Real-time Fault Detection on Small Fixed-Wing UAVs using Machine Learning

Murat Bronz, Elgiz Baskaya, Daniel Delahaye, Stéphane Puechmorel

Real-time Fault Detection on Small Fixed-Wing UAVs using Machine Learning

Murat Bronz[†], Elgiz Baskaya[*†], Daniel Delahaye[†], and Stephane Puechmorel[†]
[*]ENAC - Groupe ADP - Sopra Steria, Systemes de drones
[†]ENAC, University of Toulouse, France
Email: murat.bronz@enac.fr

***Abstract*—In this study, we have highlighted the main challenges of real-time fault diagnosis on small scale fixed-wing UAVs. The feasibility of real-time fault prediction has been shown in real flight conditions experiencing noisy measurements, communication limitations, and wrapped wing structure that breaks the geometric symmetry. A total of eleven flight logs have been recorded and shared publicly for future potential use by other researchers on fault and anomaly detection. Our proposed method uses a data driven algorithm, SVM, in order to classify the behavior of the vehicle in nominal flight phase and faulty phase. Feasibility of a basic binary classification is shown, despite the well-known over-fitting problem caused by limited data. We have shown that geometrical imperfections that are common in small UAVs can cause particular effects on the prediction performance, and we used it in our advantage to improve the detection on multi-class classification. The SVM algorithm with proposed feature trajectories was capable to detect variation of loss of control effectiveness faults up to an accuracy of 95% in real flights. The data-set and all related programs can be downloaded from (`https://github.com/mrtbrnz/fault_detection`).**

***Index Terms*—Machine Learning, Real-time Fault-Detection, Real-time Fault-Diagnosis, SVM, Paparazzi, UAV, Drones**

I. Introduction

Future generations of flight control systems are likely to be more adaptive and intelligent to cope with the safety and reliability requirements. Addressing more and more complex missions will necessitate intelligent approaches to address the emergencies that will inevitably arise for all classes of UAV operations as defined by EASA (European Aviation Safety Agency). Not only the current increase in the use of UAVs and but also the projected surge of use in the future makes the real-time diagnosis of these systems a priority. Therefore it is critical to have fault-tolerant flight control systems in these vehicles. Small UAVs are in the frontage for these new technology adoptions. However, their hardware limitations point to the utilization of analytical redundancy, rather than to the usual practice of hardware redundancy seen in manned aviation.

Most desirable fault-tolerant control would have no performance degradation in any type of fault. Of course there are limitations to this fantasy, however efforts have been started long ago with *Self-Repairing Flight Control Systems* [1] in 1984 with two main objective in mind: 1) Re-configurable control system , meaning that the controller will modify and adapt itself to the changing conditions, and 2) is to have on-line diagnostics to identify the failures.

This work is partially funded by Chaire-Drone Systems.

Since then several researchers worked on these subjects: [2] discusses multi-variable adaptive control algorithms that can reconfigure the controller in real-time from the identified dynamics. [3] re-configures the controller based on an eigenstructure assignment technique. [4] uses an adaptive controller, which does not rely on the detection and isolation algorithm, and identifies the actuator failures as a change in the parameters in the failure model. Venkataraman [5], [6] showed the possibility of flying safely with only one aerodynamic control surface. Later Bauer and Venkataramman [7] used Multiple Model Adaptive Estimation (MMAE), which requires detailed modeling of the system dynamics including the actuator dynamics, and handling of the time delays. Many failure detection method relies on the model of nominal (unfailed) system, any discrepancy with this model and the reality might lead to a false detection.

In the core of most of the real-time identification methods, or adaptive control methods lies the constant excitation requirement problem. There must be a non-zero input to the system, which is not satisfied during a level cruise flight for example. Therefore in [8], a small sinusoidal input has been added to the system to ensure sufficient excitation.

Understandably, Re-configurable control systems rely heavily on the successful detection of the faults, or estimation of the dynamics of the vehicle. Failure of such will result in degradation of the performance (sometimes even catastrophic) rather then improvement. Therefore, a successful detection and diagnosis method is always desired.

A. Related Work

Detection strategy can be mainly based on two methods: 1) Model-based, and 2) Data-driven.

There are a variety of different approaches for model-based fault detection and diagnosis in the literature. Detecting sensor and actuator faults via state estimation, utilizing an EKF is applied to a F-16 model in [9]. Parameter identification via H_∞ filter is used to indicate icing in [10]. Another method to detect and isolate actuator/sensor faults is the multiple model adaptive estimation (MMAE) method [11]. In MMAE [12], multiple Kalman filters are used.

A drawback of the model-based approaches is that they require an accurate model of the aircraft for successful detection. In a small UAV system, which is susceptible to various uncertainties/disturbances and usually lacks an accurate model, using a model-based approach might fail. Furthermore, a mathematical model of a UAV is built within the flight envelope, and does not necessarily describe the possible dynamics invoked by an on-board failure.

In data-driven methods, a detailed knowledge about the internal dynamics of the system is not necessary. The data available is the source of information with regard to the behavior of the system. Supervised learning, which requires labeling the fault cases previously in the training data, is usually used for data-centric fault detection [13]. In case of an unlabeled fault, the result is expected as a probability distribution of the available normal modes, identified fault labels and a probable unknown fault.

In [14], the author applies an SVM classifier to detect and diagnose faults in drones with an actuator failure. [15] continuously regresses an estimation of the aircraft actuator health. They have used a deep learning structure that comprise one dimensional CNN following by LSTM layers. The health information is embedded in the NDI based control system. [16] also proposed a deep learning based fault detection and identification method using CNN and LSTM. Sindhwani et al. [17], while using a highly redundant VTOL vehicle, learned the dynamics estimate function from 5000 mission flights and used this function as a detector for anomalies on a new set of 5000 mission flights. They have captured three different anomaly types.

[18] compared model-based and data-driven fault detection methods by using real-flight data gathered from flight tests.

An important initiative has been made by [19], where their *ALFA Flight data-set* has been open-sourced. Furthermore, in [20], they present the good use of their data-set for online anomaly detection. We will be continuing this initiation by open-sourcing our flight data-set and all related programs including inference too.

B. Overview and Contribution

Main contributions of this work can be listed as :

- Most of the previous work used synthetic data to show the feasibility of fault detection and reconfiguration of the controller in order to isolate the faults, we contribute with real-flight data and in-flight prediction results.
- We also offer to validate this by sharing our flight test data from outdoor experiments which includes more than 6 hours of flight with at least 2 hours of faulty data. The data-set and all related programs can be downloaded from `https://github.com/mrtbrnz/fault_detection`.
- In this work, the effect of change in the environment of the agent on learning performance has been discussed by considering flights in windy and no wind conditions.
- Also, not only the changes of the environment, but also the imperfections of the

Paper is organized as following: In Section II, we present the hardware setup and autopilot software system that is used for the experiments. In Section III, we present the collected data-set and different types of log files. Section IV describes, the important properties of the selected classification algortihm SVM, actuator failure model, and generation of feature trajectories.

Impatient readers can directly refer to Section V for the results, however a quick look at Section IV will help to get familiar with the used methodology and decisions.

II. Flight Test Platform

A. Hardware Setup

An overview of the complete flight test system is shown in Figure 1. A small fixed-wing UAV, easy to recover from upset conditions, has been used for the data gathering and real-flight experiments. A laptop running Unix/Linux for Ground Control Station is used to control the autonomous mission and also to inject the faults manually during flight. An X-Bee radio modem for telemetry and datalink communication was sufficient as the flight tests were done within 0.5km radius. An RC-Transmitter is used to ensure the safely recovering of the vehicle in case of loss of control in autonomous mode with severe faults injected. Raspberry Pi-Zero is used as a companion board. It was connected to the autopilot via serial UART peripheral, and it was mainly responsible of making the In-flight prediction and keeping a secondary on-board log. The full list of specifications of the fixed-wing aircraft is shown in Table I.

Fig. 1. Complete flight test system that is used during the data set collection and real-time detection flights. Note the small XBee radio modem attached on the side of the computer screen, used for telemetry communication between the aircraft and the Ground Control Station

B. Software

Throughout the whole flight tests, we have used the Paparazzi Autopilot system [21]. It is an open-sourced project started back in 2003 and used by several research groups, academics, and hobbyist.

Being one of the first open-source autopilot systems in the world, *Paparazzi* covers all three segments: ground, airborne, and the communication link between them. *Paparazzi* has also its own complete flight plan language, where the user

TABLE I
AIRFRAME SPECIFICATIONS

Specification		Units
Wing Span	1.2	[m]
Surface Area	0.28	$[m^2]$
Mass	0.75	[kg]
Battery Capacity	30	[Wh]
Flight speed	12	[m/s]
Flight time	60	[min]
Components		
Motor	T-Motor 2208/18 - Kv 1100	
Autopilot	Paparazzi Chimera v1.0 [1]	
GPS	U-Blox M8N	
Companion board	Raspberry Pi Zero v 1.3	

can define any possible trajectory using existing commands, such as circle, line, hippodrome, figure-eight, survey, etc. Additionally, any function written in C language can be called from the flight plan and executed. This opens up a lot of application possibilities, such as triggering a navigation procedure via a sensor output. Its integrated ground control station permits the easy use of different navigation routines and also interaction with the aircraft while in flight, such as injecting the faults, or retrieving them back. Thanks to its middle-ware communication bridge called Ivy-Bus, external software can be directly connected with the publish and subscribe method to the ground segment, without needing to modify on code. This permits the use of custom simple modules in the flight experiments.

The modifications that were necessary to make the flight tests with injected faults were : first, setup a small module that can send real-time faults from GCS to the aircraft; and second, editing the controller on-board so that injected faults configure the servo actuators without informing the controller commands, so that it really works like a hardware fault.

In any phase of the flight, once the manual control is activated from the Safety RC-Transmitter, the injected faults are cancelled and the safety pilot can control the aircraft normally.

III. COLLECTED DATA-SET

We are well aware that obtaining data from flights, especially for flights with faults, is difficult, and requires an established infrastructure on the hardware, software, and human resources, as well as flight permissions. Therefore, as one of the main contribution of this work, we have open-sourced all of the flight test logs publicly which can be downloaded from : `https://github.com/mrtbrnz/fault_detection`.

The different types of logs with their description are listed below:

GCS Log: Those are the quantities logged inside the ground control station computer which receives the messages from aircraft via telemetry.The use of small communication devices and very portable small antennas makes the system vulnerable to connection losses, therefore the GCS log can obtain several missing information during the flight. Additionally, as the bandwidth of the telemetry is limited, the frequency of the messages are low. Main objective of the GCS log is to inform the operator during the flight test, thus the data-set involved should mainly considered for that purpose.

On-board SD Card Log: The limitations on the bandwidth and the low frequency are handled by logging the data on-board the autopilot, in a separate real-time operation system thread at a high frequency (i.e. 100Hz). These quantities can be defined before the flight with their corresponding frequency. The operator is required to download the binary log file after the flight and convert it to readable text file. Each message contains a time-stamp which is used to synchronize the whole data afterwards.

On-board Companion Board Log: Currently, the companion board that is used does not have any sensor to obtain the state features that is used for the detection algorithm. Therefore, it obtains the sensory information from the autopilot through a cabled serial connection link. During the experiments only the linear accelerations, angular rates, and desired control surface deflections have been sent from autopilot to the companion board.

IV. FAULT DETECTION METHODOLOGY

The main objective of this work is to show the difficulties and challenges that needs to be overcome during a real-time fault detection with real measurements and real-life limitations, such as noisy data, computational limitations, connection problems, etc... Furthermore, what is even more interesting is to see if a classical machine learning algorithm would achieve the required performance within the complications of the real world. Therefore, Support Vector Machine (SVM), a supervised classification algorithm, has been implemented to the problem of fault diagnosis in order to predict faults on-line.

A. Support Vector Machine (SVM)

Support Vector Machine is one of the most popular *off-the-shelf* frameworks for supervised learning. As mentioned in [22], the most forthcoming properties of SVMs are :

1) SVMs constructs a decision boundary that has the largest distance to the example points, called **Maximum margin separator**. This helps on generalization.
2) SVMs can use so-called **kernel-trick**, which is to represent the linear separator in a high-dimensional space so that it can represent a non-linear separator in the original space.
3) SVMs are a **non-parametric method**, they retain training examples and potentially need to store them all. However, in practice only a small fraction of the examples (support vectors) suffice which gives SVMs the chance to combine the advantages of parametric and non-parametric models. With that, they can represent complex functions while being resistant to over-fitting.

After a SVM classifier is trained, the performance of the classification is evaluated with a variety of different metrics, depending on the nature of the classification problem. Also,

during tuning, in which the hyper-parameters of the classifiers are selected to give the best performance, requires a performance index.

A brief reminder about the performance metrics that are used in this work :

- **Accuracy** is the ratio between correctly classified points to total number of points.
- **Precision** is the ratio between correctly predicted faults to the total number of detection (both true and false). It indicates how reliable is the model when it detects a fault.
- **Recall** is the ratio between correctly predicted faults to the total number of existing fault points. It indicates how reliable is the model in detecting faults.
- **F1-score** is the harmonic mean of the precision and recall.
- **Matthews Correlation Coefficient** takes into account true and false positives and negatives and is generally regarded as a balanced measure which can be used even if the classes are of very different sizes [23].
- **Receiver Operating Characteristics Curve** is a graph showing the performance of a classification model at all classification thresholds.

B. Modeling of Actuator Failure

Common actuator failure cases for an aircraft include: 1) locked-in place, 2) floating around trim, 3) hard-over, 4) loss of effectiveness. We have used the same actuator fault model proposed in [4].

$$\boldsymbol{u}_{app} = \boldsymbol{D}\boldsymbol{u}_{com} + \boldsymbol{E} \tag{1}$$

where, $\boldsymbol{u}_{com}$ is the commanded control deflection from the autopilot to the actuators and $\boldsymbol{u}_{app}$ is the applied control deflection (i.e final movement of the surface). For example, by the use of (1), actuator loss of effectiveness can be modeled by changing $\boldsymbol{D}$, locked-in place can be modeled by making $\boldsymbol{D} = 0$ and changing the $\boldsymbol{E}$ offset to the locked-in angle.

The airframe used in this work has only two aerodynamic control surfaces, and a motor. We have only concentrated on the faults injected to the aerodynamic control surfaces. Especially, the loss of effectiveness of the control surfaces as flying with such a type fault is still possible even with an under-actuated platform as ours. Hard-over type faults typically results in a sudden spin towards the ground, and can not be controlled at all. If we write (1) in ?vector?? form

$$\begin{bmatrix} u_{app_1} \\ u_{app_2} \end{bmatrix} = \begin{bmatrix} d_1, 0 \\ 0, d_2 \end{bmatrix} \begin{bmatrix} u_{com_1} \\ u_{com_2} \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} \tag{2}$$

where, $()_1$ is for right wing control surface (i.e., right elevon) and $()_2$ is for left wing control surface (i.e., left elevon).

C. Generation of Feature Trajectories

It is usually desirable to reduce the number of input features to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model by removing the unrelated information. In this work, it is decided that a basic feature list will suffice to show the feasibility of the proposed idea. Therefore, without a complex *Feature Engineering*, we directly used the linear accelerations (a_{xyz}), angular rates (ω_{xyz}) and autopilot commanded controls (u_{com_1} , u_{com_2}) for the two aerodynamic actuators, adding up to 8 features ($\boldsymbol{X}^{'}$)in the first place.

$$\boldsymbol{X}^{'}(t) = [a_{x_t}\, a_{y_t}\, a_{z_t}\, \omega_{x_t}\, \omega_{y_t}\, \omega_{z_t}\, u_{com1_t}\, u_{com2_t}] \tag{3}$$

Information on temporal dynamics is added by concatenating 20 of those 8-element basic feature lists.

$$\begin{aligned} \boldsymbol{X}^{'}(t) &= [a_{x_t}\, a_{y_t}\, a_{z_t}\, \omega_{x_t}\, \omega_{y_t}\, \omega_{z_t}\, u_{com1_t}\, u_{com2_t}] \\ \boldsymbol{X}^{'}(t-1) &= [a_{x_{t-1}} \dots \omega_{x_{t-1}} \dots u_{com1_{t-1}}\, u_{com2_{t-1}}] \\ &\vdots \\ \boldsymbol{X}^{'}(t-19) &= [a_{x_{t-19}} \dots \omega_{x_{t-19}} \dots u_{com1_{t-19}}\, u_{com2_{t-19}}] \end{aligned}$$

Finally we obtain a 160-element vector ($\boldsymbol{X}$) as an input feature trajectory to the SVM algorithm.

$$\boldsymbol{X}(t) = [\boldsymbol{X}^{'}(t-19) \dots \boldsymbol{X}^{'}(t-2)\, \boldsymbol{X}^{'}(t-1)\, \boldsymbol{X}^{'}(t)] \tag{4}$$

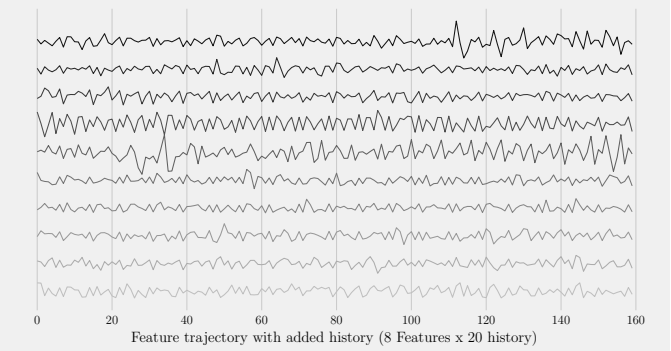

Fig. 2. A set of feature trajectories, starting from top (dark black), the newest trajectory, going down (light grey), the older trajectories. Each trajectory is an input for the inference model, and has 160 points.

SVMs assume that all features are centered around 0 and have variance in the same order. In order to prevent the domination of any single attribute in the feature vector, we standardize features by removing the mean and scaling to unit variance (using `sklearn.preprocessing.StandartScaler`). Centering and scaling happen independently on each attribute of the feature vector by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used later during inference pre-processing phase.

Figure 2 shows a set of feature trajectories, starting from top (dark black), the newest trajectory, going down (light grey), the older trajectories. Each trajectory has 160 points. Sensory information is passed to the inference computer (i.e., *RaspberryPi-Zero*) at a frequency of 10Hz, piling up the buffer (i.e., 20 time history) takes 2 seconds. We call the inference prediction only once the feature trajectory buffer is completed, so every 2 seconds only. A ring buffer can be used to increase the prediction frequency. The prediction call in the inference takes about 0.06 seconds of computation time, so increasing the prediction frequency upto 10Hz can be possible for the

current hardware. Using a more powerful companion board such as *Jetson Nano/Xavier-NX* will increase the prediction frequency at least one order of magnitude.

V. Prediction Performance in Real-Flights

In most in-flight prediction applications, the model would have been already trained with previous data and uploaded to the inference computer. During the prediction phase, the inference model will see the new data stream for the first time, and its performance will highly rely on its generalization capability. We will highlight how the day to day change in feature trajectory influences the predictions, and more importantly, how much trust one can have in real-life conditions (such as changing wind conditions) for a model that is trained with a limited data-set.

In the following subsections, we will demonstrate the in-flight prediction performance of SVM algorithm used for binary (nominal and faulty) and multi-class classification problem (nominal and 9 different variation of control effectiveness). Emphasis will be firstly on the external disturbance, such as wind, and then we will show how the imperfections in real life, such as a twisted wing, contribute to the prediction performance.

A. Effect of External Disturbance (Wind)

Flights made on 12^{th} and 13^{th} of July 2020 includes 15 minutes (between 500^{th} and 1400^{th} seconds) of similar fault pattern of reduced efficiency of right control surface ($d_1 = 0.3$).

From the above defined portion of 12^{th} of July 2020 flight, we have generated the feature trajectories as explained in Section IV-C. Data has been split into two parts: 80% *training data*, and 20% *test data*. An SVM algorithm that uses Radial Basis Function kernel is trained and optimized with grid search method using the *training data*. The best hyper-parameters are found to be $C = 5$ and $\gamma = 1/160$ (γ ='auto' takes $1/nb_{features}$ as default) for this flight. Table II shows the classification performance report of the optimized model that is evaluated on the separate *test data*, and yields Matthews Correlation Coefficient $MCC = 0.98$.

TABLE II
12^{th} July Flight Binary Classification Report

	Precision	Recall	F1-score	Support
Nominal	0.99	0.99	0.99	843
Faulty ($d_1 = 0.3$)	0.99	0.99	0.99	956
Accuracy			0.99	1799
Macro avg	0.99	0.99	0.99	1799
Weighted avg	0.99	0.99	0.99	1799

Performance metrics that are obtained encouraged us to use this model for the real-time prediction on a flight the following day. However, when the model is used on the day of 13^{th} of July, as can be seen from the Table III, and from the calculated $MCC = 0.39$, the real-time predictions on-board the vehicle was not acceptable.

TABLE III
13^{th} July Flight (with 12^{th}'s Model) Binary Classification Report

	Precision	Recall	F1-score	Support
Nominal	0.93	0.43	0.59	293
Faulty ($d_1 = 0.3$)	0.47	0.94	0.63	157
Accuracy			0.61	450
Macro avg	0.70	0.68	0.61	450
Weighted avg	0.77	0.61	0.60	450

The real-time prediction performance is shown in Figure 3, where the *Applied Faults* are shown for the ground-truth, and on the bottom, instantaneous predictions are shown in blue. Additionally in shaded green the prediction state is shown, which is basically a function that switches the state only after receiving three consecutive equal predictions. For example, while the predicted state is *nominal*, and the model predicts a *fault*, the state will not immediately change to *faulty*, unless two more consecutive *faults* are predicted. This will prevent rapid changing of the predicted state, making it more stale for the false predictions, but also less agile in the case of true predictions.

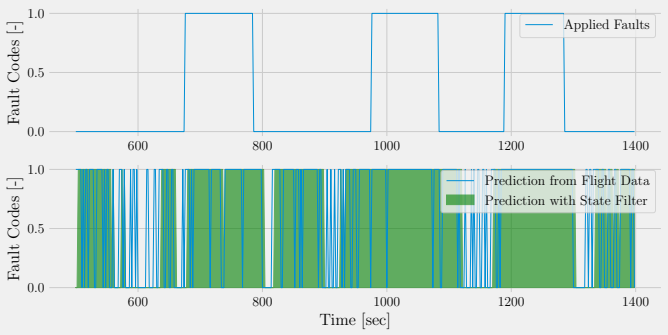

Fig. 3. In-flight prediction performance of the model that is trained on 12^{th} of July's data during 13^{th} of July flight.

One of the main possible causes for such a bad performance is the memorization of the data during the training and not being able to generalize well for the new test data, which is also known as over-fitting.

However, in the presented case, there is another source of problem coming from the data, which is the amount of excitation in the signal coming from external disturbance. Figure 4 shows the angular rates ($\boldsymbol{w}$) and the commanded deflection angles ($\boldsymbol{u}_{com}$) for the flights on July 12^{th} and 13^{th}. As it can be seen from the variations of the angular rates between the two days, the difference on the external disturbance is clearly visible, which comes from the atmospheric wind.

On the 12^{th} of July, there was nearly no wind on the ground level, and the estimated wind was below $2\,m/s$ up in the air at $90\,m$ Above the Ground Level (AGL), whereas the estimated maximum wind speed on the 13^{th} of July was close to $10\,m/s$ with an average of $8\,m/s$. The effect of the atmospheric wind is also clearly visible on the navigation path following which

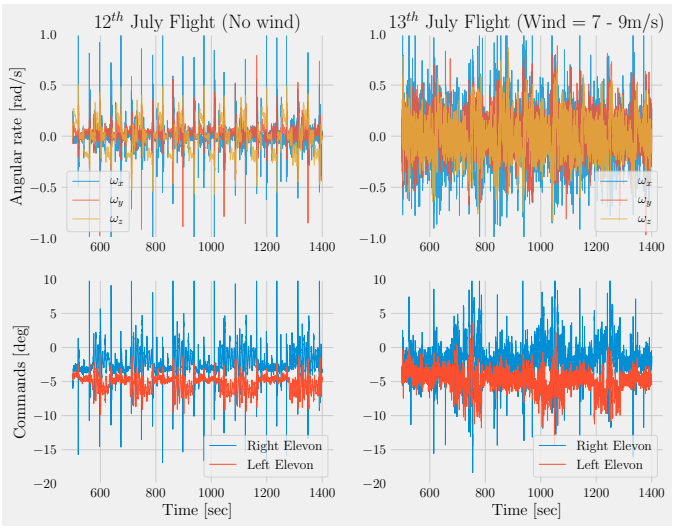

Fig. 4. The effect of day-to-day variation of the atmospheric disturbance (wind) on the sensory data is shown. On the top, the variation of the airframe body angular rates can be seen, and on the bottom, the required autopilot control outputs are visible to cope with wind effects.

is shown in Figure 5, the variance between the desired and actual path increases with the effect of the atmospheric wind.

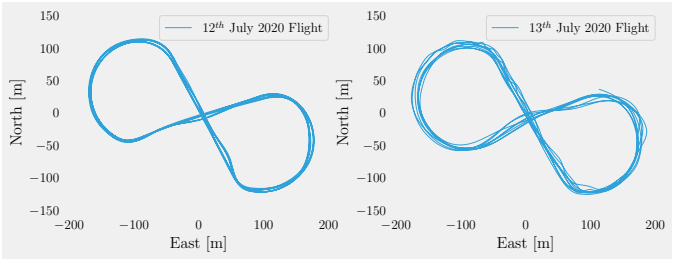

Fig. 5. Showing the increased variance between the desired and actual path as a result of the atmospheric wind on two consecutive flight days.

If only we could have changed the order of the days...

If we assume that we could have gathered the data from 13^{th} of July first, and train and optimize the model with that data, again we would have obtained very good and similar prediction performance when we compare only with that day's test data. This is shown in the Table IV, with the calculated $MCC = 0.94$.

TABLE IV
13^{th} JULY FLIGHT BINARY CLASSIFICATION REPORT

	Precision	Recall	F1-score	Support
Nominal	0.97	0.98	0.98	1169
Faulty ($d_1 = 0.3$)	0.97	0.95	0.96	629
Accuracy			0.97	1798
Macro avg	0.97	0.97	0.97	1798
Weighted avg	0.97	0.97	0.97	1798

However, different than the first case, if we use this model (i.e., trained on 13^{th} of July data), and do the real-time inference prediction with the data gathered on 12^{th} of July (i.e., *fly virtually*), we would have obtained a better prediction performance in the flight, meaning a better generalization. This is shown in Table V with the calculated $MCC = 0.75$

TABLE V
12^{th} JULY FLIGHT (WITH 13^{th}'S MODEL) BINARY CLASSIFICATION REPORT

	Precision	Recall	F1-score	Support
Nominal	0.82	0.93	0.87	211
Faulty ($d_1 = 0.3$)	0.93	0.82	0.87	239
Accuracy			0.87	450
Macro avg	0.87	0.87	0.87	450
Weighted avg	0.88	0.87	0.87	450

Like before, the real-time prediction performance is shown in Figure 6, where the *Applied Faults* are only shown for the ground-truth, and on the bottom, instantaneous predictions are shown in blue. One can see the improvement of the real-time prediction performance.

With this model, the *State Filter* also works nicely and eliminates most of the false negatives (i.e., false prediction of Nominal phase while being in fault phase), however it increases the time of the faulty phase prediction significantly as it requires three consecutive equal predictions. This is very visible on the fault number 4, just after 1000^{th} second.

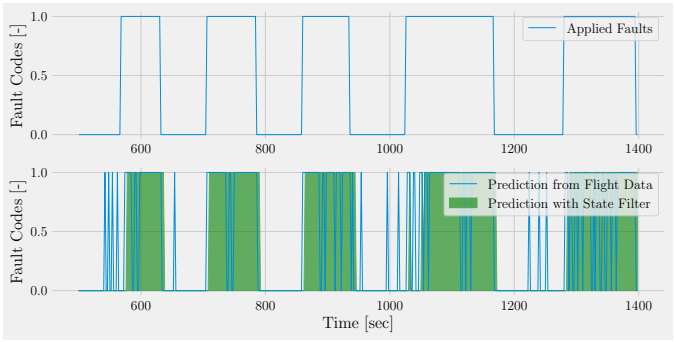

Fig. 6. In-flight prediction performance of the model that is trained on 13^{th} of July's data during 12^{th} of July flight (virtually, by using the flight data).

B. Imperfection in Real-life & Wrong Trim Effect

The test aircraft is designed according to a general requirement list and manufactured in-house. Several imperfections exists because of manufacturing method, integration of the equipment or even geometric wrapping and twisting of the vehicle during transportation to the test field. Figure 8 shows the deformation on the left wing of the test aircraft *Zagi*, which is twisted upward. This deformation increases the local angle of attack on the left wing which generates positive rolling moment (i.e., right wing going down). Therefore, the aircraft has a tendency to turn right, and again therefore in order to go straight and stabilize the rolling moment, the left control surface has to have a built-in negative deflection as shown also in Figure 8. This required negative deflection comes from the

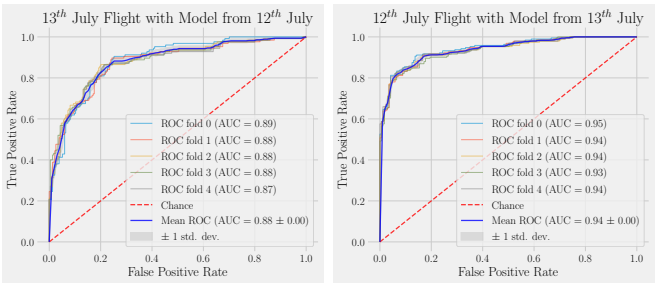

Fig. 7. Receiver Operating Characteristics curves of the prediction performance of two models that are subject to other day's flight. On the right, It can be seen that the model trained with the data coming from windy conditions (13^{th} of July) generalize better to the calm day's flight (12^{th} of July) as the area under the curve is bigger, and from the steepness of the curve.

integrator of the PID controller automatically, and the user does not realize the problem in nominal conditions. However, once there is an injected fault which reduces the efficiency of the control surfaces, the autopilot controller has to augment its commanded values (i.e., desired commands or set-points) proportionally in order to obtain the same applied control deflection.

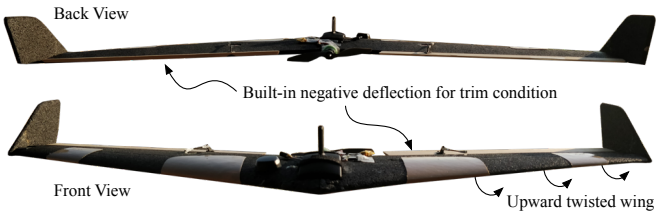

Fig. 8. Geometric twist of the left wing is shown. There is an increase in the local angle of attack on the left wing, which results a positive roll tendency, which has to be compensated by negative (upward) control surface deflection on the left wing. This compensation is handled automatically by the integrator of the PID controller.

The explained effect on the augmented commands can be observed from the flight logs. For example, Figure 9 shows the flight of 17^{th} of July, where the upper plot shows the injected faults (0: nominal, 1: right elevon $d_1 = 0.3$, 2: left elevon $d_2 = 0.3$), and the bottom plot shows the commanded controls ($\boldsymbol{u}_{com}$).

When the injected fault code is 2, meaning that left elevon control efficiency is reduced to 30% ($d_2 = 0.3$) of its nominal capacity, the commanded control values augments significantly higher compared to $d_1 = 0.3$ case. The augmented values increases the embedded information in the signal, therefore detection of the d_2 is easier compared to detection of d_1 faults.

C. Multi-Class Classification for Varying Control Effectiveness

Two consecutive and identical flights have been made on 21^{st} and 23^{rd} of July in order to concentrate on multi-class classification with more data to train on. Figure 11 shows the some sensory information gathered from 21^{st} of July's flight, just as an example to the interested reader (which can also

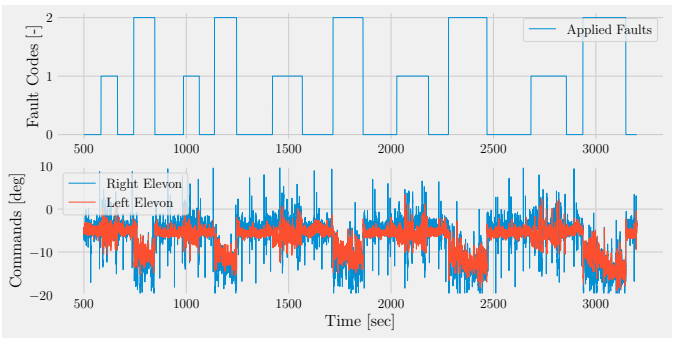

Fig. 9. Bottom plot shows the output of the autopilot's control commands $\boldsymbol{u}_{com}$, not the actual control surface movements, for right and left control surfaces. Once the effectiveness of the left control surface is reduced ($d_2 = 0.3$, Fault code : 2), controller's compensation deflection for the unwanted wing twist also reduces, therefore in order to further compensate both twist effect and the navigation requirements, the control commands increases more than the usual case.

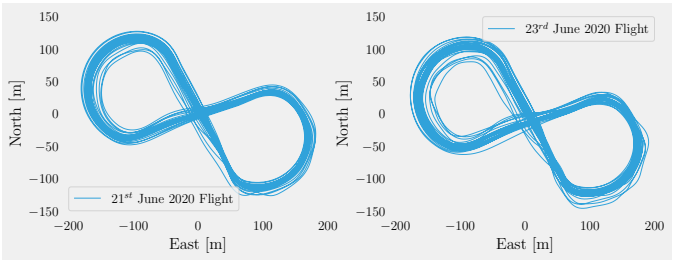

Fig. 10. Trajectories flown during the 21^{st}(Wind speed = $2.5\,m/s$) and 23^{rd}(Wind speed = $5.0\,m/s$) of July 2020.

be obtained numerically from the shared data-set). As before, the right control surface efficiency is reduced ($d_1 = 0.3$), and additional cases have also been investigated, by reducing the left control surface's efficiency in between $d_2 = 0.3 - 1.0$. The fault codes and corresponding faults are listed in the Table VI. We put more emphasis on the detection of left control surface faults (d_2), using the geometric twist caused effect as an advantage.

TABLE VI
FAULT CODES AND CORRESPONDING FAULTS FOR MULTI-CLASS CLASSIFICATION

Fault code	Fault	Fault code	Fault
0	Nominal	5	$d_2 = 0.6$
1	$d_1 = 0.3$	6	$d_2 = 0.5$
2	$d_2 = 0.9$	7	$d_2 = 0.4$
3	$d_2 = 0.8$	8	$d_2 = 0.3$
4	$d_2 = 0.7$		

The first flight was made on 21^{st} of July 2020. After a short preparation time on the ground, the aircraft is launched and immediately started its fully autonomous flight with a figure-eight flight pattern, which can be seen on the left of the Figure 10.

Three consecutive $d_1 = 0.3$ faults are injected with nominal phase between each injection. Each phase has started approximately from the same coordinates, and duration has

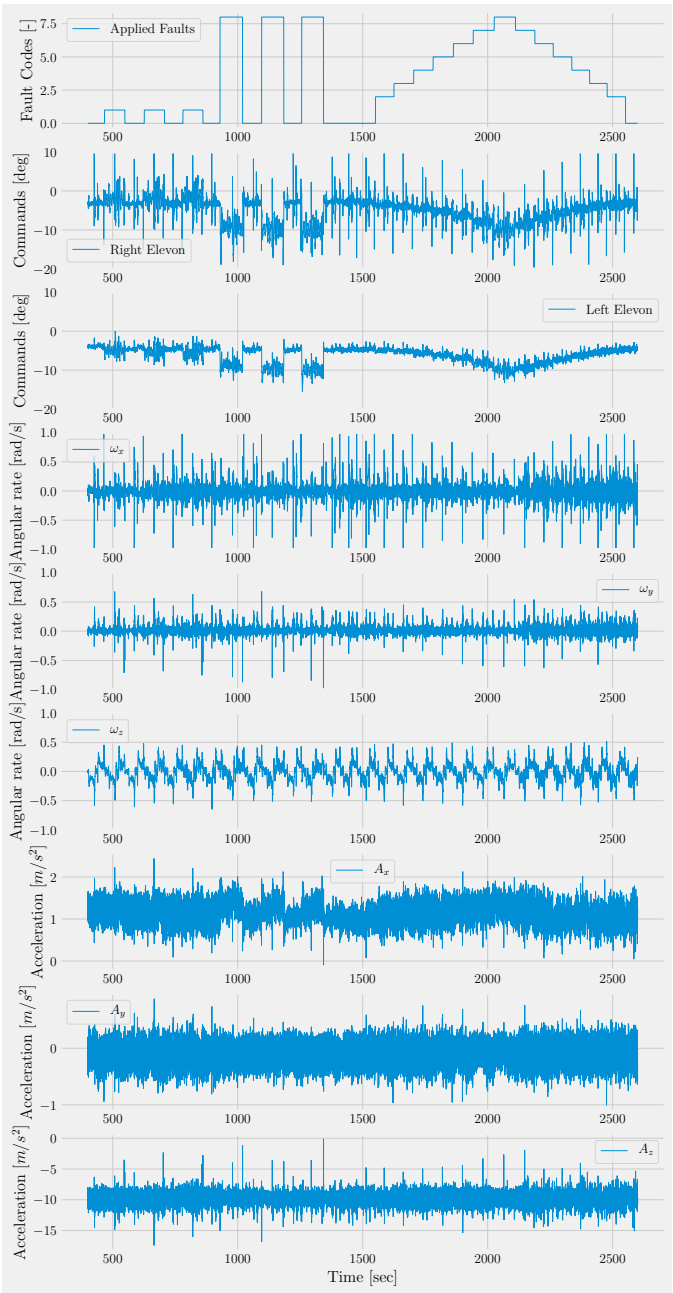

Fig. 11. Angular rates, linear acceleration, and the desired commands for right and left elevons are shown alongside the applied faults during the real flight on 21^{st} of July 2020.

been kept for a full lap of figure-eight pattern. Then another three consecutive $d_2 = 0.3$ faults are injected with nominal phase between each injection. Following that, the left control surface's efficiency has been reduced between $d_2 = 0.3 - 1.0$. The variation of the injected faults are visible in the upper plot of Figure 12.

For training the multi-class classifier model, flight portion is selected between 400^{th} and 2600^{th} seconds. As we have done in the binary classification case, in Section V-A, feature trajectories are generated, again as explained in Section IV-C.

TABLE VII
21^{st} JULY FLIGHT MULTI-CLASS CLASSIFICATION REPORT

	Precision	Recall	F1-score	Support
Nominal	0.94	0.95	0.95	346
$d_1 = 0.3$	0.88	0.98	0.93	121
$d_2 = 0.9$	0.97	0.84	0.90	74
$d_2 = 0.8$	0.99	0.92	0.95	74
$d_2 = 0.7$	0.89	0.95	0.92	75
$d_2 = 0.6$	0.90	0.90	0.90	78
$d_2 = 0.5$	0.96	0.91	0.93	77
$d_2 = 0.4$	0.92	0.96	0.94	81
$d_2 = 0.3$	0.99	0.95	0.97	175
Accuracy			0.94	1101
Macro avg	0.94	0.93	0.93	1101
Weighted avg	0.94	0.94	0.94	1101

Data has been split into two part being 80% *training data*, and 20% *test data*. Each fault category is labeled according to its fault code in order to generate the multi-labeled data. An SVM algorithm that uses Radial Basis Function kernel is trained and optimized with grid search method using the *training data*. The best parameters are found to be $C = 10$ and $\gamma = 1/160$ ($\gamma =$'auto' takes $1/nb_{features}$ as default) for this flight portion. Table VII shows the classification performance report of the optimized model that is evaluated on the separate *test data*, and additionally, Matthews Correlation Coefficient is found to be $MCC = 0.93$.

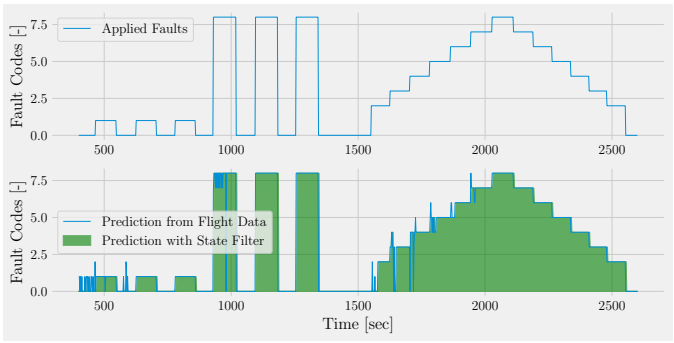

Fig. 12. Prediction performance of the model on the whole flight data, after training on 80% of the same flight data.

If we look at the prediction performance of this model on the full flight data, shown in Figure 12, it can be seen that we can detect almost all of the fault classes. There exists some false instantaneous predictions, however those are mainly eliminated by the *State Filter* shown in shaded green. The real issue is to see the generalization capability of the model on a new dataset (i.e., another flight day).

An almost identical flight has been repeated on the 23^{rd} of July 2020, with the same injected fault classes, on the same figure-eight pattern. However as mentioned before on the Section V-A, day to day conditions affect the environment which has a big effect on the prediction performance of the trained model. The model that is trained with the 21^{st} of July's data, has been uploaded and used as inference model

during the real-time prediction of the fault classes on the 23^{rd} of July flight day. Figure 13 shows the real-time prediction performance.

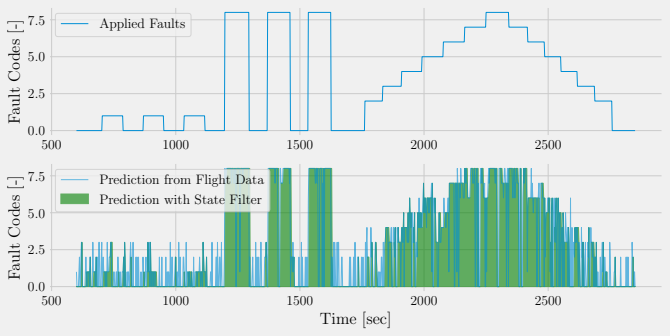

Fig. 13. In-flight prediction performance of the model that is trained on 21^{st} of July's data during 23^{rd} of July flight.

After gathering the flight data, classification report is calculated, and shown in Table VIII, and Matthews Correlation Coefficient is found to be $MCC = 0.37$.

TABLE VIII
23^{rd} JULY FLIGHT PREDICTED BY 21^{st}'S MODEL

	Precision	Recall	F1-score	Support
Nominal	0.57	0.65	0.60	362
$d_1 = 0.3$	0.50	0.42	0.45	125
$d_2 = 0.9$	0.12	0.18	0.15	71
$d_2 = 0.8$	0.24	0.32	0.27	73
$d_2 = 0.7$	0.45	0.31	0.37	74
$d_2 = 0.6$	0.51	0.28	0.36	76
$d_2 = 0.5$	0.39	0.34	0.36	77
$d_2 = 0.4$	0.29	0.30	0.29	81
$d_2 = 0.3$	0.79	0.70	0.74	187
Accuracy			0.48	1126
Macro avg	0.43	0.39	0.40	1126
Weighted avg	0.50	0.48	0.49	1126

As it can be seen from the results, the model that is trained with only limited data from 21^{st} of July's data can still detect in-flight faults on 23^{rd} of July, but with a very low confidence, which results a lot of false predictions. As mentioned before, this is again a problem of over-fitting to a limited dataset, and not being able to generalize to the new day's data. As a solution to improve the model, we can concatenate the flight data of 21^{st} and 23^{rd} of July and train and optimize the model using this bigger dataset. Trained on 80% of the data, and tested on rest 20%, we obtained a better model that can generalize for the two different conditions coming from separate days. The *virtual* real-time prediction on the concatenated data is shown in the Figure 14 and the classification report is presented in the Table IX.

The new model can detect different fault classes with a high confidence, especially after the use of *State Filter* it can neglect most of the false predictions. It is always necessary to gather more data in different type of weather and atmospheric conditions as well as different type of flight patterns and

TABLE IX
$21 - 23^{rd}$ JULY FLIGHT PREDICTED BY $21 - 23^{rd}$'S MODEL

	Precision	Recall	F1-score	Support
Nominal	0.96	0.99	0.98	878
$d_1 = 0.3$	0.99	0.95	0.97	248
$d_2 = 0.9$	0.97	0.82	0.89	146
$d_2 = 0.8$	0.91	0.95	0.93	147
$d_2 = 0.7$	0.95	0.95	0.95	150
$d_2 = 0.6$	0.97	0.93	0.95	152
$d_2 = 0.5$	0.96	0.97	0.96	155
$d_2 = 0.4$	0.96	0.93	0.94	162
$d_2 = 0.3$	0.98	0.99	0.98	364
Accuracy			0.96	2402
Macro avg	0.96	0.94	0.95	2402
Weighted avg	0.96	0.96	0.96	2402

mission definitions. However more important is to find the way to generalize the problem so that it can work in all of those given conditions for different type of vehicles and imperfections, etc... Capturing the common behaviors and finding a generic model is out of the scope of this paper, however within the interest of the authors so it will be left as a future work.

D. Additional Challenges of the In-Flight Detection

Some of the problems that were not foreseen before the flight experiments that we thought worth sharing were :

- **Sensory data transfer rate** : The communication between the autopilot and the companion board, which uses serial connection, passes the sensory information (IMU, commands) to be used in the fault detection. The parsing of the binary messages were taking too long and preventing to go over $10\,Hz$, which was not realized while we were trying higher frequency of sensory data collection for the feature trajectories. Which results phased and not aligned feature trajectories, that reduces the prediction performance of severely. We have corrected the parsing function to overcome this issue, and tested upto $100\,Hz$ with good results (only communication, no predictions).
- **Telemetry Communication Loss** : The communication between the airframe and the ground control station is done by the X-Bee radio modem. Which is proven to be reliable within our mission limits normally. However adding a companion board on-board the vehicle, without careful inspection of the electro-magnetic interactions reduced the telemetry range. Problem solved by distancing the two modules.

VI. CONCLUSION

In this study, we have highlighted the main challenges of real-time fault detection on small scale fixed-wing UAVs. A total of eleven flight logs have been recorded and shared publicly for future potential use by other researchers that are interested in fault and anomaly detection. Our proposed method uses a data driven algorithm, SVM, in order to classify

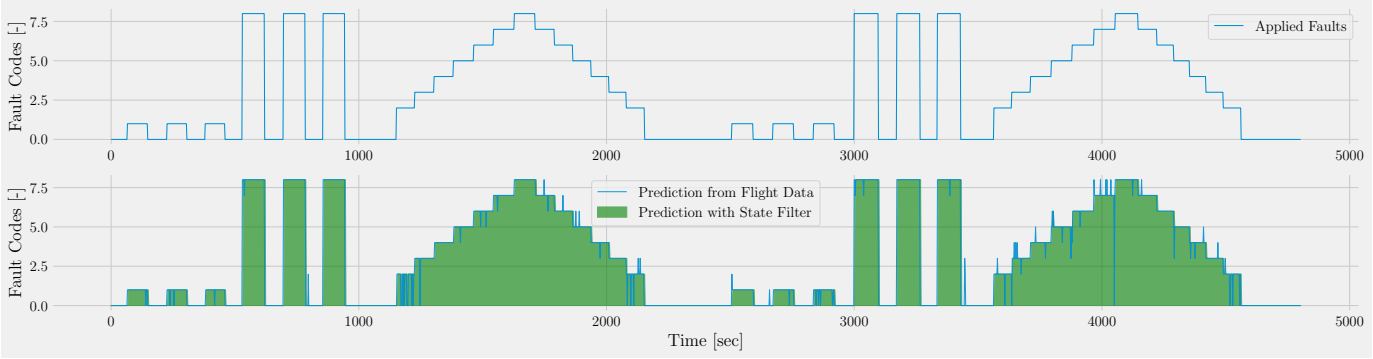

Fig. 14. After concatenating the flight data from 21^{st} and 23^{rd} of July, the model is trained and optimized using the 80% of the concatenated data, virtually flown over the whole data in order to obtain the in-flight prediction performance as shown on the bottom of the plot.

the behavior of the vehicle in nominal flight phase and faulty phase. Feasibility of a basic binary classification is shown, despite the well-known over-fitting problem caused by limited data. We have shown that geometrical imperfections that are common in small UAVs can cause particular effects on the prediction performance, sometimes being positive as our given example. We have experienced that the computational limitations of the inference hardware should be carefully taken into account during the training phase. Especially, the data transfer rate and sampling sequence synchronization has to be planned beforehand. Nevertheless, the SVM algorithm is robust enough to cope with small deviations from expected sampling time, and was still capable of demonstrating the detection of different fault classes during real-flight experiments.

Acknowledgment

The authors would like to thank Gautier Hattenberger and Xavier Paris who shared their expertise in *Paparazzi Autopilot System*.

References

[1] R. A. Eslinger and P. R. Chandler, "Self-repairing flight control system program overview," in *Proceedings of the IEEE 1988 National Aerospace and Electronics Conference*, 1988, pp. 504–511 vol.2.

[2] M. Bodson and J. E. Groszkiewicz, "Multivariable adaptive algorithms for reconfigurable flight control," *IEEE transactions on control systems technology*, vol. 5, no. 2, pp. 217–229, 1997.

[3] Y. Zhang and J. Jiang, "Active fault-tolerant control system against partial actuator failures," *IEE proceedings-Control Theory and applications*, vol. 149, no. 1, pp. 95–104, 2002.

[4] M. D. Tandale and J. Valasek, "Fault-tolerant structured adaptive model inversion control," *Journal of Guidance, Control, and Dynamics*, vol. 29, no. 3, pp. 635–642, 2006.

[5] R. Venkataraman and P. J. Seiler, "Safe flight using one aerodynamic control surface," in *AIAA Guidance, Navigation, and Control Conference*, 2016, p. 0634.

[6] R. Venkataraman and P. Seiler, "Fault-tolerant flight control using one aerodynamic control surface," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, pp. 570–584, 2019.

[7] P. Bauer, R. Venkataraman, B. Vanek, P. J. Seiler, and J. Bokor, "Fault detection and basic in-flight reconfiguration of a small uav equipped with elevons," *IFAC-PapersOnLine*, vol. 51, no. 24, pp. 600–607, 2018.

[8] T. E. Menke and P. S. Maybeck, "Sensor/actuator failure detection in the vista f-16 by multiple model adaptive estimation," *IEEE Transactions on aerospace and electronic systems*, vol. 31, no. 4, pp. 1218–1229, 1995.

[9] C. Hajiyev and F. Caliskan, "Sensor and control surface/actuator failure detection and isolation applied to f-16 flight dynamic," *Aircraft Engineering and aerospace technology*, vol. 77, no. 2, pp. 152–160, 2005.

[10] J. W. Melody, T. Hillbrand, T. Başar, and W. R. Perkins, "H_∞ parameter identification for inflight detection of aircraft icing: The time-varying case," *Control Engineering Practice*, vol. 9, no. 12, pp. 1327–1335, 2001.

[11] G. J. Ducard, *Fault-tolerant flight control and guidance systems: Practical methods for small unmanned aerial vehicles*. Springer Science & Business Media, 2009.

[12] D. Magill, "Optimal adaptive estimation of sampled stochastic processes," *IEEE Transactions on Automatic Control*, vol. 10, no. 4, pp. 434–439, 1965.

[13] J. Stutz, "On data-centric diagnosis of aircraft systems," *IEEE Transactions on Systems, Man and Cybernetics*, 2010.

[14] E. Baskaya, "Fault detection and diagnosis for drones using machine learning," Ph.D. dissertation, Université Paul Sabatier-Toulouse III, 2019.

[15] B. Eroglu, M. C. Sahin, and N. K. Ure, "Autolanding control system design with deep learning based fault estimation," *Aerospace Science and Technology*, p. 105855, 2020.

[16] V. Sadhu, S. Zonouz, and D. Pompili, "On-board deep-learning-based unmanned aerial vehicle fault cause detection and identification," *arXiv preprint arXiv:2005.00336*, 2020.

[17] V. Sindhwani, H. Sidahmed, K. Choromanski, and B. Jones, "Unsupervised anomaly detection for self-flying delivery drones," 2020.

[18] P. Freeman, R. Pandita, N. Srivastava, and G. J. Balas, "Model-based and data-driven fault detection performance for a small uav," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 4, pp. 1300–1309, 2013.

[19] A. Keipour, M. Mousaei, and S. Scherer, "Alfa: A dataset for uav fault and anomaly detection," *arXiv preprint arXiv:1907.06268*, 2019.

[20] ——, "Automatic real-time anomaly detection for autonomous aerial vehicles," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5679–5685.

[21] G. Hattenberger, M. Bronz, and M. Gorraz, "Using the paparazzi uav system for scientific research," 2014.

[22] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 2002.

[23] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using matthews correlation coefficient metric," *PloS one*, vol. 12, no. 6, p. e0177678, 2017.