https://doi.org/10.1590/jatm.v13.1186 ORIGINAL PAPER

Using Multiple Deep Neural Networks Platform to Detect Different Types of Potential Faults in Unmanned Aerial Vehicles

Ahmad Alos[1,*], Zouhair Dahrouj[1]

1. Higher Institute for Applied Science and Technology – Informatics – Damascus – Syria

*Corresponding author: ahmad.alos@hiast.edu.sy

ABSTRACT

Many researchers developed new algorithms to predict the faults of unmanned aerial vehicles (UAV). These algorithms detect anomalies in the streamed data of the UAV and label them as potential faults. Most of these algorithms consider neither the complex relationships among the UAV variables nor the temporal patterns of the previous instances, which leaves a potential opportunity for new ideas. A new method for analyzing the relationships and the temporal patterns of every two variables to detect the potentially defected sensors. The proposed method depends on a new platform, which is composed of multiple deep neural networks. The method starts by building and training this platform. The training step requires reshaping the dataset into a set of subdatasets. Each new subdataset is used to train one deep neural network. In the testing phase, the method reads new instances of the UAV testing dataset. The output of the algorithm is the predicted potential faults. The proposed approach is evaluated and compared it with other well-known algorithms. The proposed approach showed promising results in predicting different kinds of faults.

Keywords: UAV; Deep neural network; Anomaly detection; Abnormal; Fault.

INTRODUCTION

In complex systems such as the unmanned aerial vehicles (UAV), the chances of failure are hazardously high. The streamed data of the flight missions contain vast knowledge that can be used in defining and predicting potential faults. The data are stored at a fixed rate in data rows. Each data row contains the values of the UAV variables. The variables are either command (elevator, rudder, and the aileron command) or sensor readings (altitude, longitude, latitude, and airspeed). To foresee system failure, anomaly detection algorithms are used. Anomaly detection algorithms find patterns in data that do not follow an expected behavior. These algorithms are either supervised or unsupervised. The supervised algorithms are trained using datasets with labels for each instance (normal/abnormal) labels. However, the unsupervised algorithms do not involve any labeling for the datasets, and they assume that the abnormal patterns are less frequent than normal ones (Chandola *et al.* 2009).

Received: Mar. 19, 2020 | **Accepted:** Aug. 13, 2020
Peer Review History: Double Blind Peer Review.
Section Editor: Alison Moraes

Furthermore, the potential faults of the UAV are either: point, contextual, or collective (Chandola *at al.* 2009). The point fault arises when the tested value is invalid or out of bounds. The contextual fault occurs when the value of the instance is not relevant to the current context. The collective fault occurs when consecutive instances are considered invalid as a group when they put together successively (Sun *et al.* 2017). Suppose that the UAV is on a low altitude, but the altimeter shows invalid increasing values. The pilot decides to land the UAV, so he sends wrong commands to the UAV to decrease its altitude. These unsuitable commands could result in the crash of the UAV. In this example, the altitude command is annotated as the dependent variable, and the altitude sensor as the tested variable. To generalize, we associate the tested variable with the potentially defected sensor, while the dependent variable represents the context of the fault.

Our contribution involves using a new platform of multiple deep neural networks (MDNN) to analyze the relationships of every possible (dependent, tested) couple. The tested variable could be affected by more than one dependent variable, so it is essential to take into account all the pairs. Using relationships of more than two variables in the multivariate dataset might detect the fault. However, it will not permit determining the context of the fault (the dependent variable), because the effect of the different relationships will result in losing track of the context. Thus, each pair (dependent, tested) is considered separately. In addition to the relationships, it is necessary to analyze the temporal patterns in the dependent, tested relationship. The temporal patterns are achieved by considering a sliding window technique (Ding *et al.* 2014). Each sliding window consists of the previous instances of the dependent variable, and the differential values of the tested one.

The proposed approach reads a few rows of the data iteratively; then, it reshapes the rows into new subdatasets. Each new subdataset is used to train one DNN of the collection. The trained MDNN platform is used to detect abnormal instances, which could be potential faults.

The well-known "FLTz" synthetic dataset (Oza 2011) was used. The "FLTz" dataset contains several flights of a fixed-wing aircraft. Some of the "FLTz" flights were used for training the proposed model, and the rest of the flights were used for testing it.

The proposed model was compared with other well-known approaches such as the KNN (K-nearest neighbor), the One-Class SVM (support vector machine), and the Kernel SVM. The MDNN algorithm exhibited promising results in all experiments, in which it detected accurately different types of faults, and worked better than the other algorithms while processing the stuck, drift, and cut faults.

LITERATURE REVIEW

Over recent years, detecting system faults became the interest of many researchers, where they designed many algorithms to extract data anomalies to predict faults. Casas *et al.* (2016) used decision trees to detect anomalies in cellular network data. Their approach correctly recognized more than 80% of the abnormal instances with no false positives. He *et al.* (2019) presented an anomaly detection and mitigation algorithm based on online subspace tracking ("ADMOST"). Their method detected the outlier instance in the multivariate heterogeneous data stream with high accuracy and mitigated them with low error.

The progress in neural network architectures and machine learning led to a leap forward in performance and efficient processing. Most of the existing approaches used neural networks to learn the behavior of a training dataset. The learned neural network predicts the data online. It uses the error of prediction to extract the outliers during flight missions Hundman *et al.* (2018), Saurav *et al.* (2018) and Vinayakumar *et al.* (2018) presented the efficiency of the deep learning approach and long short-term memory (LSTM) networks for detecting anomalies for Android malware detection. Their approach obtained 0.987 detection rate and 0.939 accuracy using an LSTM network with six layers. Wang *et al.* (2019) built a time series prediction model based on the LSTM network. They estimated the uncertainty interval to conduct point anomaly detection. Their method scored a recall rate close to one on two test datasets. Munir *et al.* (2019) used a deep learning-based approach for anomaly detection in time-series data. Their technique was accurate even for small deviations in time series cycles. It was capable of detecting point and contextual anomalies in time series with periodic and seasonality characteristics. Althubiti *et al.* (2019) applied an optimized model of LSTM to implement an anomaly detection system. They claimed that the optimized model of LSTM obtained an accuracy of

0.8483. Also, they found that LSTM performed better than support vector machine, multiple perceptron networks, and Naïve Bayes techniques. Despite the good results, these algorithms do not provide the desired objective, as they do not specifically catch contextual faults, neither the context of the fault, and this leads to the benefit of the proposed approach.

Multiple neural networks have been utilized previously for different applications. The multiple neural networks give more reliable results for the occurrence of faults than a single neural network. Zhang (2006) proposed a fully connected architecture of a multiple neural networks platform. The outputs of the multiple DNNs were combined to give a single overall result. Karjol *et al.* (2018) used MDNN for speech enhancement and estimated the speech spectrum as a weighted average of outputs from multiple DNNs. The used platform is not fully connected as the previous platforms, because fully connected architecture may lead to mixed relationships, and this could affect the results of the presented model.

METHODOLOGY

The presented method uses a platform of MDNN. The DNN is an artificial neural network (ANN) with multiple hidden layers that are stacked together (Singh 2017). The ANN is an algorithm used in solving a wide range of problems, including classification, clustering, and pattern recognition (Ullah *et al.* 2019). It is a computational system that learns to perform tasks by training. The ANN is based on a collection of connected nodes called neurons. The neurons of the neural network are aggregated into layers. Each neuron in a given layer is connected to every neuron in the next layer. The input layer receives data from outside the network. The output layer generates the output results, and single or multiple hidden layers transform and transfer the data from the input layers to the output layers. The job of a neuron can be represented mathematically by Eq. 1, which is adapted from (Singh 2017). Each neuron receives some input signals; then, it multiplies the received inputs (element-wise multiplication) with corresponding values called weights. Next, the neuron sums the result with a bias value; then, it applies an activation function; for example, the sigmoid σ function (see Eq. 2).

$$y(x) = f(w \odot x + b) \tag{1}$$

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2}$$

where x is the input vector, w is the weight vector, v is the neuron bias, $\circ$ is the element-wise multiplication, f is the activation function, y is the neuron output, and z is the input of the activation functions. The neuron sends the results to the next neurons that are connected to it in the subsequent layers. The weights values are adjusted through the training phase. Training the network is accomplished in an organized and efficient technique, such as the error back-propagation method, which is widely used in most ANN prototypes (Wang *et al.* 2013), and it is explained briefly in (Yu and Wilamowski 2016).

Multiple deep neural network (MDNN)

Figure 1 shows a block diagram of the MDNN method, where it consists of two phases: (1) the building and training phase and (2) the testing phase. In the first phase, the algorithm collects the variables of the UAV, and form an array of elements. Each element of this array is a possible (dependent, tested) couple. Then, it builds the MDNN platform by defining one DNN for each (dependent, tested) couple. Accordingly, the algorithm reshapes the training dataset into a set of subdatasets. The rows of each subdataset are the values of a sliding window. Each sliding window consists of the previous instances of the dependent variable, and the differential values of the tested one. The algorithm trains the MDNN platform using the new subdatasets. In the testing phase, the algorithm reads the instances at each time step, constructs the sliding window for the (dependent, tested) couple, and uses the trained MDNN platform to detect the abnormal instances; consequently, it predicts the potential faults.

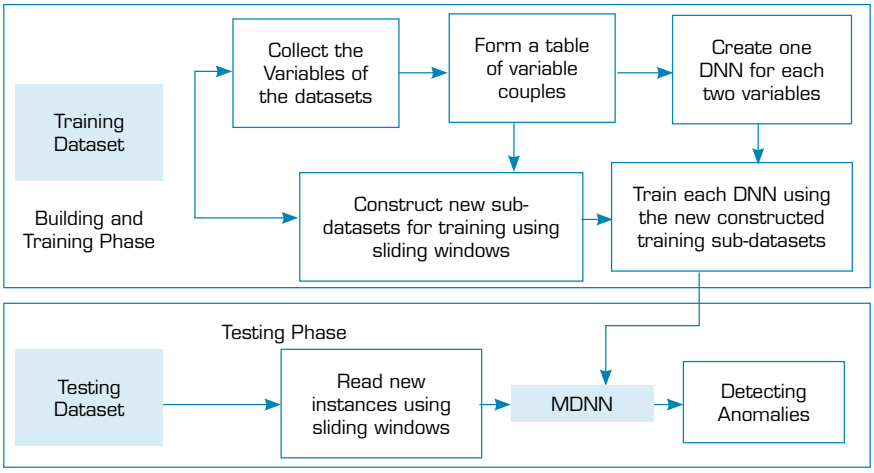

Figure 1. Block diagram of the MDNN algorithm.

The set of m tested variables is represented by $P = \{p_j : j < m\}$, and the set of n dependent variables by $Q = \{q_k : k < n\}$. The values of each couple (q_k, p_j) at time step t are $(x^t_{q_k}, x^t_{p_j})$. The goal is to predict potential faults by extracting the abnormal temporal patterns of the two variables (q_k, p_j). To build the new platform of the MDNNs, Algorithm 1 is used. This algorithm starts by declaring NN as an empty matrix, whose elements will be the new $(m \times n)$ DNNs (see Fig. 2).

Algorithm 1. Building MDNN platform.

```
procedure building_MDNN_platform()
    declare the structure NN [m × n]
        for each variable q_k do
            for each variable p_j do
                create a deep neural network NN_(q_k,p_j)
                D_(q_k,p_j) ← Ø
                    for each step t do
                        W^(t,h)_(q_k,p_j) ← [x^(t-h+1)_qk, ... ,x^t_qk, Δx^(t-h+1)_pj, Δx^t_pj]
                        add [W^(t,h)_(q_k,p_j), class (x^t_qk, Δx^t_pj)] to D_(q_k,p_j)
                    end for
                train network NN_(q_k,p_j) using D_(q_k,p_j)
            N[k,j] ← NN_(q_k,p_j)
        end for
    end for
    return NN (The Multiple Deep Neural Network)
end procedure
```

The algorithm creates a neural network $NN_{(q_k,p_j)}$ for each (q_k, p_j) couple. These couples can be nominated with the help of an expert in the field. Then, the algorithm creates a subdataset D_{q_k,p_j} using the training dataset. The subdataset D_{q_k,p_j} is used for training $NN_{(q_k,p_j)}$. Each row of D_{q_k,p_j} consists of the input and the output of the neural network $NN_{(q_k,p_j)}$. The input $W^{t,h}_{q_k,p_j}$ at time step t is a sequential list of the current and the previous instances of the two variables (q_k, p_j), and it consists of $2h$ elements (Eq. 3), where h is the size of the sliding window.

$$W^{t,h}_{q_k,p_j} = \left[x^{t-h+1}_{q_k}, \ldots, x^{t}_{q_k}, \Delta x^{t-h+1}_{p_j}, \ldots, \Delta x^{t}_{p_j}\right] \tag{3}$$

in Eq. 3: the differential values $\Delta x^t_{p_j} = x^t_{p_j} - x^{t-1}_{p_j}$ are used to increase the sensitivity of the algorithm for the occurrences of faults in the Tested variable p_j, as suggested by Khalastchi and Kalech (2018).

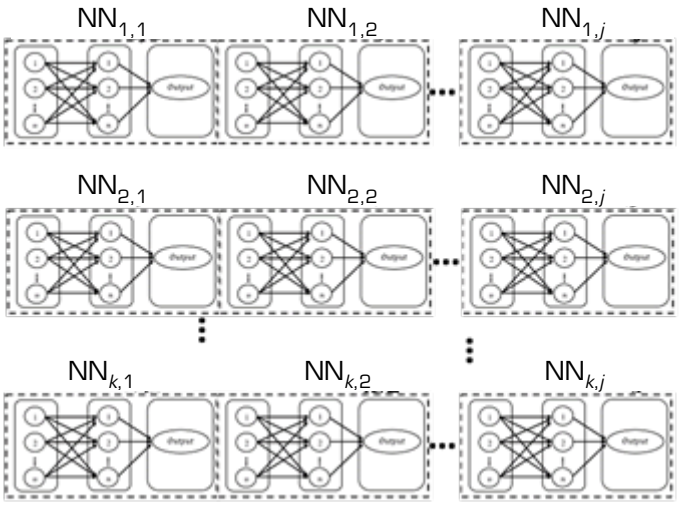

Figure 2. The MDNN structure.

The training output of the neural network $NN_{(q_k,p_j)}$ is the class of the point $(x^t_{q_k}, \Delta x^t_{p_j})$, where $class\ (x^t_{q_k}, \Delta x^t_{p_j}) \in \{Zero, One\}$. Zero is the abnormal class, and one is the normal class. By combining the input and the output, a row of D_{q_k,p_j} is constructed. Iteratively and in the next time steps, the algorithm constructs the next rows of D_{q_k,p_j}. Equation 4 shows the shape of D_{q_k,p_j}.

$$D_{q_k,p_j} = \{[W^{t,h}_{q_k,p_j}, class\left(x^t_{q_k}, \Delta x^t_{p_j}\right)]: 0 \le t \le N-1\} \tag{4}$$

where N is the size of the new subdataset. After creating D_{q_k,p_j}, the algorithm trains $NN_{(q_k,p_j)}$ and assigns it to the element $NN[k,j]$ to be used in the testing phase.

The mathematical model for a neural network $NN[k,j]$ is realized using a nonlinear functional mapping from the values of the input $W^{t,h}_{q_k,p_j}$ to the output $y^t_{k,j}$ (Eq. 5).

$$y^t_{k,j} = f(w_{k,j} \odot W^{t,h}_{q_k,p_j} + b_{k,j}) \tag{5}$$

where $w_{k,j} = \{\theta_1, \theta_2 \ldots, \theta_i, \ldots\}$ is a vector of all weight values, $b_{k,j} = \{b_1, b_2 \ldots, b_i, \ldots\}$ is a vector of all bias values of the neural network. θ_i is the i^{th} weight value and b is the i^{th} bias value. f is a nonlinear function that is determined by the structure of the neural network $NN[k,j]$ (Wang *et al.* 2013). Training the neural network using the method of error back-propagation allows the system to learn any given mapping of input to output. The back-propagation method is used by the gradient descent algorithm. The gradient descent algorithm is an iterative optimization algorithm that tries to find the local minimum of the loss function. The loss function E is calculated using Eq. 6. The descent algorithm calculates the gradient ∇E as the first-order derivative of the total error function (Eq. 7), which adapted from (Yu and Wilamowski 2016). At each iteration of the gradient descent method, the values of θ_i, b_i are adjusted using Eq. 8 and 9 (the reader is referred to Yu and Wilamowski [2016] for the mathematical extraction steps for Eqs. 8 and 9).

$$E = \left[y^t_{k,j} - class\left(x^t_{q_k}, \Delta x^t_{p_j}\right)\right]^2 \tag{6}$$

$$\nabla E = \left[\frac{\partial E}{\partial \theta_1}, \ldots \frac{\partial E}{\partial \theta_i}, \frac{\partial E}{\partial b_1}, \ldots \frac{\partial E}{\partial b_i} \ldots\right]^T \tag{7}$$

$$\Delta\theta_i = -\eta \frac{\partial E}{\partial \theta_i} \tag{8}$$

$$\Delta b_i = -\eta \frac{\partial E}{\partial b_i} \tag{9}$$

where $\eta > 0$ is the learning rate, which can be adjusted during the training process. Increasing the learning rate value makes the learning process faster, but it might affect the sensitivity of the neural network (Yu and Wilamowski 2016). In the testing phase, if the input vector $W^{t,h}_{q_k,p_j}$ outputs an Anomaly, then the algorithm considers the instance $(x^t_{q_k}, \Delta x^t_{p_j})$ as a potential fault (see Fig. 3).

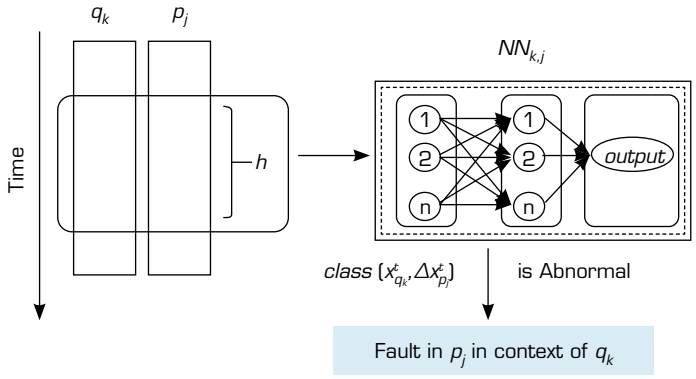

Figure 3. Detecting faults using neural network *NN*[*k,j*].

RESULTS AND DISCUSSION

The well-known "FLTz" synthetic dataset is used. This dataset is shared by Oza (2011) for public research purposes, and it contains 20 flights of a fixed-wing aircraft with periods up to 40 min. "FLTz" is a flight simulator used to develop flight control, planning, and in-flight fault detection (Chu *et al.* 2010). Each flight includes all stages as takeoff, climb, cruise, and descent. Each flight consists of 36 variables, and it is recorded at a rate of 1 Hz. The experiments were conducted by employing 14 sensor readings and four commands. The first 14 sensor readings variables were considered as the tested variables, and all the 18 variables were considered as the dependent ones. Table 1 shows the variables of the FLTz dataset.

Table 1. The variables of the FLTz dataset (bold variables are the tested ones).

Variable	Range	Unit	Variable	Range	Unit
Pitch	[-0.3,0.3]	*rad*	Lateral acceleration	[-8,8]	m/s^2
Airspeed	[50,300]	*m/s*	Vertical acceleration	[-40,40]	m/s^2
Velocity	[100-500]	*m/s*	Roll acceleration	[-0.3,0.3]	rad/s^2
Lateral velocity	[-10-10]	*m/s*	Pitch acceleration	[0.15,0.15]	rad/s^2
Vertical velocity	[-20-100]	*m/s*	Yaw acceleration	[-0.03,0.03]	rad/s^2
Roll rate	[-0.15,0.15]	*rad/s*	Left aileron command	[-10,10]	_
Pitch rate	[-0.05, 0.05]	*rad/s*	Right aileron command	[-10,10]	_
Yaw rate	[-0.06, 0.06]	*rad/s*	Elevators command	[-25,25]	_
Forward acceleration	[-10,15]	m/s^2	Rudder command	[-6,6]	_

Using the "FLTz" dataset, a new dataset was generated for the training phase with 40,000 rows by concatenating randomly multiple flights. Four different flights for testing purposes were used. Each flight was injected by one fault type (impulse, stuck, cut, and drift) into various variables such as (pitch, pitch rate, and airspeed). Table 2 shows the injected faults in the selected

four flights. The impulse fault means that the sensor shows an offset value added to its actual value in one instance (see Fig. 4 and note the small range of the impulse, which is about [0, 0.07]). The stuck fault occurs when the value of the sensor is stuck at a specific reading (Fig. 5). The drift fault means that the readings of the sensor increase through time where they should not (Fig. 6), and the cut fault occurs when the sensor shows unexpected continuous zero reading for a limited period (Fig. 7).

Table 2. The injected faults in the testing flights.

Dataset	Fault type	Faults count	Defected sensor
Flight1	impulse	48	Pitch rate
Flight2	stuck	1085	Airspeed
Flight3	drift	435	Pitch
Flight4	cut	199	Airspeed

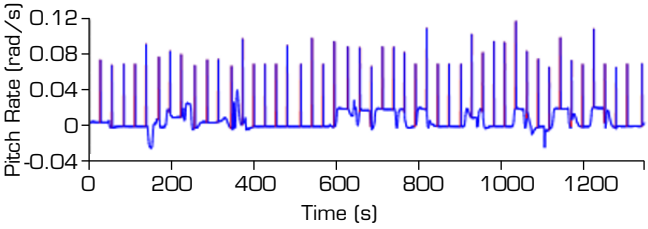

Figure 4. Pitch rate impulse faults.

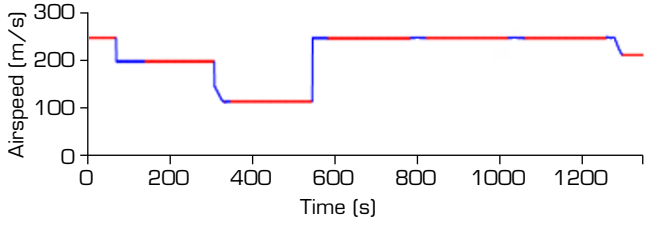

Figure 5. Airspeed stuck faults.

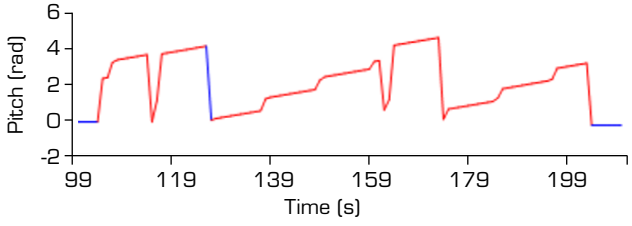

Figure 6. Pitch drift faults.

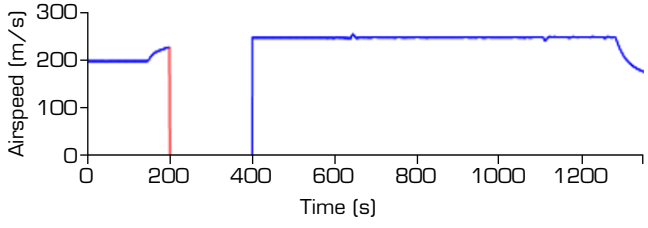

Figure 7. Airspeed cut faults.

Multiple experiments were conducted to evaluate the results of the presented approach. The results were compared with the results of other known algorithms such as KNN, One-Class SVM, and Kernel SVM. All algorithms were tested using the same flights.

The MDNN platform (Algorithm 1) consisted of 238 DNNs. Each neural network consisted of one input layer, one output layer, and three stacked hidden layers. In the experiments, the input layer contained $2h = 8$ neurons ($2h$ is the length of the input $W^{\tau,h}_{q_k p_j}$), the hidden layers structure had [8, 16, 8] neurons in three layers, respectively. Using three hidden layers helped to get acceptable results. The output layer had one neuron for detecting anomalies.

EVALUATION INDICATORS

To evaluate the anomaly detection algorithms, the following indicators were used: recall or detection rate (Eq. 10), false alarm rate (FAR) (Eq. 11), and precision (Eq. 12). Precision is an indicator of whether the detected anomalies are trustworthy. Also, the Score (Eq. 13) was used to evaluate both the precision and the recall (Lee and Kim 2019). (These indicators are widely used in the area of anomaly detection).

$$Recall = \frac{TP}{TP+FN} \tag{10}$$

$$FAR = \frac{FP}{FP+TN} \tag{11}$$

$$Precision = \frac{TP}{TP+FP} \tag{12}$$

$$F.Score = \frac{2*Precision*DR}{Precision+DR} \tag{13}$$

where TP is the true positive (FP) is the false positive (FP), (TN) is the true negative (TN), and (FN) is the false negative (FN). The algorithm with maximum *Recall*, *Precision*, and *F.score* as well as minimum *FAR* (Yong *et al.* 2017) presents better efficiency.

FAULT DETECTION EXPERIMENTS

The proposed approach was compared with other well-known algorithms such as the KNN, the One-Class SVM (support vector machine), and the Kernel SVM. The K nearest neighbor algorithm KNN is used for classification problems by estimating the local density of the data points (Ullah *et al.* 2019). It depends on the distance between the tested value and its nearest neighbors. Usually, the Euclidean distance is used. The One-Class SVM (support vector machine) finds a boundary that surrounds the normal instances of the training dataset. For that, it decides that the test instance falls within the region of the learned boundary, through a linear decision function (For more details of the linear decision functions, the reader is referred to Chandola *et al.* [2009] and Bounsiar and Madden [2014]). The SVM algorithm increases its efficiency to perform nonlinear classification by applying kernel functions (Kernel SVM) (Guo *et al.* 2015). The kernel functions map pairs of objects to their similarity. The values of the similarity range between one and zero, where value one is given for maximum similarity, and value zero is given for no similarity (Das *et al.* 2010). Commonly, the Gaussian kernel with Euclidean distance measure is used (Juvonen *et al.* 2015).

Table 3 shows that the required training time for the Kernel SVM algorithm was the longest; this was because of the computational complexity of the kernel function (Janakiraman and Nielsen 2016). The training time for the MDNN algorithm was also long because of the large number of neurons that need to be trained.

Table 3. The total training time for the anomaly detection algorithms.

Algorithm	Training time (ms)
MDNN	10,409,078
KNN	126,257
SVM	104,856
Kernel SVM	17,500,046

The MDNN algorithm scored good results considering all the evaluation indicators (Table 4), where its recall and precision had the highest values, and its false alarm rate approached zero. Multiple deep neural networks scores were better than the scores of Kernel SVM and KNN algorithms; for example, the precision of the MDNN algorithm was generally better. Additionally, in the testing phase, the KNN algorithm was the slowest, while the One-Class SVM was the fastest as Fig. 8 shows.

Table 4. Results of testing the anomaly detection algorithms.

Algorithm	Flights	Recall	FAR	Precision	F.Score
MDNN	Flight1	1	0.02	0.96	0.98
	Flight2	1	0.14	0.97	0.98
	Flight3	1	0.03	0.94	0.97
	Flight4	1	0.01	0.96	0.98
KNN	Flight1	1	0	1	1
	Flight2	1	0.53	0.89	0.94
	Flight3	1	0.02	0.96	0.98
	Flight4	1	0.09	0.67	0.8
SVM	Flight1	1	0	1	1
	Flight2	1	1	0.8	0.89
	Flight3	0.03	0	1	0.05
	Flight4	0.02	0	1	0.03
Kernel SVM	Flight1	1	0	1	1
	Flight2	1	0.85	0.83	0.9
	Flight3	1	0.09	0.84	0.91
	Flight4	0.82	0.03	0.86	0.84

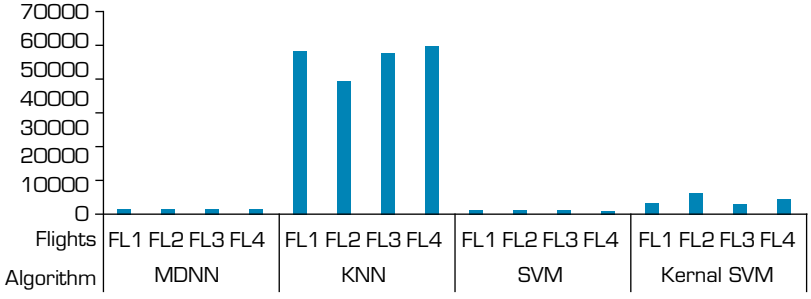

Figure 8. The testing time to perform the algorithms for each dataset.

The One-Class SVM was better in detecting impulse faults in Flight1; however, its false alarm rate was high for stuck faults in Flight2, and its recall approached zero in the case of the drift in Flight3 and cut faults in Flight4, and this means that One-class SVM failed to detect faults that have a continuous nature.

In Flight1 (impulse-faults, Fig. 4), the precision of the MDNN algorithm was acceptable (0.96), but it was a bit lower from that of KNN, SVM, and Kernel SVM, and the reason is the small range of the impulses. In Flight2 (stuck faults), Flight3 (cut faults), and Flight4 (drift faults), the precision indicator of the KNN, SVM, and Kernel SVM was high. However, the false alarm rate was high, and this was because it was difficult for these algorithms to separate the significant number of outliers from the normal instances due to the continuous nature of the faults. For the same reasons, the One-Class SVM showed low detection rate when processing Flight3 (sensor-drift) and Flight4 (sensor-cut), therefore the F.Score indicator was low too. However, the precision indicator was high (refer to Eq. 12).

K-nearest neighbor efficiency was similar to the MDNN algorithm in processing Flight3 and Flight4. The reason is the increased distance of these outliers from the normal instances, which increased the KNN reliability. Note that the MDNN algorithm was not affected by the continuous nature of the outliers, and this caused the MDNN algorithm (which is a supervised algorithm) to be better from the other algorithms.

CONCLUSION

In this paper, a novel method is proposed for predicting several types of potential faults by analyzing the relationships between the variables of the UAV flights and considering the temporal patterns of the previous values. The new method depends on a platform of MDNN. Each DNN is responsible for detecting anomalies in the instances of a couple (dependent, tested) variables. The MDNN algorithm worked better than other algorithms while processing the stuck, drift, and cut faults, which have continuous nature. On the other hand, it was sensitive to the small values of the abnormal impulses, but its precision was a bit lower than the other algorithms.

Future work includes improvements to make the algorithm faster in the training phase by optimizing the structure of the neural networks and increase their precision or by enhancing the performance by testing different platforms, such as using one tested variable versus many dependent variables. Also, new methods could be explored for choosing the appropriate couples of variables in order to minimize the size of the processed values. Moreover, the new platform can be used to test and compare other machine learning classifiers, such as long short-memory networks and logistic regression.

AUTHOR'S CONTRIBUTION

Conceptualization: Alos A and Dahrouj Z; **Methodology:** Alos A; **Computational Works:** Alos A; **Analysis:** Alos A and Dahrouj Z; **Writing – Original Draft:** Alos A; **Writing – Review and Editing:** Alos A and Dahrouj Z; **Supervision:** Dahrouj Z.

DATA AVAILABILITY STATEMENT

The data will be available upon request.

FUNDING

Not applicable.

ACKNOWLEDGEMENTS

Not applicable.

REFERENCES

Althubiti SA, Jones EM, Roy K (2019) LSTM for anomaly-based network intrusion detection. 2018 28th International Telecommunication Networks and Applications Conference. IEEE; Sydney, Australia. https://doi.org/10.1109/ATNAC.2018.8615300

Bounsiar A, Madden MG (2014) One-class support vector machines revisited. Paper presented ICISA 2014 - 2014 5th International Conference on Information Science and Applications. IEEE; Seoul, South Korea. https://doi.org/10.1109/ICISA.2014.6847442

Casas P, Fiadino P, D'Alconzo A (2016) Machine-learning based approaches for anomaly detection and classification in cellular networks. Paper presented 8th Traffic Monitoring and Analysis (TMA2016) Workshop. TMA; Louvain La Neuve, Belgium.

Chandola V, Banerjee A, Kumar V (2009) Anomaly detection: a survey. ACM Comput Surv 41(3):15. https://doi.org/10.1145/1541880.1541882

Chu E, Gorinevsky D, Boyd S (2010) Detecting aircraft anomalies cruise flight data. Paper presented AIAA Infotech@ Aerospace 2010. AIAA, Atlanta, USA. https://doi.org/10.2514/6.2010-3307

Das S, Matthews BL, Srivastava AN, Oza NC (2010) Multiple kernel learning for heterogeneous anomaly detection. Paper presented KDD '10: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. KDD; New York, USA. https://doi.org/10.1145/1835804.1835813

Ding X, Li Y, Belatreche A, Maguire LP (2014) An experimental evaluation of novelty detection methods. Neurocomputing 135:313-27. https://doi.org/10.1016/j.neucom.2013.12.002

Guo X, Denman S, Fookes C, Mejias L, Sridharan S (2015) Automatic UAV forced landing site detection using machine learning. Paper presented 2014 International Conference on Digital Image Computing: Techniques and Applications. IEEE; Wollongong, Australia. https://doi.org/10.1109/DICTA.2014.7008097

He Y, Peng Y, Wang S, Liu D (2019) ADMOST: UAV Flight data anomaly detection and mitigation via online subspace tracking. IEEE Trans Instrum Meas 68(4):1035-1044. https://doi.org/10.1109/TIM.2018.2863499

Hundman K, Constantinou V, Laporte C, Colwell I, Soderstrom T (2018) Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. KDD '18: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD; New York, USA. https://doi.org/10.1145/3219819.3219845

Janakiraman VM, Nielsen D (2016) Anomaly detection in aviation data using extreme learning machines. Paper presented 2016 International Joint Conference on Neural Networks. IEEE; Vacouver, Canada. https://doi.org/10.1109/IJCNN.2016.7727444

Juvonen A, Sipola T, Hämäläinen T (2015) Online anomaly detection using dimensionality reduction techniques for HTTP log analysis. Comput Netw 91:46-56. https://doi.org/10.1016/j.comnet.2015.07.019

Karjol P, Kumar MA, Ghosh PK (2018) Speech Enhancement using multiple deep neural networks. Paper presented 2018 IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE; Calgary, Canada. https://doi.org/10.1109/ICASSP.2018.8462649

Khalastchi E, Kalech M (2018) A sensor-based approach for Fault detection and diagnosis for robotic systems. Auton Robot. 42:1231-1248. https://doi.org/10.1007/s10514-017-9688-z

Lee S, Kim HK (2019) ADSaS: Comprehensive real-time anomaly detection system. Paper presented WISA 2018 International Workshop on Information Security Applications. Jeju Island, Korea. https://doi.org/10.1007/978-3-030-17982-3_3

Munir M, Siddiqui SA, Dengel A, Ahmed S (2019) DeepAnT: A deep learning approach for unsupervised anomaly detection in time series. IEEE Access 7:1991-2005. https://doi.org/10.1109/ACCESS.2018.2886457

Oza N (2011) FLTz flight simulator. NASA Dash Link. [accessed Jun 06 2020]. https://c3.ndc.nasa.gov/dashlink/resources/294/

Saurav S, Malhotra P, Vishnu TV, Gugulothu N, Vig L, Agarwal P, Shroff G (2018). Online anomaly detection with concept drift adaptation using recurrent neural networks. Paper presented CoDS-COMAD '18 Proceedings of the ACM India Joint International Conference on Data Science and Management of Data. ACM, New York, USA. https://doi.org/10.1145/3152494.3152501

Singh A (2017) Anomaly Detection for Temporal Data Using Long Short-Term Memory (LSTM) (Master's thesis). Stockholm: Kth Royal Institute of Technology School of Information and Communication Technology.

Sun R, Cheng Q, Wang G, Ochieng WY (2017) A novel online data-driven algorithm for detecting UAV navigation sensor faults. Sensors 17(10):2243. https://doi.org/10.3390/s17102243

Ullah I, Fayaz M, Kim D (2019) Improving accuracy of the Kalman filter algorithm in dynamic conditions using ANN-based learning module. Symmetry 11(1):94. https://doi.org/10.3390/sym11010094

Vinayakumar R., Soman KP, Poornachandran P, S. Kumar SS (2018) Detecting Android malware using long short-term memory (LSTM). J Intell Fuzzy Syst 34(3):1277-1288. https://doi.org/10.3233/JIFS-169424

Wang B, Wang Z, Liu L, Liu D, Peng X (2019) Data-driven anomaly detection for UAV sensor data based on deep learning prediction model. Proceedings. Paper presented 2019 Prognostics and System Health Management Conference. IEEE; Paris, França. https://doi.org/10.1109/PHM-Paris.2019.00055

Wang L, Haofei Z, Jia S, Ling L, Sohail C (2013) An ARIMA-ANN hybrid model for time series forecasting. Syst Res Behav Sci 30(3):244-259. https://doi.org/10.1002/sres.2179

Yong D,Yuanpeng Z, Xu Y, Yu P, Datong L (2017) Unmanned aerial vehicle sensor data anomaly detection using kernel principle component analysis. Paper presented IEEE 2017 13th International Conference on Electronic Measurement and Instruments. IEEE; Yangzhou, China. https://doi.org/10.1109/ICEMI.2017.8265777

Yu H, Wilamowski BM (2016) Levenberg-Marquardt training. In: Wilamowski BM, Irwin JD, editors. 2th ed. The Industrial Electronics Handbook: Intelligent Systems, 46:1404–9. Boca Raton: CRC Press.

Zhang, J (2006) Improved On-line process fault diagnosis through information fusion in multiple neural networks. Comput Chem Eng 30(3):558-571. https://doi.org/10.1016/j.compchemeng.2005.11.002