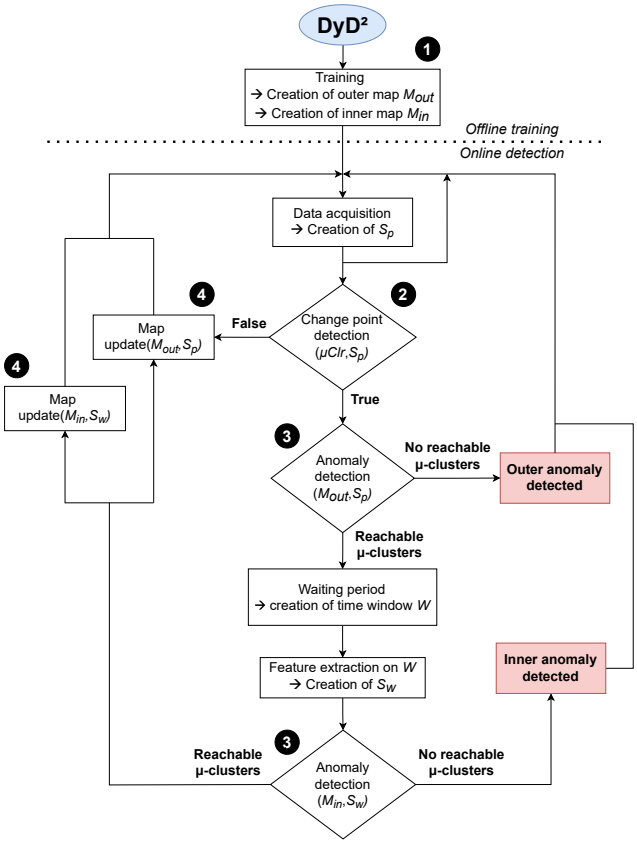

Fig. 1. DyD^2 flow chart

2.2 *Offline training* 1

DyD^2 training is performed offline. As a one class classification method, DyD^2 only needs normal data for the training phase.Therefore, the nominal behaviour of the system is learned from historical normal data streams and stored in the detection maps M_{out} and M_{in}.
The training phase is detailed in Algorithm 1. First, a time window W of fixed size is defined (lines 3 and 4).

W is used to create S_w and acts as a short term memory that stores recent points and moves through the data set. For each new incoming data point, the oldest one is erased (line 7). Given the incoming data stream, point samples S_p and window samples S_w are created (lines 8, 14, 15) to be checked for reachability against the μ-clusters of M_{out} and M_{in} respectively (lines 9, 16). If no μ-cluster of the detection map is reachable by the sample, a new μ-cluster is created using the sample characteristics (lines 10, 16). Otherwise, reachable μ-clusters are updated according to the sample feature vector (lines 12, 18).

Algorithm 1: Training

```
1  input: training data set X
2  output: inner map M_in and outer map M_out
3  for i ∈ [0, timewindowsize[ do
4  |  W_i ← X_i ;
5  end
6  foreach data point X_i (with i >= time window
   size) do
7  |  add X_i to W_i and remove W_0 ;
8  |  S_p ← (X_i, date) ;
9  |  if reachable(M_out, S_p) = false then
10 |  |  M_out ← initμCluster(M_out, S_p) ;
11 |  else
12 |  |  M_out ← update(M_out, S_p) ;
13 |  end
14 |  S_w ← featureExtraction(W) ;
15 |  if reachable(M_in, S_w) = false then
16 |  |  M_in ← initμCluster(M_in, S_w) ;
17 |  else
18 |  |  M_in ← update(M_in, S_w) ;
19 |  end
20 end
21 return M_out, M_in
```

2.3 Online detection

When offline training is finalised and the two detection maps M_{in} and M_{out} have been created, DyD^2 can run onboard to detect anomalies on the fly based on the incoming data stream.

Change point detection ❷ Aminikhanghahi and Cook (2017) define a *change point* in a time series as a "transition between different states in the process that generates the time series data". In the considered application, anomalies are all change points. Hence, the first stage of DyD^2 is a change point detection to quickly exclude non-anomalous data from further processing, thus saving computation time. Algorithm 2 describes the change point detection method used in DyD^2. Change point detection is performed using a single μ-cluster called *rupture μ-cluster* and denoted μCl_r. Unlike the μ-clusters of M_{out} and M_{in}, μCl_r is given a specific size $s_r \in (R^+)^{m_{out}}$ and it is created during the online phase of the algorithm (line 4). A point sample S_p is created with each arriving data point in the data stream, and the first is used to initialise μCl_r. A change point is identified when μCl_r is not reachable by some S_p (line 6). In that case, a new rupture μ-cluster μCl_r is created using S_p (lines 7,8). Otherwise, μCl_r is updated using S_p.

Double anomaly detection ❸ As said in section 2.1, DyD^2 includes two detection phases in order to distinguish critical from non-critical anomalies. For the first phase, the outer map M_{out} is used with outer features directly given by the coordinates of data point samples S_p. This detection phase targets time-critical anomalies that are heavily out of the normal distribution. For the second phase, the inner map M_{in} is used along the window W from which an elaborate feature extraction process is performed to obtain window samples S_w. This feature extraction process is key to detect complex anomalies. The features to be extracted from the data window are left to the user, as it depends on each application.

Algorithm 2: Change point detection

```
1  input: rupture μ-cluster μCl_r and point sample S_p
2  output: true if a change point is detected, false
   otherwise
3  if μCl_r does not exist then
4  |  initialise μCl_r with S_p ;
5  else
6  |  if reachable(μCl_r, S_p)=false then
7  |  |  destroy μCl_r ;
8  |  |  initialise new μCl_r using S_p ;
9  |  |  return true ;
10 |  else
11 |  |  update μCl_r using S_p ;
12 |  |  return false ;
13 |  end
14 end
```

Double anomaly detection is performed by checking in sequence S_p and S_w for reachability against the μ-clusters of the detection maps M_{out} and M_{in} respectively. A critical *outer anomaly* is detected if no μ-cluster of M_{out} is reachable by S_p. If this is not the case, detection goes on with S_w with respect to M_{in}. An *inner anomaly* is detected if no μ-cluster of M_{in} is reachable by S_w.

Update map ❹ This step reflects the dynamic aspect of the DyD^2 method. It allows the integration of new knowledge about normal behaviour in the models used for anomaly detection by updating the detection maps M_{out} and M_{in} with samples that have been identified as normal. The update process is described in Algorithm 3. In this process, the characteristic vectors of μ-clusters are updated, modelling in particular μ-cluster center displacement and the μ-cluster ageing.

- *Center displacement* of μ-clusters is only performed on reachable μ-clusters in the map (line 3). Given a μ-cluster μCl_k, the displacement of its center is weighted by the number of samples n_k already present. The higher the amount of samples inside μCl_k, the smaller the displacement due to the integration of a new sample S. The new μ-cluster center C'_k is given by the following formula to be understood as the center of mass:

$$C'_k = \frac{S + n_k C_k}{1 + n_k} \tag{3}$$

  μ-cluster centers update is performed in line 5.

  The number of samples of the μ-cluster and the last update time are also updated (lines 7,8).
- *μ-clusters ageing* is performed to prioritise the most recent samples and forget those observed in the past. Indeed, mapping a dynamic behaviour means that the newest information is more representative of the system's current behaviour. For a μ-cluster μCl_k of a map, the last update time tu_k is compared to the date of the last to arrive sample S (line 10). If the difference is higher than a fixed threshold, a penalty is applied to n_k (line 11). By doing so, old μ-clusters are

the ones that are the most impacted by updates. The penalty is a linear decrease and is written as follows:

$$n'_k = n_k * penalty \quad (4)$$

where n'_k is the new number of samples in μCl_k after applying the penalty.

Algorithm 3: Update map

```
input: Detection map M and sample S
foreach k μ-clusters μCl_K in M do
    if μCl_k reachable by S then
        foreach dimension m_k do
            update C_k using eq (3) ;
        end
        n_k + + ;
        tu_k ← t_S;
    end
    if tu_k is older than a threshold then
        Ageing using eq (4) ;
    end
end
return M
```

3. EXPERIMENTAL EVALUATION ON SPACE APPLICATION

This work is motivated by designing software protection for space electronics against radiation faults. From long term damage caused by cumulative ionising exposition called *Total Ionizing Dose (TID) effects*, to electronic events caused by one highly energetic particle called *Single Event Effects (SEEs)*, space agencies have to find countermeasures to detect and protect the satellite components.

3.1 Radiation effects on space electronics

Total ionising dose (TID) Exposure to intensive radiation environments leads to long term damage. Space components are designed to withstand a specific total dose without being affected to fulfil the mission. Shielding is a method commonly used to limit the effect of TID on components. Unfortunately, devices still suffer from various adverse effects such as increased power consumption, which makes the design of component protection difficult.

Single event effect (SEE) SEEs are faults induced by an ionising particle causing *soft errors* or *hard errors*. (Gaillard (2011)). *Soft errors* are failures affecting the signal or data level without damaging the component physically. For example, a *single event upset (SEU)* is a non-destructive soft error that shifts memory cells' values. A *single event fault interrupt (SEFI)* leads to a temporary interruption of the component. *Hard errors* physically damage the components and persist even after power-cycling. For example, a *single event gate rupture (SEGR)* caused by a single energetic particle strike creates a new conducting path in a MOSFET component. *Single event latch-ups (SEL)* occur when an energetic particle triggers a low impedance path leading to a parasitic structure in the substrate of a micro-component.

Most SEEs lead to *high current events (HCE)*. Detection of these faults is crucial as power-cycling nullify the effects of soft errors. Also, it is possible to avoid most permanent damage of some hard errors if the fault is detected fast enough. For example, in the case of SEL, extensive research is done in order to improve detection (Carlsen (2018), Cibils (2019), Chang; Joseph Sylvester (2020)). Today's baseline solution revolves around a threshold-based detection method. Although effective for destructive hard errors, it fails in detecting faults hidden in the supply current.

3.2 Data sets

The component on which DyD^2 is tested is the ATMEL SAM3X8E microcontroller. It is the COTS version of the SAM3X8ERT that is used in space applications. However, studies show that even the rad tolerant version is subject to radiation faults, compromising space missions (Pilia et al. (2021)).

Multiple simulation scenarios were elaborated to simulate a complex supply current behaviour. Supply current data gathered by these experiments are considered the normal behaviour of the component. Moreover, a laser testing campaign was performed as it is possible to simulate single event effects on electronic devices by striking sensible nodes (Faraud et al. (2011)). From these experiments, a simulation framework was designed, giving access to a significantly larger data base (Dorise et al. (2021)). An example of a simulated data set is shown in Fig. 2.

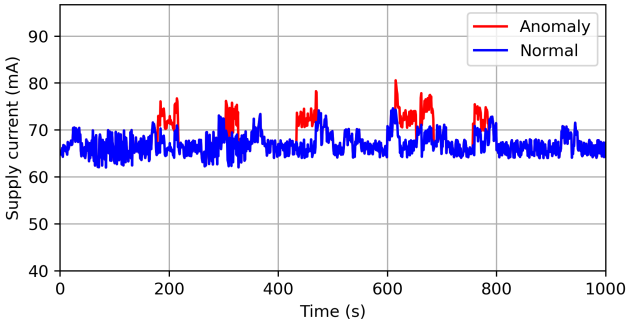

Fig. 2. Simulated data set of a SAM3X8E supply current

The performance of the detection method is tested on the simulated data set. Training is performed using a batch of simulated data sets free of anomalous data. Then, three types of data sets are used to evaluate the performance. The first one aims to evaluate the detection of destructive faults corresponding to outer anomalies. The second aims to test the detection of small and hidden faults corresponding to inner anomalies (see Fig 2 for an example). Finally, the last test shows a linear trend for the supply current to test the dynamic behaviour of DyD^2 (see Fig. 3). Information on these data set are displayed in table 1

Table 1. Information about the data sets

	Train set	Test 1	Test 2	Test 3
Number of set	10	20	20	10
Data per set	1.10^4	1.10^3	1.10^3	1.10^4
Total data	1.10^4	2.10^5	2.10^5	1.10^6
Positive rate	0%	15.93%	13.09%	8.43%

Supply current values are used to create the point samples S_p. Statistical features on sliding time windows of the supply current are used to create the window samples S_w. The minimum, maximum, geometrical mean, variance, standard deviation of the mean, median absolute deviation and k-statistic are the calculated features.

4. RESULTS

State of the art one class classification (OCC) algorithms, namely Elliptic description (ED), Local outlier factor (LOF), isolation forest (IF), one-class support vector machines (OCSVM) and auto-encoders (AE) (Perera et al.

(2021)), are used to evaluate the performances of DyD^2. These algorithms do not perform double detection like DyD^2, so a comparison is performed on inner anomalies by feeding these algorithms with the features of the window samples S_w. ED, LOF, IF, and OCSVM are implemented using the Python library *scikit-learn*. AE is implemented using the Python package *keras*. In the case of DyD^2, the algorithm is written in the C language, as we strive to extend the work on a microcontroller. Also, results of the baseline threshold detection method are presented. The threshold level is set at a value reasonably higher than normal behaviour, as it is done for space applications.

As DyD^2 is a dynamic detection method with online learning, removing faults when detected is a crucial aspect of the process. Otherwise, DyD^2 would adapt to faulty behaviours, thus skewing the results. It is why an extra step of removing the anomalies when detected was performed only for DyD^2.
An overview of the overall results is displayed Fig. 4.

4.1 Parameters

The parameters required by DyD^2 are:

- **Rupture μ-cluster μCl_r size:** Defines the value at which a change point is detected. The higher this parameter, the less points are analysed by DyD^2, but the higher the risk of false negatives.
- **Outer and inner maps μ-cluster size:** Define the double detection accuracy. Small size means being able to detect smaller and hidden faults, but it increases the risk of false positives. Note that it is possible to define a different μ-cluster size for each dimension of the feature space.
- **Time window size**: Defines the number of points used for creating the window samples and their features, which determines the inner map. A higher time window size means that more precise information can be given by the extracted features, but it also increases the detection time of DyD^2.
- **Ageing threshold:** Defines the date limit when a μ-cluster starts to decay. Decaying starts when the difference between the last update of a μ-cluster and the current date is greater than this parameter. A low value tends to favour dynamic behaviour, but it can increase the number of false positives as mapped behaviours may evolve too quickly.
- **Decay penalty:** Defines the decay hastiness of an old μ-cluster. It works as a decreasing linear function on the μ-cluster number of samples n_k (see eq. 4).

Parameters used for the tests are detailed on Table 2 and stay unchanged for all tests.

Table 2. Parameters used

μCl_r size	outer μCl size	inner μCl size	window size	age threshold	decay penalty
0.19	0.05	0.15	20	150	0.95

4.2 Stationary tests

Table 3 shows the results for destructive faults, while Table 4 shows the results for small non-destructive faults. First, the baseline method is able to detect all destructive fault, but is ineffective for non-destructive faults. Globally, AE is the least-performing algorithm regarding anomaly detection. It is probably due to the fact that there are not enough features to train a neural network. On the other hand, OCSVM seems to be the best performing algorithm. DyD^2 performances are on par with the state of the art OCC algorithms. Nevertheless, in terms of computation time, DyD^2 is the fastest among all algorithms implemented using python libraries.

Table 3. Test 1 : destructive faults

	TP(%)	FP(%)	FN(%)	TN(%)	time
Baseline	15.93%	0.00%	0.00%	84.06%	NA
DyD^2	15.83%	**3.00**%	0.10%	**81.06**%	**0.036**s
EC	15.93%	4.06%	0.00%	80.01%	0.1798s
LOF	15.93%	5.34%	0.00%	78.72%	0.442s
IF	15.93%	3.70%	0.00%	80.37%	24.485s
OCSVM	15.93%	3.87%	0.00%	80.19%	0.083s
AE	15.93%	8.98%	0.00%	74.99%	0.096s

Table 4. Test 2 : Non destructive faults

	TP(%)	FP(%)	FN(%)	TN(%)	time
Baseline	0.00%	0.00%	13.09%	86.91%	NA
DyD^2	11.14%	4.53%	1.95%	82.38%	**0.060**s
EC	11.07%	4.43%	2.02%	82.47%	0.179s
LOF	10.52%	3.67%	2.57%	83.24%	0.450s
IF	10.57%	3.84%	2.52%	83.07%	24.378s
OCSVM	**11.17**%	**3.60**%	**1.92**%	**83.31**%	0.082s
AE	11.13%	10.92%	1.96%	75.99%	0.096s

4.3 Linear trend tests

The last test is performed on data showing a linear trend in order to emphasise the dynamic behaviour of DyD^2. To demonstrate the importance of μ-cluster ageing in DyD^2, a test on a version where this functionality is not active is performed (ageing penalty parameter is set to 1). This is referred to as DyD^2-NA (Non Ageing). Table 5 reports the results of all algorithms regarding this particular test. As expected, results show that state of the art OCC algorithms cannot follow the trend, resulting in a high quantity of false positives. The same is observed for DyD^2-NA because μ-cluster ageing is not activated. On the other hand, DyD^2 false positive rate is significantly lower than that of compared algorithms. Hence, it is clearly shown that DyD^2 performs better on non-stationary data sets.

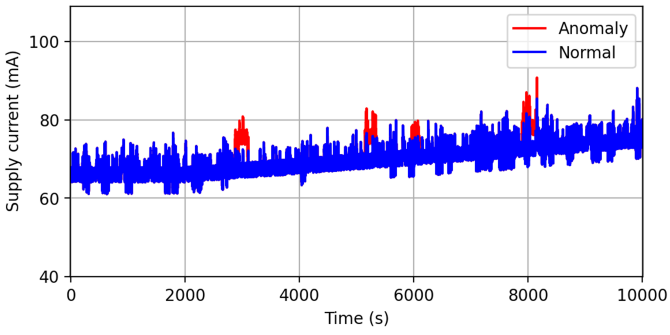

Fig. 3. Simulated data set with linear deviation

Table 5. Test 3 : Deviation testing

	TP(%)	FP(%)	FN(%)	TN(%)	time
Baseline	0.00%	0.00%	8.43%	86.86%	NA
DyD^2	**8.35**%	**4.79**%	**0.08**%	**86.78**%	**0.247**s
DyD^2 NA	8.25%	9.92%	0.18%	81.64%	0.254s
EC	8.20%	32.02%	0.23%	59.54%	1.818s
LOF	6.87%	21.70%	1.56%	69.86%	4.495s
IF	8.26%	37.89%	0.17%	53.68%	254.449s
OCSVM	7.73%	26.98%	0.70%	64.55%	0.846s
AE	8.26%	36.21%	0.17%	55.36%	0.355s

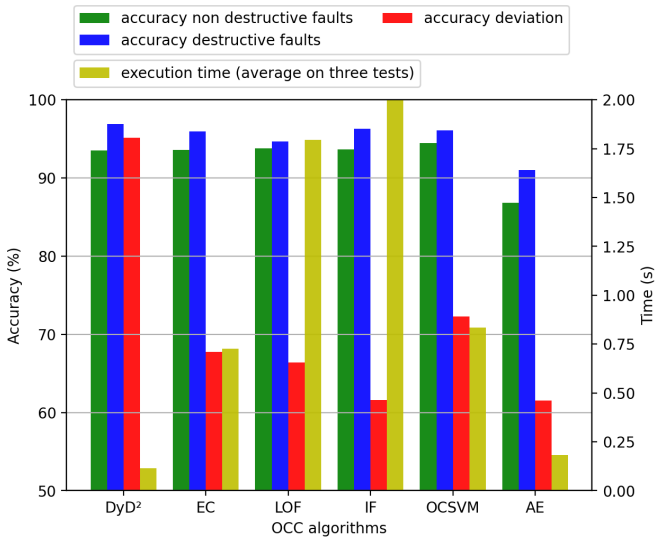

Fig. 4. Overall results

5. CONCLUSION

This paper proposes a new machine learning algorithm for anomaly detection called *Dynamic Double anomaly Detection* (DyD^2). It is designed to analyse the data only when possible anomalies are present thanks to a primary change point detection method. Subsequently, two anomaly detection methods are applied in cascade. The first one detects high range destructive faults and hands over to the second method when no such fault is detected. DyD^2 is an on-board compatible one class classification algorithm as only normal data are needed for training. In the experimental part of the paper, a space application is used to evaluate and compare DyD^2 with state-of-the-art OCC algorithms. DyD^2 performances are on par with these algorithms for stationary data sets. Plus, results show that DyD^2 outperforms the compared algorithms for non-stationary data sets and computation time.

Several points need to be investigated in the future. First, experiments made with DyD^2 showed that it hardly follows fast trends Improvements are required in this respect. Second, experiments must be performed to test the performances of DyD^2 on a on-board system such as a microcontroller. Finally, radiation testing must be conducted to test the detection accuracy of DyD^2 in real case scenarios.

ACKNOWLEDGEMENTS

We kindly thank François Vacher, Leny Baczkowski and Thomas Torloting from CNES for their help in this work.

REFERENCES

Aggarwal, C.C., Hinneburg, A., and Keim, D.A. (2001). On the surprising behavior of distance metrics in high dimensional space. In J. Van den Bussche and V. Vianu (eds.), *Database Theory — ICDT 2001*, 420–434. Springer Berlin Heidelberg, Berlin, Heidelberg.

Aminikhanghahi, S. and Cook, D. (2017). A survey of methods for time series change point detection. *Knowledge and Information Systems*, 51. doi:10.1007/s10115-016-0987-z.

Banbury, C.R., Reddi, V.J., Lam, M., Fu, W., Fazel, A., Holleman, J., Huang, X., Hurtado, R., Kanter, D., Lokhmotov, A., Patterson, D.A., Pau, D., Seo, J., Sieracki, J., Thakker, U., Verhelst, M., and Yadav, P. (2020). Benchmarking tinyml systems: Challenges and direction. *CoRR*, abs/2003.04821. URL `https://arxiv.org/abs/2003.04821`.

Barbosa Roa, N., Travé-Massuyès, L., and Grisales, V.H. (2019). DyClee: Dynamic clustering for tracking evolving environments. *Pattern Recognition*, 94, 162–186. doi:10.1016/j.patcog.2019.05.024.

Carlsen, J.B. (2018). *Design and Validation of Two Single Event Latch-up Protection Solutions. Comparing a New Single Event Latch-up Test Circuit with the IDEAS IDE3466 Single Event Latch-up Detection Module*. Ph.D. thesis, University of Oslo.

Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3). doi:10.1145/1541880.1541882.

Chang; Joseph Sylvester, Shu; Wei, J.J. (2020). Electronic circuit for single-event latch-up detection and protection. *Patent US20170237250A1*.

Chen, J., Sathe, S., Aggarwal, C., and Turaga, D. (2017). *Outlier Detection with Autoencoder Ensembles*, 90–98. doi:10.1137/1.9781611974973.11.

Cibils, R. (2019). Método para actualizar el umbral de referencia de al menos un parámetro operativo, unidad de protección para la mitigación de un evento simple de latchup (sel) en un dispositivo electrónico usando el umbral de referencia y disposición para la mitigación de un evento simple de latchup (sel) en un conjunto. *Patent n°AR116929A1*.

Dorise, A., Alonso, C., Subias, A., Travé-Massuyès, L., Baczkowski, L., and Vacher, F. (2021). Machine learning as an alternative to thresholding for space radiation high current event detection. In *Radiation and its Effects on Components and Systems - RADECS 2021*. Vienna, Austria. URL `https://hal.laas.fr/hal-03331030`.

Etiemble, D. (2018). 45-year CPU evolution: one law and two equations. *CoRR*, abs/1803.00254. URL `http://arxiv.org/abs/1803.00254`.

Faraud, E., Pouget, V., Shao, K., Larue, C., Darracq, F., Lewis, D., Samaras, A., Bezerra, F., Lorfèvre, E., and Ecoffet, R. (2011). Investigation on the sel sensitive depth of an sram using linear and two-photon absorption laser testing. *IEEE Transactions on Nuclear Science*, 58, 2637–2643.

Fuertes, S., Picart, G., Tourneret, J.Y., Chaari, L., Ferrari, A., and Richard, C. (2016). Improving Spacecraft Health Monitoring with Automatic Anomaly Detection Techniques. In *14th International Conference on Space Operations (SpaceOps 2016)*, pp. 1. Daejeon, South Korea. URL `https://hal.archives-ouvertes.fr/hal-01490731`.

Gaillard, R. (2011). *Single Event Effects: Mechanisms and Classification*, 27–54. Springer US, Boston, MA. doi:10.1007/978-1-4419-6993-4_2. URL `https://doi.org/10.1007/978-1-4419-6993-4\_2`.

Hyde, R., Angelov, P., and MacKenzie, A.R. (2017). Fully online clustering of evolving data streams into arbitrarily shaped clusters. *Information Sciences*, 382, 96–114.

Khan, S.S. and Madden, M.G. (2014). One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(3), 345–374. doi:10.1017/S026988891300043X.

Pat Gelsinger, A.K. (2021). Intel accelerated webcast. *Intel Accelerated*. URL `https://www.intel.com/content/www/us/en/events/accelerated.html`.

Perera, P., Oza, P., and Patel, V.M. (2021). One-class classification: A survey. *CoRR*, abs/2101.03064. URL `https://arxiv.org/abs/2101.03064`.

Pilia, R., Espinasse, R., Poulet, C., Bezerra, F., Gillot, L., Treuillard, B., and Dumortier, S. (2021). SEE Radiation Analysis And Mitigation on SAM3X8ERT Microcontroller.