UAV-FD: a dataset for actuator fault detection in multirotor drones*

Alessandro Baldini[1], Lorenzo D'Alleva[1], Riccardo Felicetti[1],
Francesco Ferracuti[1], Alessandro Freddi[1], Andrea Monteriù[1]

***Abstract*— Multirotor drones are equipped with propellers that may get damaged in flight in case of a collision with an obstacle or a rough landing. In view of safety-critical applications, such as flying over crowded areas or future passenger drones, being aware of a damaged actuator becomes essential to enhance system integrity. Therefore, in this paper we present a public dataset, namely UAV-FD, where real flight data from a multirotor under the effects of a chipped blade are collected. A conventional ArduPilot-based controller is employed, where the ArduPilot firmware is customized to increase the signal logging rate of selected variables, thus capturing information at higher frequencies. Moreover, the actual speed of each motor is measured and made available. Finally, we provide an illustrative fault detection strategy, based on MATLAB Diagnostic Feature Designer toolbox, to show how the dataset can be used and the blade chipping can be detected.**

SUPPLEMENTARY MATERIAL

The dataset and the code are available on Zenodo [1] at `https://doi.org/10.5281/zenodo.7648996` .

I. INTRODUCTION

As multirotor Unmanned Aerial Vehicles (UAVs) are pervading civilian and military activities, such as remote sensing [2], transportation [3], and patrolling [4], many authorities all over the world are introducing new UAV regulations in order to protect privacy and public safety [5]. In many countries, restrictions may apply when flying above people, in order to avoid accidents due to a crash. Preventing the malfunctioning of UAVs is then crucial to protect public safety when multirotors must be employed in crowded areas; this is the case, for example, of last-mile delivery and surveillance. Small-scale UAVs, in particular, are acknowledged to suffer a high loss rate [6] and, according to [7], 64% of accidents and incidents involving UAVs are due to equipment problems. Recent studies point out that actuator faults are a major issue [8]: in fact, if not tackled in time, they can cause loss of control in flight and, consequently, a threat to public safety. Moreover, in view of the future spreading of passenger drones (e.g., airtaxi), accidents prevention is of paramount importance to increase both safety and public acceptance [3].

In order to train and test Fault Detection (FD) algorithms, especially for signal-based FD methods, the main prerequisite is the availability of data. Hence, a public dataset that includes data from faulty multirotors could be helpful to accelerate the research in this sense. For this reason, in this paper we propose, a public dataset, namely "UAV-FD", to specifically deal with actuator faults on multirotor UAVs. In the literature, several datasets regarding drones have been proposed, however, most of them focus on artificial vision (see [9] and references therein), including human gestures [10] and behaviour [11], and object detection [12], [13]. On the contrary, only few datasets on UAVs are available as regards FD. In [14], a dataset for FD in fixed-wing UAVs is provided. Such dataset includes many kind of faults in different conditions. Please note that the actuator faults affecting multirotors and fixed-wing vehicles are, however, different: the former consist in motor and propeller faults, while the latter consist in stuck control surfaces. Hence, the features to detect them are radically different as well. In [15], a dataset for propeller FD in a hexarotor is provided. In particular, the authors of [15] focus on additional high-rate accelerometers installed on the UAV arms directly, while the on-board flight controller accelerometers are sampled at low rate.

In the proposed UAV-FD dataset, instead, we focus on the on-board sensors, embedded in the flight controller, because this kind of data is much likely to be available in any multirotor, and we show that such data are sufficient for the purpose of propeller FD. In the UAV-FD dataset, we also include the actual speed of the motors, that are measured by the Electronic Speed Controllers (ESCs), thus providing additional information for diagnostic purposes without the use of additional devices. Finally, differently from [15], in the proposed UAV-FD dataset, we inject faults, in turn, on each motor. Hence, the UAV-FD dataset can be employed to test fault isolation algorithms (i.e., identifying *which* propeller is faulty) as well.

We remark that data acquisition is performed without any additional device on the UAV, i.e., in the UAV-FD dataset the data is measured by sensors that are embedded in essential devices, i.e., in the flight controller or in the ESCs. In fact, every payload, including sensors and computing boards, impairs the vehicle maneuverability and depletes the battery faster, especially in the case of small-size UAVs. Moreover, additional on-board devices can easily exceed the cost of the UAVs itself [16]. For every UAV that is equipped with a flight controller, the Inertial Measurement Unit (IMU) data, including the measurement of linear acceleration and rotational speeds, can be made available without the need for additional sensors, together with additional estimated variables provided by sensor fusion operated by the attitude and heading reference system, commonly integrated

*This work was not supported by any organization

[1] A. Baldini, L. D'Alleva, R. Felicetti, F. Ferracuti, A. Freddi, and A. Monteriù are with the Department of Information Engineering, Università Politecnica delle Marche, Ancona, 60131 Italy, e-mail: a.baldini@univpm.it; lorenzodalleva97@gmail.com; r.felicetti@univpm.it; f.ferracuti@univpm.it; a.freddi@univpm.it; a.monteriu@univpm.it .

on-board. Among the available variables, the acceleration signals represent the richest source of information, as vibration signals are successfully employed to detect faults in rotating machinery [17], [18], bearings [19], clutches [20], and milling cutters [21]. However, according to the recent survey [22], acceleration signals are not commonly employed as a diagnostic feature for UAVs. Relevant exceptions are provided in [23], where wavelet packet decomposition on a quadrotor vibration data is employed to extract FD features, in [24], where a FD algorithm based on acceleration signals for a fixed-wing UAV is presented, and in [25], where acceleration measurements are employed to detect propeller faults. Hence, in proposed UAV-FD dataset, we increase the IMU sampling rate (as much as the flight controller permits it) to capture potentially interesting features. However, as the sensor set is mounted on the flight controller, the data is also influenced by flight commands and aerodynamic effects (e.g., frame drag and blade flapping), thus making FD challenging.

The paper is structured as follows. In Section II, we introduce the experimental setup, including the hardware specifications, the customization of the ArduPilot firmware to enhance data acquisition, the software setup, and finally the acquisition of experimental data with actuator (propeller) faults. The structure of the raw dataset and the procedure to load data are described in Section III. In Section IV, we propose a direct procedure to handle the dataset in order to use Matlab's Diagnostic Feature Designer (DFD) toolbox to extract diagnostic features and Matlab's Classification Learner (CL) to develop a FD classifier. The results are then detailed in Section V. Conclusions and some remarks on practical requirements for FD in UAVs end the paper in the last Section.

II. Experimental setup

In this Section, we introduce the hardware equipment that has been used to acquire real flight data from a multirotor drone. Then, we detail the firmware modifications that have been performed to improve the logging rate of the selected variables.

A. Hardware specifications

For the acquisition of the dataset, a hexarotor is preferred over a conventional quadrotor. In fact, hexarotors are inherently less sensitive to actuator faults, thanks to the actuator redundancy, so data can be acquired in safer conditions. The hexarotor (see Figure 1) has been internally assembled, starting from the DJI FlameWheel F550 frame. The flight controller is the PixHawk PX4 Cube Black, which embeds a MPU9250 [26] IMU. The battery is a 4-cells Tattu 25C LiPo, whose nominal capacity is 6700 mAh. Six T-Motor Air Gear 350 outrunner motors are used, together with the T-Motor T9545 self-tightening propellers (three ClockWise (CW) and three CounterClockWise (CCW)) that are made of plastic. Each motor is driven by a Tekko Holybro D-Shot 125 ESC, and the ESCs are configured to provide a motor speed measurement to the flight controller. The remote control is provided by a FrSky Taranis X8R receiver and a X9D on-board transmitter. The flight data of the present dataset are logged on a conventional Secure Digital (SD) card. As usual, the flight controller is mounted on a commercial anti-vibration platform to mitigate the impact of vibrations on the IMU measurements, resulting in a smoother attitude estimation and consequent more stable flight. Please note that, despite the presence of the anti-vibration platform, it is still possible to detect blade chipping from the available data, as later shown in the experimental results.

Fig. 1: The hexarotor UAV employed to acquire data.

B. Custom firmware and software setup

The ArduPilot flight controller provides logs selected variables from a predefined list to a SD card. The default log rate for many variables, including the ones coming from the IMU, is hardcoded. In the case of IMU data, such rate is 25 Hz, which is unsatisfactory because it does not allow to capture high-frequency dynamics, such as oscillations generated by the motors' rotation and blade unbalancing. In fact, the average propeller rotation frequency is in the order of 100 Hz, while the lower frequencies (from 0 to 5 Hz) are mainly determined by the pilot's maneuvers. According to Nyquist's sampling theorem, a sampling rate in the order of 200 Hz could be sufficient to get the main propeller dynamics, while a 400 Hz sampling rate may also allow to catch the second harmonic, that is related to blade unbalancing. To overcome this issue, the ArduPilot firmware can be edited. Starting from the latest (V4.3.1) ArduPilot version, that is available on the official GitHub repository[1], we have modified the `Copter.cpp` file in Eclipse[2]. It is sufficient to modify the scheduled rate of the task "twentyfive_hz_logging" from 25 Hz to the desired rate.

However, the logging rate cannot be increased arbitrarily. The ArduPilot code is split in tasks, which are managed by a scheduler, according to their (hardcoded) priority. The scheduler runs at 400 Hz, and each task is scheduled at a fixed rate that is less or equal to 400 Hz. The ArduPilot

[1]https://firmware.ardupilot.org/Copter/stable/CubeBlack/ , last access: 15-12-2022.

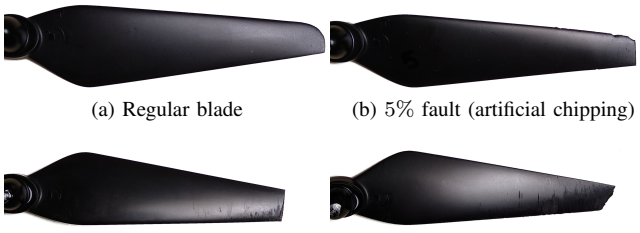
[2]https://www.eclipse.org/downloads/, last access: 15-12-2022.

scheduler cannot guarantee deadline compliance, as it is not a hard real time framework. If the available time in the current iteration is not sufficient to perform a task (given its estimated duration, that is hardcoded), the task is simply skipped and postponed to the next iteration. When the logging rate is too high, the logging rate is not stable and the secondary tasks with lower priority are affected as well. The target logging rate has been finally set to 350 Hz, as the result of a trial-and-error procedure.

Once modified, the source code can be compiled using Cygwin[3] and a suitable GCC compiler[4]. The compiled custom firmware has been loaded on the UAV using the Mission Planner Ground Station software. The usual further steps are needed before the flight, i.e., calibration of the IMU, radio configuration and calibration, and the ESCs calibration.

C. Dataset acquisition

The dataset is a collection of 18 flights under different fault conditions. Each flight is performed outdoor, in wind calm conditions and in manual flight mode, by a skilled operator. Each flight consists of a take-off, a free flight in a delimited square area, and landing. Also, please note that the battery has not always been recharged at the beginning of each flight, implying that each flight is performed with a different State Of Charge (SOC). The battery was replenished or substituted when the SOC could not guarantee another full flight. The choice of manual flight mode and variable battery SOC makes the dataset more challenging and realistic.



(a) Regular blade (b) 5% fault (artificial chipping)

(c) 10% fault (artificial chipping) (d) Chipped blade due to a crash

Fig. 2: Blade chipping.

The actuation fault is realized by damaging the propellers. In detail, as the propellers usually get chipped after a collision with an obstacle (see Figure 2d), we have artificially chipped four blades:

- a CW propeller with 5% fault,
- a CCW propeller with 5% fault (Figure 2b),
- a CW propeller with 10% fault (Figure 2c),
- a CCW propeller with 10% fault.

The regular blade length is 11 cm (Figure 2a), so we denote with 5% fault a propeller with a regular blade and a chipped blade whose remaining length is 10.45cm (Figure 2b), while we denote with 10% fault a propeller with a regular blade and a chipped blade whose remaining length is 9.9cm (Figure 2c). We remark that, in the hexarotor under investigation, both the 5% and 10% faults are relatively small, meaning that they have a minor impact on the manoeuvrability of the UAV.

[3]https://www.cygwin.com/install.html, last access: 15-12-2022.

[4]https://firmware.ardupilot.org/Tools/STM32-tools, last access: 15-12-2022.

III. Dataset description

The raw data consists of 18 .mat files, one for each flight, namely:

1) `NO_FAULT1.mat`
2) `NO_FAULT2.mat`
3) `NO_FAULT3.mat`
4) `NO_FAULT4.mat`
5) `NO_FAULT5.mat`
6) `NO_FAULT6.mat`
7) `FAULT_M1_5.mat`
8) `FAULT_M2_5.mat`
9) `FAULT_M3_5.mat`
10) `FAULT_M4_5.mat`
11) `FAULT_M5_5.mat`
12) `FAULT_M6_5.mat`
13) `FAULT_M1_10.mat`
14) `FAULT_M2_10.mat`
15) `FAULT_M3_10.mat`
16) `FAULT_M4_10.mat`
17) `FAULT_M5_10.mat`
18) `FAULT_M6_10.mat`

The files from 1) to 6) refer to flights in nominal conditions, i.e., without any fault. The files from 7) to 12) refer to flights with a 5% fault on a single motor, while files 13) to 18) are affected by a 10% fault on a single motor. The file name describes the fault condition by indicating both the faulty motor and the fault amplitude (e.g., `FAULT_M3_10.mat` refers to motor 3 that is affected by a 10% fault). The motors' numbering can be seen in Figure 1, i.e., clockwise from the top: motor 1, 4, 6, 2, 3, and 5. The IMU orientation can be derived as well from the figure: the front of the vehicle is oriented to the left of Figure 1.

A. Logged data

The raw data include the following log variables.

Measurements:

- IMU data for each of the three IMUs (variable names: IMU_0, IMU_1, IMU_2)
- ESC data, including measured speed, voltage, and current (ESC_0, ESC_1, ESC_2, ESC_3, ESC_4, ESC_5)
- Compass data (MAG_0, MAG_1, MAG_2)
- GPS data and accuracy (GPS_0, GPA_0)

Commands:

- Servo channel output data (RCOU, RCO2)
- Desired (and achieved) vehicle attitude and attitude rate (ATT, RATE)
- Arming status (ARM)

Estimations:

- EKF3 estimator data, variance, and timing for each IMU (XKF1_0, XKF1_1, XKF2_0, XKF2_1, XKF3_0, XKF3_1, XKF4_0, XKF4_1, XKF5_0, XKV1_0, XKV2_0, XKT_0, XKT_1)
- EKF yaw estimation (XKY0_0, XKY0_1, XKY1_0, XKY1_1)
- Vibration information for each IMU (VIBE_0, VIBE_1, VIBE_2), i.e., the estimated standard deviation of the measured accelerations
- Canonical vehicle position (POS)
- EKF quaternion rotation data (XKQ_0, XKQ_1)

Parameters:

- PID parameters (PIDA, PIDP, PIDR, PIDY)
- Parameter values (PARM)

Additional variables are available in the dataset, but they are not employed in the remainder of the paper: please refer to the ArduPilot documentation [27] for further details.

B. Data loading and pre-processing

As the data is divided into several files, we propose to perform the following steps to load data:

- load each raw log file (i.e., the .m log files listed in Section III-A),
- create a single timetable by synchronizing a subset selected variables (e.g., as in Section IV-A),
- define the fault codes,
- compute diagnostic features (e.g., as in Section IV-B),
- delete unreliable features (e.g., more than 25% of invalid values) and replace sporadic invalid values (infinity or not a number),
- split the feature data into training and testing (e.g., 30% holdout, with stratification with respect to the flight and a fixed random seed, in the proposed script).

The `main.m` script, that is enclosed with the dataset, performs the listed steps, and it finally saves the synchronized selected variables (*dataTable* variable), the table of data and features together (*data_feature_Table*), the features table (*feature_Table*), as well as the training (*feature_Table_Train*) and the testing (*feature_Table_Test*) partitions of the features table for repeatability.

All the aforementioned variables are available inside the `data_feature_Table_zoh.mat` file, that is shared together with the raw dataset for convenience. Otherwise, the entire offline procedure from the raw datasets to the extraction of the features takes 11 minutes (approximately 3 minutes for loading and synchronization and 8 minutes to generate the selected features) on a PC equipped with an Intel i9-11900k CPU, Corsair CMW32GX4M2D3600C18 (2x16GB DDR4) RAM, and a Samsung 970 Evo Plus SSD.

IV. Fault detection using Diagnostic Feature Designer

In this Section, we detail how the diagnostic features can be extracted for the purpose of FD. First of all, in Section IV-A, we select only a subset of meaningful variables to reduce the complexity of the classification algorithm. Then, we perform some pre-processing: denoising, synchronization, and removal of take-off and landing. The take-off and landing are removed because the propellers rotate at very different speeds with respect to regular flight, so such data are not consistent. Finally, in Section IV-B, we define a set of conventional time-based and frequency-based features to be extracted using the DFD.

A. Data synchronization

Each logged quantity has its own timestamp, expressed in microsecond, which implies that different variables are logged asynchronously, with different timestamps and a different logging rate. Moreover, the logging rate of the same variable is not strictly constant, as the flight controller is not a hard real time device. Resampling data with a fixed rate is suggested to facilitate the extraction of the features. The data synchronization function `synchronize.m`, that is available together with the dataset, takes each raw data file from Section III separately, and:

- keeps only the accelerometer and gyroscope measurements, the motor commands, the measured motor speeds and currents, the estimated attitude and the desired one, the estimated velocity, and the vibration data, for a total of 38 scalar variables,
- averages redundant data (three on-board IMUs, three VIBE variables, two EKF estimations) to reduce noise,
- synchronizes the data, either with zero order hold or interpolation, to obtain data records that are uniform in time (from 0 s to the end of each flight),
- removes data records related to the take off and the landing phases.

Employing the zero order hold for simplicity (i.e., it is faster and it shows marginal differences with respect to interpolation), the output of the procedure is then converted to a timetable (dataTable variable, that is available inside the `data_feature_Table_zoh.mat` file).

B. Feature calculation

Once the timetable is created, the DFD can be employed to generate a set of diagnostic features. It is sufficient to run the graphical DFD tool, select the data table (*dataTable* variable), and make sure that the fault condition (*faultCode* field) is correctly recognized. We have employed a frame based analysis with a 1 s window without overlapping, i.e., the FD algorithm is designed and tested on short time windows. The objective of the frame based analysis is twofold: to increase the number of runs (each 1s window becomes a train or a test record) and to investigate if a timely FD can be potentially performed online.

Once the features of interest are selected in the tool, we recommend to export the code from the tool and then to generate the features using the MATLAB code, instead of the graphical DFD tool. In fact, the size of the ensemble and the repetitiveness of manually extracting all the features for each variable make the MATLAB code more practical and efficient.

For the purpose, we provide the `diagnosticFeaturesfull.m` script, that is autogenerated exporting the code from the DFD. In particular, for each input variable and each 1 s frame, the following time-based features are derived: clearance factor, crest factor, impulse factor, mean, peak value, root mean square, signal-to-noise and distortion ratio, signal-to noise-ratio, shape factor, skewness, standard deviation, total harmonic distortion. Moreover, the power spectrum for each variable in each frame is calculated (approximating data with a 10th order autoregressive model and a forward-backward approach), and the following frequency-based features are obtained: first peak amplitude and frequency,

TABLE I: Accuracy of the trained classifiers using the entire feature table (573 features).

Classifier	Accuracy (test)
Ensemble (subspace discriminant)	98.2%
Ensemble (boosted trees)	96.2%
SVM (quadratic)	95.7%
SVM (cubic)	95.7%
SVM (linear)	94.0%

second peak amplitude and frequency, band power. The outputs of this procedure are a table of data and frame based features (*data_feature_Table* variable) and a table of features only (*feature_Table*). Both are available inside the `data_feature_Table_zoh.mat` file.

V. Fault Detection Results

In order to train a classifier on the training data, the Classification Learner (CL) can be exploited. The *feature_Table_Train* data is imported in the CL, making sure that the *Ensemble ID* and the *faultCode* are removed from the predictors and the faultCode is the desired response (i.e., the label). To limit the overfitting, a 5-fold cross-validation is performed on the training data. A separate test set (*feature_Table_Test*) is used to check the performances of the trained classifiers.

We select all of the available classifiers in the CL for training, as their training takes approximately two minutes. Table I reports the accuracy of the best classifiers in the test set.

The classifiers that show the best accuracy ($> 96\%$) are the subspace discriminant and boosted trees classifiers; they are followed by the Support Vector Machines (SVMs) (quadratic, cubic, and linear, respectively).

Simpler classifiers, such as the coarse binary classification tree, reach a 85.4% accuracy; however, the computational load of the coarse tree is dramatically reduced in view of online implementation. In fact, analyzing the binary classification tree, we note that only the following variables are exploited:

- mean of VIBE on the y axis
- mean of VIBE on the x axis
- peak value of VIBE on the y axis
- band power of commanded Pulse Width Modulation (PWM) on motor 6

Please note that the VIBE variables are available online, and the mean and the peak value in the 1s window can be easily computed. On the other hand, calculating the band power for the acceleration and the PWM command on motor 6 requires spectral estimation in each time window, whose computational complexity could be still infeasible on-board.

The confusion matrix in the test set, adopting the best classifier (i.e., the subspace discriminant), is proposed in Figure 3, while the one for the coarse classification tree is reported in Figure 4.

In order to reduce the number of input features, thus decreasing the computational effort, the Principal Component Analysis (PCA) can be employed. In Table II and Table III,

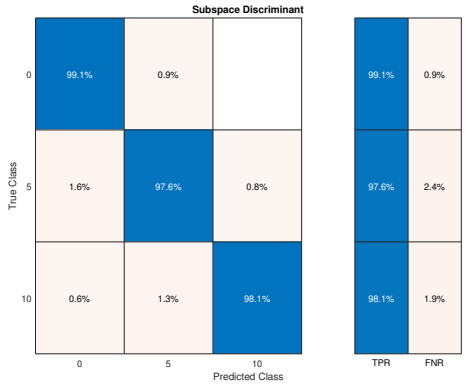


Fig. 3: Confusion matrix with a subspace discriminant classifier using the entire feature table.

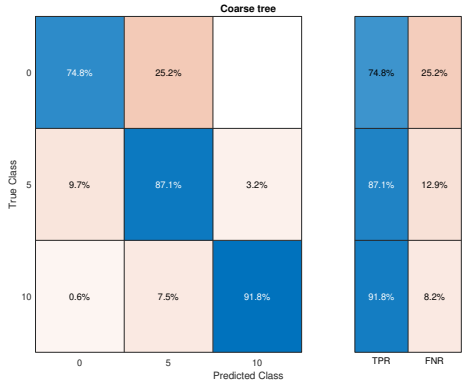


Fig. 4: Confusion matrix with a coarse classification tree using the entire feature table.

we report the performances when the explained variance in PCA is set to 99.9% and 99%, respectively. The number of input features is reduced, from 573 to 51 for 99.9% explained variance, and from 573 to 16 for 99% explained variance.

As the feature space is poorer, in both cases the best results come from non-linear classifiers, such as the SVMs with quadratic, cubic, or gaussian kernels, as well as ReLU neural networks, because they are able to capture more complex relations; nonetheless, the accuracy drops to 85.4% and 75.6%, respectively for 99.9% and 99% explained variance.

Alternatively, a preliminary feature selections by means of ANalysis Of VAriance (ANOVA) can be performed to reduce the number of features, in place of PCA. In Table IV (and Table V), we report the performances when the best 51 variables (16 variables, respectively) are kept.

As in the case of PCA, the best results come from non-

TABLE II: Accuracy of the trained classifiers using PCA and 99.9% explained variance (51 features).

Classifier	Accuracy (test)
SVM (cubic)	85.4%
SVM (fine gaussian)	85.2%
SVM (quadratic)	84.9%
KNN (cosine)	80.9%
Neural network (Wide)	80.9%

TABLE III: Accuracy of the trained classifiers using PCA and 99% explained variance (16 features).

Classifier	Accuracy (test)
Neural network (narrow)	75.6%
Neural network (wide)	74.1%
SVM (quadratic)	72.1%
Neural network (medium)	71.9%
Neural network (trilayered)	70.6%

TABLE IV: Accuracy of the trained classifiers using the best 51 features according to ANOVA.

Classifier	Accuracy (test)
SVM (quadratic)	98.5%
SVM (cubic)	97.7%
Neural network (wide)	97.2%
SVM (medium gaussian)	97.0%
Neural network (medium)	95.7%

linear classifiers. However, despite the same number of input features is employed, the accuracy with ANOVA (98.5% and 96.5%) is much better than the one with PCA (85.4% and 75.6%), and it is comparable with the accuracy using the entire set of features (98.2%). The confusion matrix in the test set, adopting the best classifier (i.e., the quadratic SVM) on the best 51 features according to ANOVA, is proposed in Figure 5.

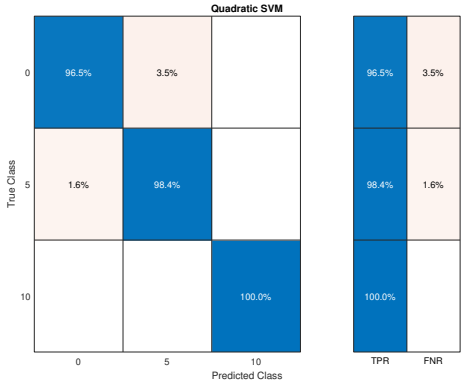


Fig. 5: Confusion matrix with a quadratic SVM classifier using the best 51 features according to ANOVA.

VI. Conclusions

In this paper, we have presented a novel dataset, namely UAV-FD: the objective is to ease the design and the testing of FD algorithms for multirotor vehicles. Along with the raw dataset and the pre-processed data, we provide the MATLAB code to perform data loading and pre-processing. Finally, to disclose the potential of the dataset, we have extracted common diagnostic features from a subset of the available variables, making use of the DFD, and we have trained several classifiers using the CL. The best accuracy in the test set (98.5%) has been reached by a quadratic SVM classifier, trained on the best 51 features according to ANOVA.

In view of practical applications of FD strategies for multirotor drones, the following questions should be addressed:

- can the false positive rate be reduced to meet the standards in the aeronautical field, while retaining a good accuracy?
- can the strategy be run on the flight controller, or additional computational power is needed?
- is latency of the FD algorithm satisfactory?

In fact, we point out that avoiding false positives is a priority in the aeronautical field, as false positives usually trigger safety procedures that must be avoided when unnecessary (e.g., reconfiguration, emergency landing). Also, the FD algorithm should run online, so two problems arise: computational power and latency. To cope with computationally heavy algorithms, external computational power could be installed on-board; alternatively, the telemetry could be employed to communicate data to a ground station that is in charge of running the FD algorithm. In any case, the FD should be timely enough to deal with the fault, especially in case of severe faults, to recover the vehicle before a crash occurs.

Finally, we highlight that, in this paper, we have dealt with fault detection; we deem fault isolation is far way harder. In principle, the information on the motor speed could be exploited to infer which motor is faulty, together with the drone attitude error or the vibration signals. However, whether attaining satisfactory results in fault isolation from the available data is possible or not is still an open question.

TABLE V: Accuracy of the trained classifiers using the best 16 features according to ANOVA.

Classifier	Accuracy (test)
SVM (cubic)	96.5%
SVM (quadratic)	96.2%
KNN (weighted)	96.0%
SVM (medium gaussian)	95.2%
Neural network (trilayered)	95.2%

References

[1] European Organization For Nuclear Research and OpenAIRE, "Zenodo," 2013. [Online]. Available: https://www.zenodo.org/

[2] M. Longhi and G. Marrocco, "Ubiquitous flying sensor antennas: Radiofrequency identification meets micro drones," *IEEE Journal of Radio Frequency Identification*, vol. 1, no. 4, pp. 291–299, 2017.

[3] R. Kellermann, T. Biehle, and L. Fischer, "Drones for parcel and passenger transportation: A literature review," *Transportation Research Interdisciplinary Perspectives*, vol. 4, p. 100088, 2020.

[4] A. Restas *et al.*, "Drone applications for supporting disaster management," *World Journal of Engineering and Technology*, vol. 3, no. 03, p. 316, 2015.

[5] C. Stöcker, R. Bennett, F. Nex, M. Gerke, and J. Zevenbergen, "Review of the current state of uav regulations," *Remote sensing*, vol. 9, no. 5, p. 459, 2017.

[6] H. Shraim, A. Awada, and R. Youness, "A survey on quadrotors: Configurations, modeling and identification, control, collision avoidance, fault diagnosis and tolerant control," *IEEE Aerospace and Electronic Systems Magazine*, vol. 33, no. 7, pp. 14–33, 2018.

[7] G. Wild, J. Murray, and G. Baxter, "Exploring civil drone accidents and incidents to help prevent potential air disasters," *Aerospace*, vol. 3, no. 3, p. 22, 2016.

[8] A. Bondyra, M. Kołodziejczak, R. Kulikowski, and W. Giernacki, "An acoustic fault detection and isolation system for multirotor uav," *Energies*, vol. 15, no. 11, p. 3955, 2022.

[9] P. Mittal, R. Singh, and A. Sharma, "Deep learning-based object detection in low-altitude uav datasets: A survey," *Image and Vision computing*, vol. 104, p. 104046, 2020.

[10] A. G. Perera, Y. Wei Law, and J. Chahl, "Uav-gesture: A dataset for uav control and gesture recognition," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[11] T. Li, J. Liu, W. Zhang, Y. Ni, W. Wang, and Z. Li, "Uav-human: A large benchmark for human behavior understanding with unmanned aerial vehicles," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 16 266–16 275.

[12] A. Antonini, W. Guerra, V. Murali, T. Sayre-McCord, and S. Karaman, "The blackbird uav dataset," *The International Journal of Robotics Research*, vol. 39, no. 10-11, pp. 1346–1364, 2020.

[13] B. Kellenberger, D. Marcos, and D. Tuia, "Detecting mammals in uav images: Best practices to address a substantially imbalanced dataset with deep learning," *Remote sensing of environment*, vol. 216, pp. 139–153, 2018.

[14] A. Keipour, M. Mousaei, and S. Scherer, "Alfa: A dataset for uav fault and anomaly detection," *The International Journal of Robotics Research*, vol. 40, no. 2-3, pp. 515–520, 2021.

[15] S. Gururajan, K. Mitchell, and W. Ebel, "Flights of a multirotor uas with structural faults: Failures on composite propeller (s)," *Data*, vol. 4, no. 3, p. 128, 2019.

[16] B. Wang, Y. Shen, and Y. Zhang, "Active fault-tolerant control for a quadrotor helicopter against actuator faults and model uncertainties," *Aerospace Science and Technology*, vol. 99, p. 105745, 2020.

[17] Z. Peng, W. T. Peter, and F. Chu, "An improved hilbert–huang transform and its application in vibration signal analysis," *Journal of sound and vibration*, vol. 286, no. 1-2, pp. 187–205, 2005.

[18] P. Gangsar and R. Tiwari, "Signal based condition monitoring techniques for fault detection and diagnosis of induction motors: A state-of-the-art review," *Mechanical systems and signal processing*, vol. 144, p. 106908, 2020.

[19] D.-T. Hoang and H.-J. Kang, "A survey on deep learning based bearing fault diagnosis," *Neurocomputing*, vol. 335, pp. 327–335, 2019.

[20] G. Chakrapani and V. Sugumaran, "Transfer learning based fault diagnosis of automobile dry clutch system," *Engineering Applications of Artificial Intelligence*, vol. 117, p. 105522, 2023.

[21] T. Wu and K. Lei, "Prediction of surface roughness in milling process using vibration signal analysis and artificial neural network," *The International Journal of Advanced Manufacturing Technology*, vol. 102, no. 1, pp. 305–314, 2019.

[22] G. K. Fourlas and G. C. Karras, "A survey on fault diagnosis and fault-tolerant control methods for unmanned aerial vehicles," *Machines*, vol. 9, no. 9, p. 197, 2021.

[23] X. Zhang, Z. Zhao, Z. Wang, and X. Wang, "Fault detection and identification method for quadcopter based on airframe vibration signals," *Sensors*, vol. 21, no. 2, p. 581, 2021.

[24] A. Benini, F. Ferracuti, A. Monteriù, and S. Radensleben, "Fault detection of a vtol uav using acceleration measurements," in *2019 18th European Control Conference (ECC)*. IEEE, 2019, pp. 3990–3995.

[25] B. Ghalamchi and M. Mueller, "Vibration-based propeller fault diagnosis for multicopters," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2018, pp. 1041–1047.

[26] TDK InvenSense, "Mpu-9250, nine-axis (gyro+accelerometer+compass) mems motiontracking™ device," 2016, https://invensense.tdk.com/download-pdf/mpu-9250-datasheet/.

[27] "Ardupilot onboard message log messages," https://ardupilot.org/copter/docs/logmessages.html, accessed: 2023/01/15.