Noname manuscript No.
(will be inserted by the editor)

Fault diagnosis for UAVs using flight data via machine learning

Elgiz Baskaya · Murat Bronz · Daniel Delahaye

Received: date / Accepted: date

Abstract The new era of small UAVs populating the airspace introduces many safety concerns due to the lack of a pilot onboard and less accurate nature of the sensors. This necessitates intelligent approaches to address the emergency situations that will be inevitable for all classes of UAV operations defined by EASA (European Aviation Safety Agency). The hardware limitations for these small vehicles point the utilization of *analytical redundancy* rather than the usual practice of hardware redundancy in manned aviation. In the course of this study, machine learning practices are implemented to diagnose faults on a small fixed-wing UAV to avoid the burden of accurate modeling needed in model-based fault diagnosis. A supervised classification method, SVM (Support Vector Machines), is used to classify the faults. The attributes used to diagnose the faults are gyro and accelerometer measurements. The idea to restrict the data set to accelerometer and gyro measurements is to check the method's abilities of classification even with a small and inexpensive chip set without the need to access the data from the autopilot such as the control input information. This work only addresses the faults in the control surfaces of a UAV. More specifically, the faults considered are the control surface stuck at an angle and loss of effectiveness.

Real flights have been arranged to generate faulty flight data by manipulating the open source autopilot *Paparazzi*. All data and the code are available in code sharing and versioning system *Github*. The training is held offline due to the need for labeled data and computational burden of the tuning phase of the classifiers. The results show that over the flight data, SVM yields an F1 score of 0.98 for classification of control surface stuck fault. For the loss of efficiency fault, some feature engineering, involving the addition of past measurements is needed to attain the same classification performance.

Keywords SVM · Fault detection and diagnosis · Machine learning · Feature engineering · *Paparazzi* · Real flight data

E. Baskaya
ENGIE Ineo - Groupe ADP - SAFRAN RPAS Chair
ENAC, 7, avenue Edouard Belin, 31400, Toulouse, France
Tel.: +33-616460022
E-mail: benelgiz@gmail.com

M. Bronz
UAS Lab, ENAC Lab
ENAC, 7, avenue Edouard Belin, 31400, Toulouse, France
E-mail: murat.bronz@enac.fr

D. Delahaye
ENAC Lab
ENAC, 7, avenue Edouard Belin, 31400, Toulouse, France
E-mail: deniel.delahaye@enac.fr

1 Introduction

Lately, the popularity and reachability of UAS have risen steeply thanks to the advancements in electronic components and yet the decrease in their cost. This called the regulators to duty due to the concerns from the manned aviation community after incidences of drones flying close to civil aircrafts and airports as well as the public witnessing a number of accidents. The U-Space concept in Europe (UTM in the US, which aims at enabling safe integration into airspace), gives the insight, by showing in their roadmaps, that level of drone automatization and level of drone connectivity (drone to drone and drone to infrastructure) will guide the pace of the U-Space services (U1 to U4) [27]. These enablers will allow intelligent agents to share information and automate complex situations such as emergencies.

Thus, future of aviation will inevitably go towards automatization.

In this study, we propose a concept that will contribute to the automatization of drones, that will make them more intelligent and contribute to a safer integration of drones into airspace. These awareness abilities allow to mitigate risks in accordance with the risk assessment procedures offered by JARUS (SORA) [16] defined by [8]. The abilities of a drone after an emergency should be assessed and depending on the availability of the environment, a recovery procedure might be initiated. For the cases that a recovery is not possible, safe ditching might be needed to reduce the harm in the air or on the ground.

This work introduces an end-to-end design to diagnose faults on drones. The discussion starts by fault tolerant control systems in general and is followed by two different methods for fault detection and diagnosis, model-based and data-driven approaches. Then, as a data-driven approach, SVM is introduced. SVM, being a supervised classification method, requires labeled data, both faulty and nominal. This can be achieved via simulations or real flights. Since a number of works discuss the fault detection & diagnosis (FDD) based on simulated measurements and a few utilizes real flight data, we have selected to issue real flights to include many effects which is usually ignored, such as the controller's effects on the diagnosis. Open-source autopilot, *Paparazzi*, is utilized to inject faults during flights and is the next to be explained briefly. After that, modifications to autopilot control system and modes is discussed. Labeling data is done offline after the flight and is explained under the section post-processing of data after the flight. Since labeled data and the method is presented to the reader at that point, we proceed to apply the SVM classification to labeled flight data. In the following section, three different phases (training, tuning, evaluation) of SVM application to classification are explained with a focus on drone actuator fault detection. Finally, tuned classifiers are evaluated for two types of faults (loss of effectiveness and control surface stuck) and the results are given in tables.

2 Fault Detection and Diagnosis

All systems are susceptible to faults of different nature. Fault is defined by Isermann in his book [14] as "A fault is an unpermitted deviation of at least one characteristic property (feature) of the system from the acceptable, usual, standard condition". Existing irregularities in sensors, actuators, or controller could be amplified due to the control system design and lead to failures which is by definition "a permanent interruption of a system's ability to perform a required function under specified operating conditions". Until now, the usual procedure to mitigate the emergency situations in aviation is the pilot taking the control and following the predefined steps for recovery. But due to the systems gaining more autonomy, the abilities of pilots are degrading in terms of dealing this irregular and not very common situations. A joint work by NASA Langley Research Center and Federal Aviation Administration (FAA) [7] discusses the abilities of pilots to follow predefined recovery procedures and results show that more autonomy will be beneficial to achieve safer systems in case of failures. Another reason for the need of further autonomy is the emergence of unmanned aviation, where there is either no pilot or a remote one, lacking situational awareness.

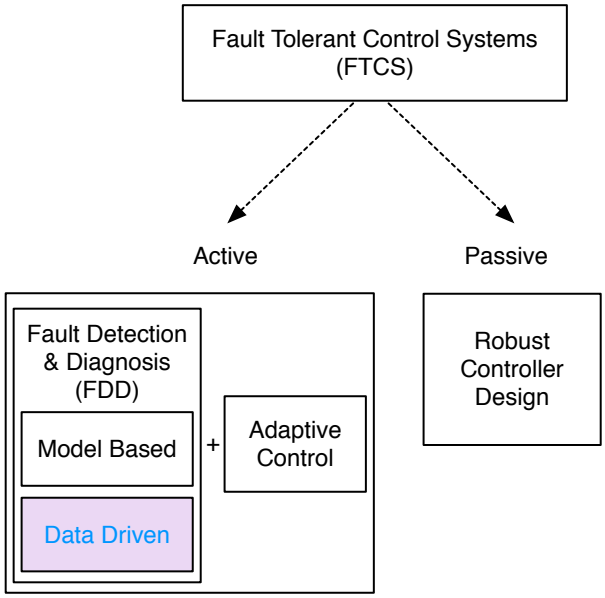

Fig. 1 Variations of fault tolerant control systems

Fault tolerant control systems (FTCS) are commonly categorized under two categories: passive and active FTCS. In passive FTCS, the flight controller is designed in such a way to accommodate the faults. Active FTCS first distinguishes the fault via fault detection and diagnosis module and then switch between the designed controllers specific to the fault case or design a new one online [2]. While active FTCS requires more tools to handle faults as seen in Fig. 1, for faults not predicted and not counted for during the design of the robust controller, it is necessary to utilize an active method. One of those tools, Fault detection and diagnosis (FDD), is handled in two main steps: fault detection and fault di-

agnosis. Fault diagnosis encapsulates fault isolation and fault identification.

Even with a long list of available methods, aerospace industry has not implemented FTCS widely, except some space systems, due to the evolving nature of the methods, the tricky nonlinear nature of the problem, design complexity and high possibility of wrong alarms in case of large disturbances and/or modeling uncertainties. For now, the most common reliability measure is the hardware redundancy because of its ease and maturity of being implemented on various critical missions involving human lives. The size and cost reduction of the systems call for a change such as implementing analytical redundancy rather than the hardware redundancy especially for reduced risk operations.

Two distinct options to proceed in analytical redundancy are the model based approaches and data-driven approaches. Model based fault diagnosis highlights the components of a system and the connections in-between, and their corresponding fault modes. Data driven fault diagnosis rely on the observational data and prefers dense, redundant and with a frequency larger than the failure rate.

2.1 Model based approaches

Model based approaches are based on detecting the fault by exploiting the relations between estimated states and measurements. Estimation of the states, estimation of the model parameters, or parity-space are the most common ways to implement a model based approach. The output of the model based fault detection methods is the stochastic behavior with mean values and variances. Change detection methods are utilized to detect the deviations from the normal behavior. Along those detection methods, the most commonly used ones are mean and variance estimation, likelihood-ratio-test and Bayes decision, run-sum test and two-probe t-test. Detection is mainly supported by simple threshold logic or hypothesis testing in most of those applications [15]. Detecting sensor and actuator faults via state estimation, utilizing an EKF is applied to a F-16 model in [13] and parameter identification via H_∞ filter is used to indicate icing in [19] are examples of model based fault detection based on simulations. A drawback of model-based approaches is the accurate model requirement of the aircraft to achieve successful detection. In a small UAV system susceptible to various uncertainties/disturbances, selection of a model-based approach might not yield to satisfactory detection performance. A way to handle that is to offer solutions to cope with the uncertainties. A fairly old study in 1984, investigates the design problem FDI systems robust to uncertainties within the models. One of the two steps of FDI, two steps being the residual generation and decision-making, is targeted. They offer to handle model uncertainties, by designing a robust residual generation process [5]. Another study deals with model uncertainties by determining the threshold of the residual in a novel way with an application to detect aileron actuator fault [22]. [24] utilize two cascade sliding mode observers state estimation and fault detection to guarantee staying in sliding manifold in the presence of unknown disturbances and faults. Another drawback is that those relatively small and cheap systems rarely have an accurate model, or even if they do, their mathematical model is generally constructed within the flight envelope, and does not necessarily describe the possible dynamics invoked by a failure on board.

2.2 Data-Driven methods

The usual autonomy practices of manned aviation have implemented model-based methods successfully for a long time now. With the introduction of UAVs, model-based methods are not always the best option considering the cost of accurate modeling, the varieties of the manufactured drones, and their sensitivity to disturbances. Also lacking a pilot onboard for the emergency situations, autopilots would be needed to cope with the failure cases, where the models are not usually fitted for and more difficult to acquire. All these challenges call for alternative approaches for the quite challenging problem of FDD. The increased efficiency of sensors on board, the increase in the computational capabilities of autopilot processors, and the advances in machine learning techniques in the last decade may offer efficient data-driven solutions to FDD.

In data driven methods, models for dynamics of the system is not necessarily utilized. The data available is the source of information with regard to the behavior of the system. **********Supervised learning, which requires labelling the fault cases previously in the training data, is usually utilized for data-centric inference of causes. **********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. What is needed at that point is to first detect and localize the fault and then to consult domain experts for labeling for further integration of this fault into the diagnosis scheme [25].

For now, application of machine learning algorithms in the safety systems are bounded by the available certification practices. But recently, for drones in the certified category, there exists quite an effort to search for alternative certification methods that will allow machine

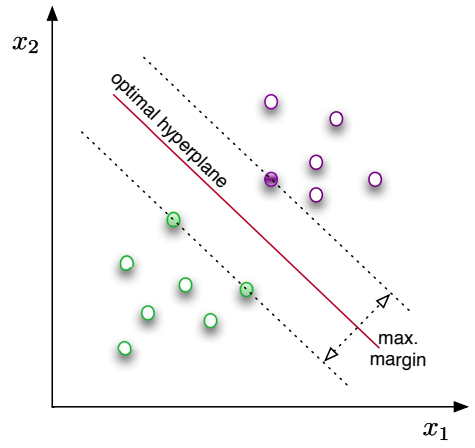

Fig. 2 SVM working principle

learning methods to be used as parts of safety systems onboard a drone [1,9]. Advances in this domain shows that the automation systems might change in that direction.

Amidst data driven methods for FDD, such as Neural Networks [23] and Principal Component Analysis (PCA) [26], Support Vector Machines (SVM) appear more recently in the literature. [10] argues artificial intelligence methods for fault detection of complex systems. Comparison between PCA and model based stochastic parity space approaches is given in [12]. In [18], the authors argues to use dynamic PCA since UAV flight controls is a dynamic system itself and DPCA can reflect unknown disturbances, while model-based approaches can only model typical disturbance. SVM is introduced in 1964 in the statistical learning theory domain and relies on structural risk minimization principle [29]. Although the theory has old roots, its application to classification as a machine learning algorithm is recent and originally offer solutions for two-class classification [3,28]. SVM's first application as a classifier was mainly on object classification in images and followed by fault detection lately. The use of SVM on fault detection has gained popularity thanks to its improvement in accuracy of detection [17]. Application of SVM on fault detection is mostly held in mechanical machinery, such as roller bearings, gear box, turbo pump rotor and sometimes other systems; semi-conductors, refrigeration systems and chemical processes. Its application on complex systems has not been very widely adopted yet and forms the basis of study for our research.

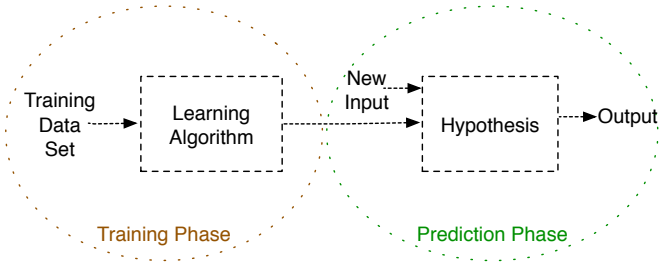

Fig. 3 Supervised learning is achieved in two steps: *Training Phase* in which the model parameters are calculated given labeled data and *Prediction Phase* in which the label is predicted for a new input using the trained model.

3 Support Vector Machines

SVM is a relatively new approach for classification offering better generalization property thanks to its foundations on the structural risk minimization principle while other classifiers (such as logistic regression) usually minimizes the empirical risk [11,30]. Thanks to that property, the capacity of generalization is improved even with a small number of instances by reducing the risk of overfitting for a nicely tuned parameters setting. SVM also offers a vast number of advantageous features such as its compatibility with nonlinear systems. Furthermore, taking advantage of convex optimization problems in the solution of SVM models, SVM avoids local minimas, to which Neural Networks is inherently prone to.

The idea behind SVM is to find an optimal hyperplane that will linearly separate the classes. This is achieved with the introduction of maximum margin concept which is the distance in between the boundaries when they are extended until hitting the first data point as in Fig. 2. The points closest to the hyperplane (decision boundary) are called the support vectors and are the representatives of the data sets to be used for the decision process. This helps to decrease the data to handle abruptly, enhancing the ability to cope with the curse of dimensionality and reducing the computational complexity. For problems where the classes are not linearly separable, kernels are used to map data into higher dimensional feature spaces where they can be separated with a linear hyperplane.

SVM being a supervised classification algorithm has two main phases as shown in Fig. 3. In the training phase, the model is learned to fit the labeled data that is fed to the SVM algorithm. This phase is usually followed with a tuning phase where some of the parameters of SVM are changed and results are compared to have the best fit via cross validation to avoid overfitting. The last phase is the prediction, where for a new

instance, the classifier predicts if it corresponds to a faulty or nominal condition.

Training data set includes labeled data where the label can belong to one of two possible cases. This data set is saved in a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ where m, n correspond to number of instances and features respectively. The label information corresponding to the measurement instances is also fed to the SVM algorithm during the training phase as output vector $\boldsymbol{y} \in \{-1, 1\}$. The aim of SVM is to find an optimal hyperplane maximizing the margin by solving the optimization problem for non-linearly separable datasets

****** DENKLEMI KONTROL ET VE ACIKLA BIRAZ ********

$$min_{\gamma,\omega,b} \quad \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i \tag{1}$$

$$s.t. \quad y^i(\omega^T x^(i) + b) \geq 1 - \xi_i, \ i = 1, \cdots, m \tag{2}$$

$$\xi_i \geq 0, \ i = 1, \cdots, m \tag{3}$$

The data set available is first divided into two portions with a percentage of 20%, 80% where the larger set is the training set and the remaining is the test set. Further, the training set is divided into cross-validation and training sets. The idea to split data is to avoid overfitting. When a model is overfit, it fits very accurately to the data it is trained with, but fails to generalize to new data. This results in a poor prediction performance. To avoid overfitting, which is the main problem of parametric discrimination approaches such as neural networks, parameter C is tuned to result in an optimal fit with the cross validation set. Tuning C also gives the user the ability to change the classifier's sensitivity to outliers, since sometimes finding a hyperplane separating all data perfectly is not the favorable option (might cause overfitting). Then the final performance of the classifier is tested on the test set.

The classifier with the best performance is selected and then used in the prediction phase. For a new data input, the classifier is used to predict which class the new data belongs to. On the contrary to some classification methods (such as logistic regression), the prediction does not give with which probability the new input belongs to its predicted class. But there exist methods which can be used to calculate the posterior probabilities, which is the probability that the new measurements belongs to its predicted class[21].

BURASI PROBLEM DEPENDENT

To classify faulty and nominal measurements, a binary classifier has been used in our study. A variety of faults have been investigated which might mainly grouped as classification of stuck and the loss of efficiency of one or both control surfaces.

4 Fault detection from real flight data

This study investigates SVM as a classification tool to diagnose faults. Hence simulating faulty data is a must-have to feed the SVM classifier as well as the nominal flight data. Since SVM is a supervised learning algorithm, faulty data is necessary for training as well as the nominal data. Real flights have been realized to gather data. An open-source autopilot, *Paparazzi* [4], have been adapted to generate faulty control inputs during the flight. The *Paparazzi* Autopilot System provides a software and hardware solution for low-cost mini and micro unmanned air vehicles. Being one of the first open-source autopilot system in the world, *Paparazzi* deals with all three segments: Ground, Airborne, and the communication link between. Airborne software is written in C, however there is an on-going effort to support C++. *Modules* serves as a flexible way to implement new features into *Paparazzi*. With its 130 modules by several developers and researchers, *Paparazzi* includes functions to apply to various domains such as meteorology, imagery, surveillance, advance navigation, formation flight and collision avoidance and many others.

4.1 Injecting faults realtime from *Paparazzi* ground control station (GCS)

For the faulty flight data gathering, some modifications to the *Paparazzi* autopilot was necessary in two main parts: Injecting the faults real-time from GCS, and editing the controller onboard so that the sent faulty input values configures the servos as manipulated from the GCS.

For the former part, a slider is added to the ground station to set the fault during flight and set it back to normal flight conditions if necessary. Fig. 4 shows the GCS view with the fault settings open. This pane can be found under *Settings* > *FAULT* as highlighted in pink in Fig. 4. The four row configuration represents from top to bottom, the multiplicative error in the right elevon, the multiplicative error in the left elevon, additive error in the right elevon, and finally the additive error in the left elevon. The nominal condition where there is no fault, the configurations is $[right\ left\ right\_offset\ left\_offset] = [1.0\ 1.0\ 0\ 0]$. This configuration lets the user to realize all types of actuator faults, such as control surface inefficiency or stuck. To generate the fault of right elevon stuck at its nominal position, setting the first slider ($right$) to zero is enough. The generate stuck fault at other positions, $right\_offset$ slider should be changed to desired stuck position while keeping the first slider at zero.

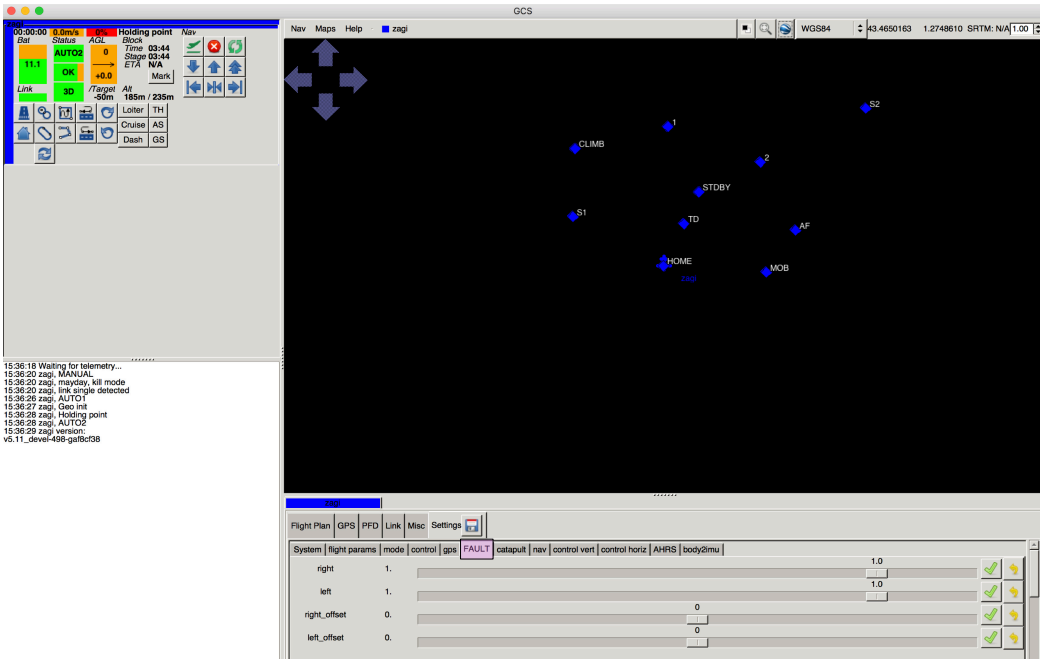

Fig. 4 View of fault injection tool in Paparazzi ground control station view of in during flight

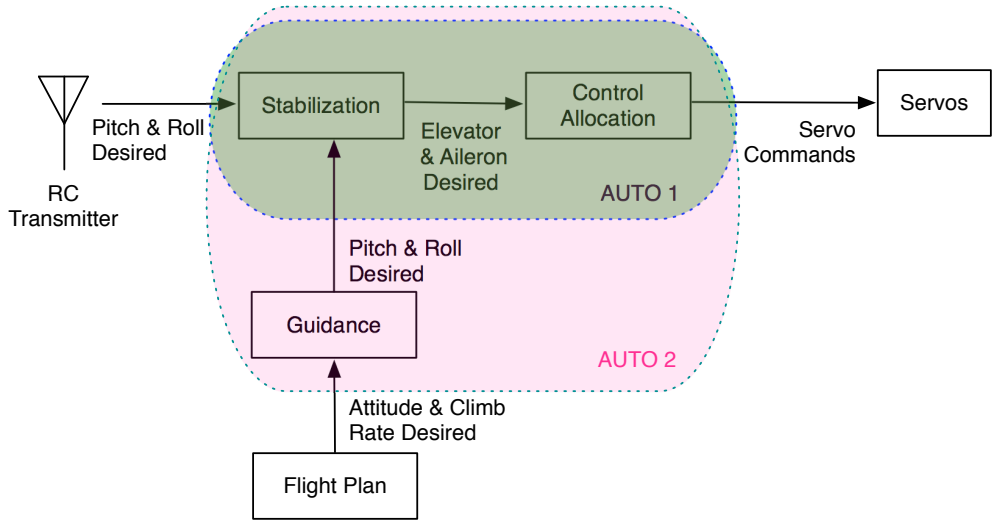

Fig. 5 *Paparazzi* autonomy modes

4.2 Modifications to *Paparazzi* autopilot controls to inject faults during flight

For the second part, which is to modify the servo command from the autopilot to the servos, a look at the *Paparazzi* flight modes is necessary. Most of the times, there are three modes for fixed-wings from control perspective: *Auto 1*, *Auto 2*, *Manual*. In *Auto 1*, the pilot is still in the loop and gives the desired pitch and roll values to the controller and the desired elevator and aileron commands are calculated by the autopilot and passed to control allocation where the final desired servo commands are sent to servos as highlighted *Auto 1* in Fig. 5. In the *Auto 2* mode, there is no need for the pilot since the navigation is also held by the autopilot for a given flight plan. This mode is also given as *Auto 2* in Fig. 5. In *Manual* mode, the pilot gives the desired elevator and aileron commands and desired servo commands are calculated in the autopilot's control allocation phase. So still there is a very low sense of autonomy in the manual phase.

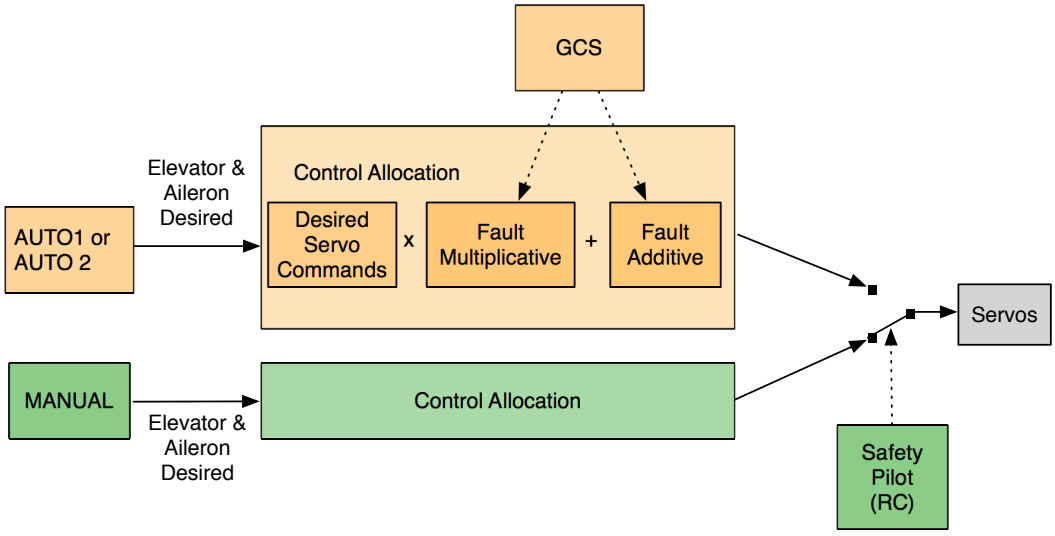

Fig. 6 Modifications on the control modes of *Paparazzi* autopilot

Flying with faults is a challenge. The risk to crash is increased on purpose, so a back up plan is necessary to recover from faulty situations if the drone seems to be out of control and/or about to crash. For that purpose, the faults are only injected to *Auto* modes and *Manual* mode is always free of faults even a fault is given from the ground station. So, when the pilot sees a safety problem during the faulty operation, s/he can switch to *Manual* mode from the remote controller and have the control of the control surfaces free from faults. Depending on the nature of the fault injected, this might be a game changer such as gathering data for control surface stuck fault in which it is sure the drone is going to crash unless an action taken. This is shown in Fig. 6 with a switch initiated by the pilot's remote control. This figure shows that during the control allocation phase, which is the calculation of the desired servo commands from given desired elevator/aileron commands, faults are injected if the mode is *Auto 1* and *Auto 2* when fault multiplicative or fault additive values are changed from the GCS by the operator. When switched to *Manual* mode, the control allocation do not consider the injected faults.

4.3 Post-processing of data after the flight

An example of a part of the flight data [1] is given in Fig. 9 and whole file is available in *Github* [2]. The UAV used to realize the faulty flights in order to generate labeled data is given in Fig. 7. The duration of the flight was around an hour. The flight has been practiced under strong wind. 34 different faults are injected. During the flight, the effect of the faults on the drone was sometimes visible to human eye and sometimes not. For the control surface stuck faults, even for one control surface was stuck case, it immediately gets out of control and safety pilot takes the initiative. Thanks to the piloting skills, no crashes occurred. For the loss of effectiveness of the control surfaces, in which the controller has still an effect but not as efficient as before, error in navigation was observed.

Fig. 7 The flying-wing: *ZAGI*

Now that the .data file is ready, next is to detect the time stamps at which the faults are injected and then label all the data corresponding to this fault interval as designated with different colors in Fig. 9. Since, most of the times, there has been more than one fault type generated during flights, another step in data manipulation is to choose which kind of fault is of interest for further investigations.

The faults or change from fault condition to nominal mode is done via the GCS and there is a corresponding

[1] 17_07_06__10_21_07_SD.data

[2] https://github.com/benelgiz/cureDDrone/tree/master/data/v4_multiplicativeAdditive_MURET_06_07_2017

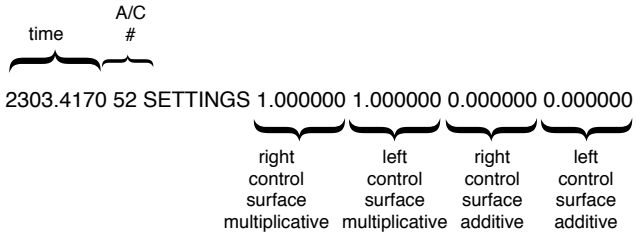

Fig. 8 SETTINGS data saves the multiplicative and additive fault values inserted from the GCS. [1.0 1.0 0.0 0.0] corresponds to nominal case where no fault is injected

message saved to SD card onboard in the messages indicated via *Settings*. *Settings* gives multiplicative fault and additive fault values and only appears in the flight data when one of the control surface effectiveness values is changed. The value of the *Settings* for nominal phase is [1.0 1.0 0.0 0.0] and a corresponding example line in the data during nominal phase entry command can be seen in Fig. 8. It means to multiply the value given by the controller by 1 and add 0, so does not change the values given by the autopilot.

As soon as the value changed in the GCS, a new settings value is saved to the file (Shown with arrows in Fig. 9).

When the actuators are healthy, actual control input signal will be equal to the given input signal. In case of a fault the actual signal can be modeled as

$$\boldsymbol{u}(t) = \boldsymbol{E}\boldsymbol{u}_c + u_f \tag{4}$$

where $\boldsymbol{u}_c$ is the desired control signal, u_f additive actuator fault and $E = diag(e_1, e_2, e_3)$ is the effectiveness of the actuators where $0 \leq e_i \leq 1$ with $(i = 1, 2, 3)$. This model makes it possible to simulate all four types of actuator faults shown in Fig. 10.

To find the indexes where the nominal and faulty data starts and ends, the values of the *Settings* message is investigated. So when there is a *Settings* message in the flight, it should be either a fault generation or going back to nominal condition after a fault. If the message contains the *Settings* message equal to [1.0 1.0 0.0 0.0], this data index is selected as the nominal condition start index and a previous index before next *Settings* message is the last index of this nominal phase. For the fault indexes a similar approach is followed except that *Settings* messages selected are the ones which is different than [1.0 1.0 0.0 0.0]. An example can be seen as Fault #23 in Fig. 9 (the first line of that fault phase data which is a right elevon stuck fault at 0°).

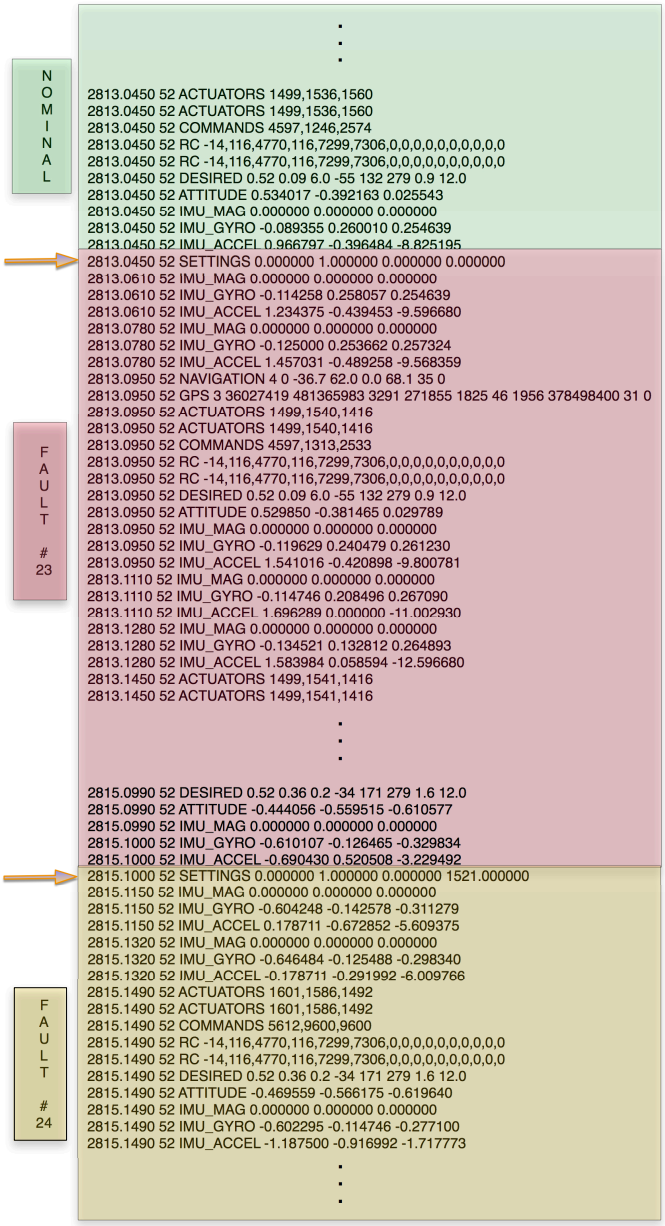

Fig. 9 A small part of the flight data corresponding to nominal and two different fault phases of the flight. *Settings* message exists only if there is a change in the multiplicative and additive fault parameters via GCS

5 Classification of fault via SVM

A binary classifier is used in this work to classify two classes, faulty and nominal. The faults considered in this study are the loss of effectiveness of the control surfaces and the control surface stuck. The last section explains how the faulty data is generated in flight and then labeled on the ground to be fed to the classifier.

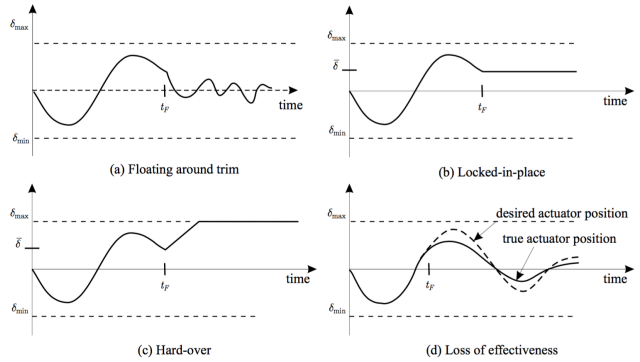

Fig. 10 Common actuator faults [6]

SVM being a supervised classification algorithm has two main phases as shown in Fig. 3: training and prediction. The labeled data set is first divided to two portions with a percentage of 20%, 80% where the larger is the training set and the remaining is the test set. Further, the training set is divided as cross-validation and training sets. The idea to split data is to avoid overfitting.

5.1 Training of the classifier

The first step is to normalize the features of the data in order to make the values of features change with the same order of magnitude. The reason is due to its benefits to the calculation of parameters of the hypothesis via an optimization algorithm and its convergence rate. An important point is to keep that values μ_j, s_j and may be standard deviation if it is used instead of range s_j. During prediction phase, the data first should be scaled with these values attained from learning data. If in the hypothesis artificial feature $x_0 = 1$ is added to the features, do not apply scaling to the artificial feature, but this does not apply to SVM classification.

The aim of SVM is to find an optimal hyperplane maximizing the margin by solving the optimization problem for non-linearly separable datasets. SVM implements the idea of having a confidence in the prediction by using the concept of separating data with large margin. Some other classification methods, such as logistic regression, outputs the probability of a new instance's class, giving the confidence of the prediction as output inherently. On the other hand, SVM does only output if the new instance belongs to a class not necessarily pointing the confidence on this decision. Rather than including this confidence information as an output in terms of probabilities, this information is introduced with the functional and geometric margins. These definitions also serve for mathematically convenience, so that the optimization problem can be represented as a convex optimization problem. Furthermore, introducing the Lagrange duality to obtain the dual form of the optimization problem, use of kernels, to work efficiently in higher dimensional spaces, is eased. The dual form also allows to utilize efficient optimization solvers such as Sequential Minimal Optimization (SMO) [20] which is the solver used in this work as well. Conventionally, training phase of SVM requires to solve a large quadratic programming (QP) problem. Especially for large training set, computational heaviness of this phase might limit the applicability of SVM to specific problems sets. To overcome this constraint, SMO breaks the QP problem into a series of smaller QP problems which can be solved analytically. Default settings for SVM binary classification, included in Matlab Statistics and Machine Learning Toolbox, utilizes SMO for optimization if outlier fractions has not been specified during the function call.

Kernels are at the core of efficient SVM classifiers. A kernel, in general is defined as

$$K(x,z) = \phi(x)^T \phi(z) \tag{5}$$

where ϕ represents the feature mapping. Default settings of Matlab's binary SVM classifier fits a linear model, which results in a linear decision boundary. If the system of interest requires a more complicated decision boundary to effectively classify the training data, mapping the original features might be necessary. Usually the original features of the systems are named as attributes while the mapped set are called the features. In other words, ϕ maps the attributes to the features. Kernels offer various elegant properties such as computational efficiency. Cleverly selected, SVM classifiers can learn in high dimensional spaces represented by ϕ without the need to explicitly find or represent ϕ, but instead calculating $K(x,z)$, which might be computationally more efficient. In this study, Gaussian Kernel, which corresponds to an infinite dimensional feature mapping, is utilized to map the attributes.

$$K(x,z) = exp\left(- \frac{\|x-z\|^2}{2\sigma^2} \right) \tag{6}$$

5.2 Tuning of the classifier

Training phase is usually followed by a tuning phase where some of the parameters of SVM are tuned and results are compared in order to have the best fit via cross validation set. This is the phase where the parameters of the classifier are fixed to be used in the last phase, prediction.

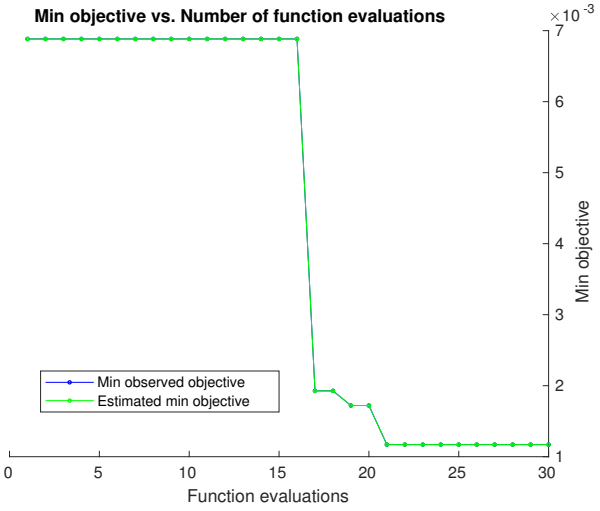

Fig. 11 Convergence of minimum objective function

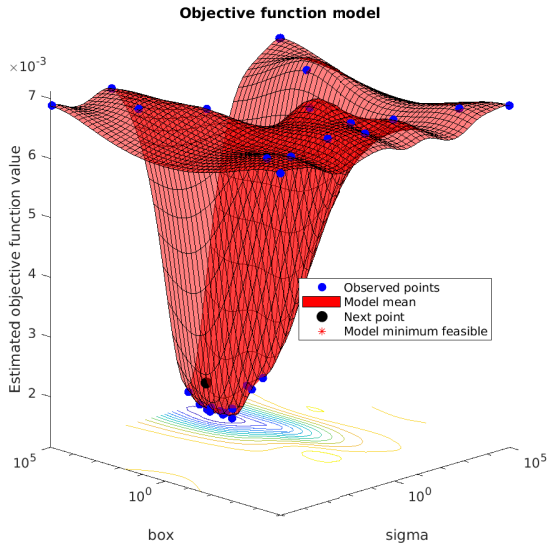

Fig. 12 Objective function values for different box parameter and sigma values

To avoiding overfitting, which is the main problem of parametric discrimination approaches such as neural networks, C (box constraint) and the kernel scale (sigma), are tuned. The classifier is trained with the training set and tuned via cross validation set, and then the selected classifier is evaluated with the test set. Cross-validation set selection of Matlab utilizes a random selection over the data set. To compare different methodologies for tuning and also the untuned classifiers, the script has been revised to generate random values from the same seed value to be consistent in comparisons. Box constraint (C) and kernel scale (sigma) are tuned considering the presence of the outliers to generalize the distribution of the data rather than resulting in fine fits for each individual data in the training set.

Two different ways are utilized to tune the classifier, heuristic and Bayesian optimization. In heuristic optimization, the strategy is to try a geometric sequence of the kernel scale (sigma parameter) scaled at the original kernel scale. Also a geometric sequence of the box constraint parameter, 11 values, from 1e-5 to 1e5 by a factor of 10 have been tried. Increasing box constraint might decrease the number of support vectors, but also might increase training time. Usually, increasing box constraint too much induces overfitting (high variance).

Bayesian optimization tool from Matlab can be used in conjunction with the classification tool to optimize box constraint and the kernel scale. This tool outputs minimum objective value as a function of number of function evaluations (max. number of function evaluations can be changed by passing arguments to the function) as shown in Fig. 11. Fig. 12 shows the objective function values for a variety of box constraint and kernel scale values. The optimized values for box constraint and kernel scale will be presented in Tables 1 - 6 for each classification problem considered.

With a satisfactory result of the training & tuning is followed by the prediction where the classifier predicts if the new measurement data belongs to the faulty or nominal class.

5.3 Evaluating the classifier

The performance of the classifier is first quantified by *loss of classification* as indicated with *kFoldLoss* in this work. The training data set is separated to 10 folds. For each fold, the *loss of classification* computed for in-fold observations with a model trained on out-of-fold observations. Finally the *loss of classification* is calculated as the *classification error* averaged over all folds. The idea to split and evaluate the performance of classifiers on different data sets is to avoid overfitting since the fitted classifier would give better results on the data set that it learnt from, but might not generalize well to new data. For that reason, both for evaluating the classifiers during tuning phase and finally evaluating the final classifier possessing the tuned parameters, were realized with different data sets.

Another means to evaluate performance of a classifier, available under Matlab Statistics and Machine Learning Toolbox, is via observing the *classification edge*. The *edge* is the weighted mean of the *classification margins*. Here, the *classification margin* for a binary classifier is defined, for each observation, as the

difference between the *classification score* for the *true class* (faulty measurements in the considered problem) and the *classification score* for the *false class* (nominal measurements in the considered problem). In this definition, *classification score* is considered as the signed distance from each observation to the decision boundary.

While all these variables would be satisfactory for performance analysis of classifiers, due to the skewed-class nature of the problem, another means of evaluations is necessary. When the number of instances for different classes in a data set has a big difference, it is named as a skewed-class problem. The inherent trickiness to evaluate classifiers for such problems lies on the fact that, predicting only the more frequent class might lead to a misunderstanding that the classifier gives superior performance although it might not be even learning at all. For the problem of fault detection of a control surface would serve as a good example to clarify more. Since nominal data is not difficult to generate in real flight while it is difficult to fly faulty, the nominal data is much vast compared to the faulty data

For such problems, in order to define a single metric to evaluate the abilities of classification, *precision* and *recall* should be defined. Fig. 13 shows the confusion matrix which is used to calculate the precision and recall. Here, in general, the class indicated by 1 is the skewed class, corresponding to the fault class in this study. True positive refers to the number of instances that are predicted faulty and are actually faulty, false positive refers to the number of nominal instances which are miscalculated or predicted as faulty, false negative is the number of instances that are actually faulty but the classifier predicted that they are nominal, and finally the true negative is the number of instances that are predicted as nominal and are actually nominal. Precision gives a measure on the fraction that was actually faulty of all the measurements that was predicted as faulty. Precision can be related to number of false alarms which is cautiously avoided in flight control systems. Precision is defined as

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{7}$$

Recall refers to the fraction of correctly detected faults of all the situations that was actually faulty. Recall can be related to the sensitivity of diagnostic systems. Actually an ideal health monitoring system is the one which accomplishes a reasonable balance between the false alarm rate and ability to detect reasonably small faults.

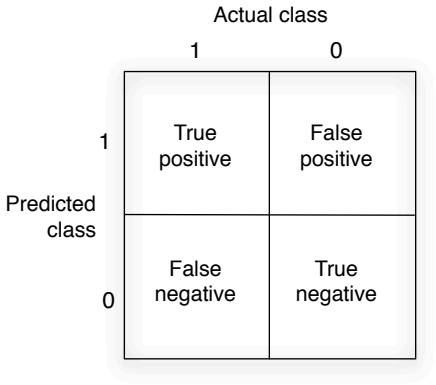

Fig. 13 Confusion matrix

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{8}$$

Finally as a single metric to evaluate the performance of the classification, F1 score is defined as

$$f1Score = \frac{2 * precision * recall}{precision + recall} \tag{9}$$

F1 score, indicated as f1Score in the tables throughout this study is used as the main parameter to evaluate classifiers.

6 Results from real flight

SVM classifier has been trained and tuned with the flight data, generated following the steps covered above, in order to classify the faulty and nominal classes. The codes have been developed under Matlab coding environment. Some tools, such as calculation of *f1Score*, were written as Matlab scripts rather than changing some arguments of toolbox functions, since they were not found available in the toolbox either it does not exist or was difficult to obtain.

The simulations have been executed on HP Z820 Workstation with 3.1 GHz, 32 cores.

This work investigates only binary classification, not multi-class classification considering multiple different faults. But different faulty cases have been studied to further attack the problem of multi-class classification in the future studies. The classifier's abilities in different fault conditions have been investigated. The main two types of faults of interest in this study were control

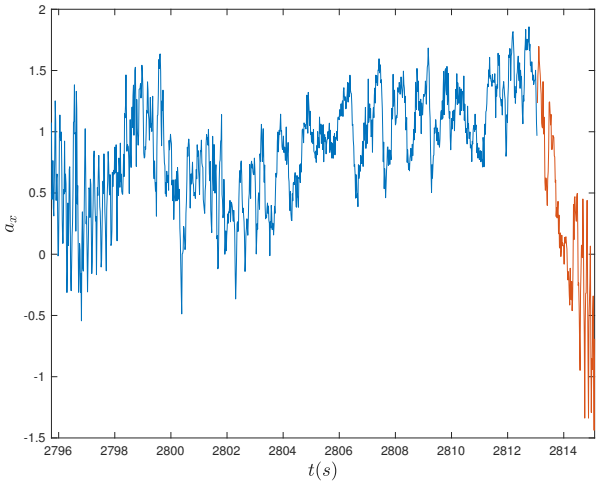

Fig. 14 Accelerometer readings along x-direction during flight interval just before control surface stuck (nominal, represented with blue) and during control surface stuck at 0° until the safety pilot's intervention

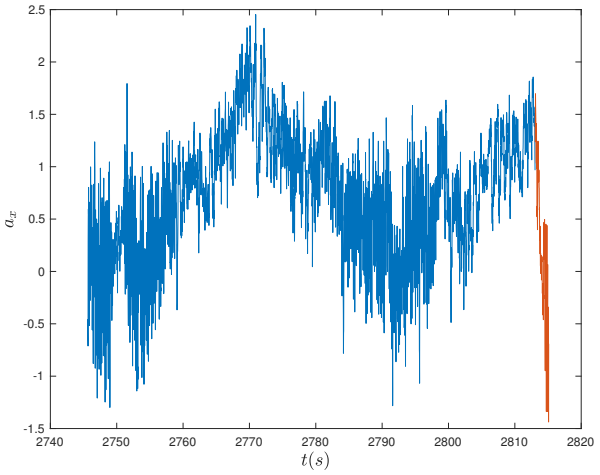

Fig. 15 Accelerometer readings along x-direction during flight interval before control surface stuck (nominal, represented with blue) and during control surface stuck at 0° until the safety pilot's intervention

surface stuck and LOE. The flight data[3] and code[4] that has been developed are publicly available under *Github*.

6.1 Control surface stuck fault

During the flights, the most challenging fault to realize was the control surface stuck. The drone reacted very fast with an uncontrollable dive towards the ground and the safety pilot initiated manual recovery by triggering the safety switch shown in Fig. 6. So the time past start-

[3] https://github.com/benelgiz/cureDDrone/tree/master/data/v4_multiplicativeAdditive_MURET_06_07_2017

[4] https://github.com/benelgiz/cureDDrone/tree/master

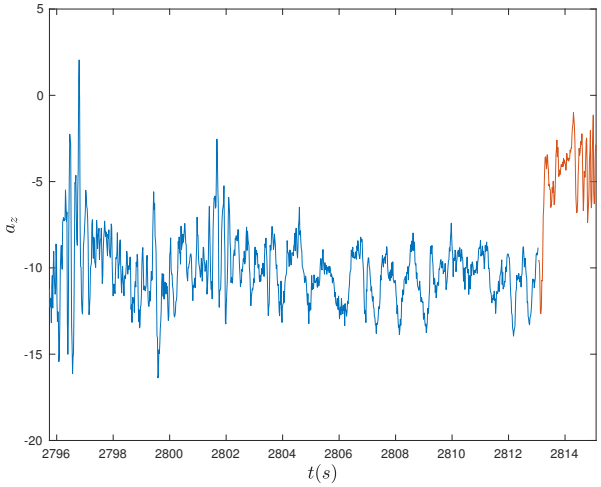

Fig. 16 Accelerometer measurements along z direction a_z for faulty and nominal flight data

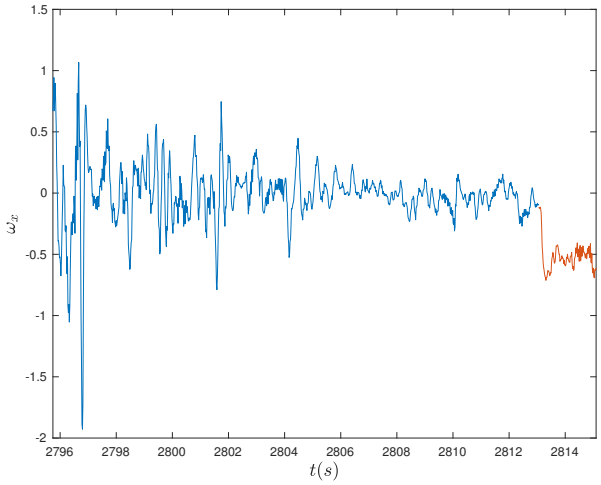

Fig. 17 Angular velocity w_x for faulty nominal flight data

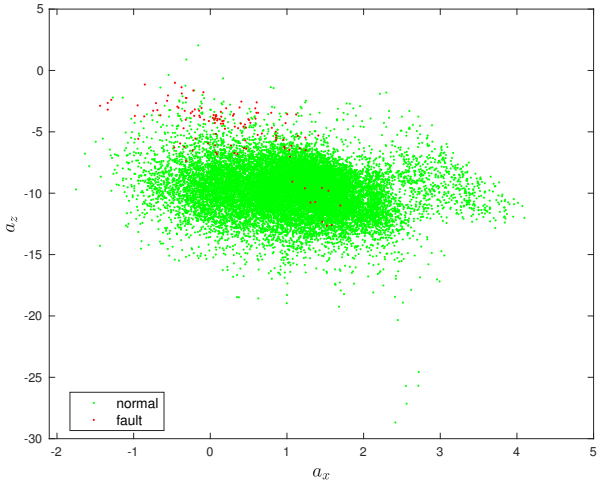

Fig. 18 a_x vs a_z feature space for faulty nominal flight data

Table 1 Untuned and tuned via heuristic approach and tuned via Bayesian optimization SVM classification evaluations

	Untuned linear kernel	Untuned Gaussian kernel	Tuned heuristic Gaussian kernel	Tuned Bayesian Gaussian kernel
kernel scale	1	1	2.1187	10.3581
box constraint	1	1	1	1323.1
margin	10.5927	2.0248	2.4763	5.3004
edge	10.5875	2.0245	2.4761	5.3004
kFoldLoss	0.2×10^{-2}	1.2×10^{-3}	7.571×10^{-4}	1.2×10^{-3}
precision	0.76	1	1	0.913
recall	0.8261	0.8696	0.913	0.913
f1Score	0.7917	0.9302	0.9545	0.913
comp. time	3.75s	3.4s	5917.5s	1784.7s

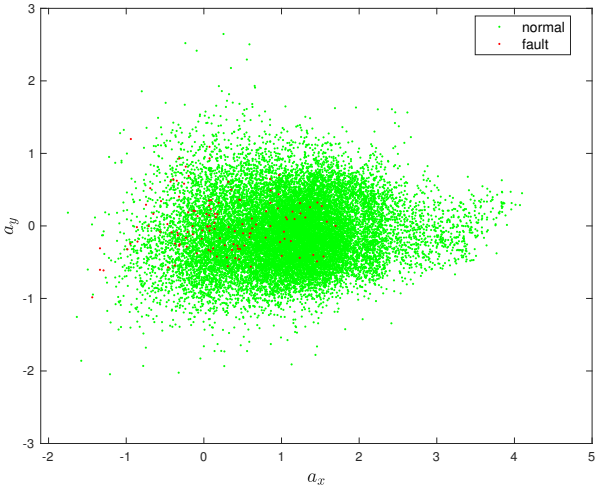

Fig. 19 a_x vs a_y feature space for faulty nominal flight data

ing from the ignition of the control surface stuck fault until the manual pilot's intervention is very short (∼2 seconds) which can be seen from the accelerometer x-direction readings in Fig. 14. This causes the problem of skew-class classification where there is a big difference between the number of instances belonging to different classes and requires some special treatment as explained above under section 5.3.

First problem considered for classification is the right elevon stuck at 0°. Gyro and accelerometer measurements are saved to SD card onboard at 60 Hz. Nominal class involves ∼5 minutes of accelerometer and gyro data while the faulty class comprised of ∼2 seconds of data, thus the problem is treated as skew-class classification.

Having knowledge about data is critical for machine learning applications. Usually, the performance of the learning algorithms are quite dependent on the level of the engineer's experience since some of the critical decision are handled by her/him such as selection of features (not for all machine learning methods). Fig. 14 shows accelerometer x-axis measurements for a duration of ∼20 seconds. Measurements plotted in blue indicates the part of the flight where elevon works correctly while red part of the plot corresponds to the faulty phase. It can be interpreted from the Fig. 14 that the fault injected shows a distinct change in the x-axis accelerometer measurements when the fault is injected. To investigate further, measurements have been plotted during a larger time scale involving ∼70 seconds of measurements as shown in Fig. 15. Here, it is seen that the accelerometer measurements corresponding to faulty phase could be observed during the nominal flight phase as well, thus it might be necessary to check the time change of translational acceleration to catch a difference in behavior. Fig. 16 and Fig. 17 show the accelerometer measurements along z-direction and angular velocities along x-direction during a short time interval. Here, although an average of faulty measurements would result in an obvious difference, individual measures might correspond to nominal phase of the flight as well. The measurements are also plotted in feature spaces a_x-a_z and a_x-a_y as shown in Fig. 18 and Fig. 19. It shows a_x-a_z space gives a distribution that would be easier to classify the faulty and nominal phases while in a_x-a_y feature space, the measures are quite similar and difficult to classify.

Table 1 shows the results of SVM classification for untuned linear kernel, untuned Gaussian kernel, and tuned Gaussian kernel via two methods (heuristic and Bayesian) respectively in its columns. Although a variety of variables used in the evaluation of classifiers have been presented to the reader for completeness, f1Score will be the main variable of concern for this study for the reasons explained before. The classifier with a *box constraint* = 2.11 and a *kernel scale* = 1 have been found to give the highest *f1Score* (*f1Score* = 0.9545).

Table 2 Untuned and tuned via heuristic approach and tuned via Bayesian optimization SVM classification evaluations

	Untuned 24 features	Tuned Heuristic 24 features	Tuned Bayesian Opt. 24 features
kernel scale	1	6.0609	67.33
box constraint	1	10	47330
margin	1.9733	3.6353	5.6484
edge	1.9678	3.6317	5.6404
kFoldLoss	5.6×10^{-3}	3.44×10^{-4}	6.19×10^{-4}
precision	1	1	1
recall	0.2632	0.8947	0.8421
f1Score	0.4167	0.9444	0.9143
comp. time	12.4s	5581.9s	1205.2s

Table 3 Untuned and tuned via heuristic approach and tuned via Bayesian optimization SVM classification evaluations

	Untuned 24 features	Tuned Heuristic 24 features	Tuned Bayesian 24 features	Untuned Heuristic 24 features %60 test/training	Tuned Heuristic 24 features %60 test/training	Tuned Bayesian 24 features %60 test/training
kernel scale	1	5.5154	24.9026	1	4.7577	24.9769
box constraint	1	10	9172.9	1	10	85.3263
margin	1.9697	4.0001	5.8651	1.9685	3.7267	4.7728
edge	1.9680	3.99	5.8651	1.9691	3.7270	4.7734
kFoldLoss	5.5×10^{-3}	4.8×10^{-4}	6.8×10^{-4}	5.4×10^{-3}	5.5×10^{-4}	7.3421×10^{-4}
precision	1	1	1	1	1	0.92
recall	0.1739	1	1	0.22	0.96	0.92
f1Score	0.2963	1	1	0.3607	0.9796	0.92
comp. time	11.3s	5853.3s	1061s	8.5s	2924.3s	235.671s

Due to previous observations on data, feature set have been widened via additions of data from past measurements for each attribute. The idea is to introduce the time change behavior of data to be represented in each instance. The number of features have been increased to 24 in Table 2, which means only 3 previous measures for each different attribute have been added to the feature set ((3 past + 1 instant) * 6 axis). This addition to the feature set, in general, deteriorated the performance of the classifier, especially the classifier with an untuned Gaussian kernel (*f1Score*= 0.4167) ($\Delta f1Score = -0.5135$). For the classifiers with tuned Gaussian kernels, *f1Score* decreases by a small amount for the classifier tuned with heuristic method (*f1Score* = 0.9444) and increases by an even smaller amount(*f1Score* = 0.9143) hence the effect is not really obvious.

Table 2 and first 3 columns of Table 3 are the results of the same algorithm with same number of features and percentage of training/test set ratio. Due to usage of random functions, the seed value has been set to a constant value for reproducibility. Stratified sampling has been implemented to assure fair amount of data from both classes in both sets (training and test). Finally, it has been discovered that even with the selection of same number of data for the training and test sets for both classes (faulty and nominal), some combinations of possible training/test set selections result in more precise classifiers than others. This can be seen by observing the tuned *f1Scores* (*f1Scores*=0.9444 in and *f1Score*= 0.9143) in Table 2 increasing to the *f1Score* (*f1Score*= 1) in Table 3.

Table 3 shows the evaluation results of SVM classifiers trained with different percentage of training and test sets. First 3 columns present the results for a test to training set ratio of 20-80 while the last 3 columns present the ratio of 40-60. *f1Score* trained and tuned with %80 training data implies that this percentage is a good choice for this problem (with an *f1Score* of 1 for tuned classifiers).

6.2 Control surface loss of efficiency fault

Loss of efficiency fault was generally more difficult to diagnose. First, a fault of %10 loss of efficiency on the left elevon has been investigated (SETTINGS 1.0 0.9 0.0 0.0). This fault was the first injected fault in the course of the flight so the corresponding nominal phase is quite long. During those first minutes of the flight, the nominal mode kept quite long to gather more nominal data from the flight before initiating the fault sequences. The results showed the insufficiency of the method to diag-

Table 4 %10 and %40 Loss of efficiency fault in left elevon classification results with Gaussian kernel

	Untuned LOE1	Tuned Heuristic LOE1	Tuned Bayesian Opt. LOE1	Untuned LOE2	Tuned Heuristic LOE2	Tuned Bayesian Opt. LOE2
f1Score	NaN	0.1235	0.0111	0.2716	0.2756	NaN
kFoldLoss	5.32×10^{-2}	5.19×10^{-2}	5.33×10^{-2}	3.84×10^{-2}	3.7×10^{-2}	5.33×10^{-2}
precision	1	0.66	NaN	1	0.66	NaN
recall	0.0016	0.119	0	0.0016	0.119	0

Table 5 %40 Loss of efficiency fault in both elevon classification results with Gaussian kernel for two different number of nominal data sets

	Untuned ∼15min	Tuned Heuristic ∼15min	Tuned Bayesian Opt. ∼15min	Untuned ∼3min	Tuned Heuristic ∼3min	Tuned Bayesian Opt. ∼3min
f1Score	NaN	0.1235	0.0111	0.2712	0.2756	NaN
kFoldLoss	4.76×10^{-2}	4.62×10^{-2}	4.73×10^{-2}	19.01×10^{-2}	18.87×10^{-2}	20.61×10^{-2}
precision	NaN	0.66	0.27	0.7109	0.6992	NaN
recall	0	0.0681	0.0057	0.1679	0.1716	0

nose the fault (see Table 4 noted as LOE1) even with a Gaussian kernel which resulted in exceptional results for stuck control surface diagnostics. One of the explanations for that might be, since the control surfaces is not moving in a wide range during the nominal course of action, %10 degradation in the movement of the elevon does not really change the dynamic behavior of the drone yet not easy to pinpoint from the measurements. Another reason, likely to be the main driver, could be the compensation of the fault via the controller. The aim of the controller onboard is to minimize the error to track a given reference trajectory and corresponding attitude references. Hence, when this error is not minimized, for one reason or another, the controller recalculates the control signal to minimize this error. So either an error due to a different wind condition or a fault in the control surface, the controller's duty is to minimize this error, so a well designed controller might handle that especially for the condition where the effect of disturbance is not catastrophic. This controller's compensation for the fault could be known by observing output of the controller and feeding this information as well in the set of instances. But, during the course of this study, the abilities of a classifier without the information from the controller is investigated. The main reason behind is to challenge the abilities of a small and cheap set of health monitoring system that does not rely on the information from the autopilot.

The loss of efficiency of the left elevon is increased to %40 degradation to investigate if this time the SVM classifier will be able to classify the fault. This fault is coined as LOE2 in the Table 4. The results show no recuperation in the classification results for neither untuned nor tuned classifiers with $f1Score = 0.0032$ and $f1Score = 0.2$ respectively.

Since the results were very poor in terms of classifying faults, the faults have been increased to have %40 degradation in both elevon. The idea is to see at which level the result of classification will improve and also to search for any reasons behind this ineffectiveness except controller's compensation. Another issue to investigate was the effect of number of instances for the larger datasets (nominal class) to classification. For that purpose, three different size for nominal data set (∼15min, ∼6min, ∼3min) was investigated in terms of their effects to classification performance for a faulty measurement data set of ∼1min. Classification results for two of the situations (∼15min, ∼3min) are given in Table 5. The reduction of the nominal data set was not random selection throughout the complete set, but only the last part of the nominal data set that is closer to the faulty data in terms of flight time instance was selected. The first simulation involved ∼15min nominal measurements, hence the most skewed classification. An untuned SVM classifier with a Gaussian kernel was not able to classify ($f1Score = NaN$) the fault. Even when tuned, the classification performance was too poor to be used for predicting faults ($f1Score = 0.12$) as shown in Table 5. Reducing the number of the larger class, gyro and accelerometer measurements during nominal phase, an untuned SVM classifier with Gaussian kernel still results poorly in classification ($f1Score = 0.0073$) and tuning does not help much ($f1Score = 0.13$). Compared to the larger nominal class measurements case, it gives the idea that reducing the number of instances of the bigger class might help to improve classification performance ($\Delta f1Score = 0.01$). A further reduction in the num-

Table 6 %40 Loss of efficiency fault in left elevon classification results with Gaussian kernel for 24 - 120 - 300 features cases

	Untuned 24	Tuned Heuristic 24	Untuned 120	Tuned Heuristic 120	Untuned 300	Tuned Heuristic 300
f1Score	0.1375	0.6414	NaN	0.9635	NaN	0.9896
kFoldLoss	0.1898	0.1194	0.2075	0.0308	0.2071	0.0091
precision	0.9545	0.7254	NaN	0.9804	NaN	0.9981
recall	0.0741	0.5732	0	0.9471	0	0.9812

ber of instances of the nominal class (∼3min) still gives poor classification results with an untuned SVM classifier with a Gaussian kernel ($f1Score = 0.2712$) although an improvement observed compared to the wider nominal data set ($\Delta f1Score = 0.2616$). Tuning the classifier did not enhance the classification result noticeably ($f1Score = 0.2756$).

Since reducing the number of instances of the larger class (nominal phase measurements) in the classification does help but not result in satisfactory results for classification, adding new attributes to the feature set is considered. For that, the previous measurements of the same attribute is added to the input matrix to have the time dependent change in the behavior of the observed physical variable. Results for a set of total number of 24, 100 and 300 features has been given in Table 6. To start with the discussion of the effect of adding previous measurements as separate features to the feature set, previous 3 measurements have been added to the input data set, resulting in 4 x 6 = 24 features. Adding 3 previous instance in time for each sensor measurement decreased the performance of the untuned classifier with a Gaussian kernel $f1Score = 0.1375$ which is an classification performance decrease of $\Delta f1Score = 0.1341$. Adding more features the classifier still not able to classify when untuned but by tuning the *f1Score* can be improved heavily producing a classifier with an *f1Score* of 0.9635 for 120 feature case and can be even more fined to a *f1Score* of 0.9896 for 300 features in total. So adding features advances the classification but only with proper tuning is achieved. Otherwise, untuned, the results are even worse than before adding features.

7 Conclusion

This work is an end-to-end design to achieve data-driven fault diagnosis for control surface faults on drones. A short survey on fault detection and diagnosis initiates the study and is followed by an introduction to SVM, the method used to classify the faults in this study. Since SVM is a supervised classification method, labeled data is necessary to train the algorithm. For that reason, an open-source autopilot *Paparazzi* has been modified to realize faulty flights to save faulty and nominal flight data to train on. Two types of faults have been mainly investigated in real flight with a small UAV, the control surface stuck and loss of effectiveness of the elevon. Results indicate that the control surface stuck can be detected relatively easily with 3 gyros and 3 accelerometers data. Addition of features to accommodate previous measurements improve classification performance for tuned classifiers while the untuned classification performance deteriorates. Classification performs poorly for loss of efficiency faults especially for smaller ineffectiveness values. Addition of features and decreasing the number of instances from larger set improves the performance. This work shows that SVM gives satisfactory performance for classification of faults on control surfaces of a drone using flight data.

Acknowledgements This work was supported by the ENGIE Ineo - Groupe ADP - SAFRAN RPAS Chair. Special thanks to Gautier Hattenberger and Torbjoern Cunis for code modifications to *Paparazzi* autopilot system, Michel Gorraz for recovering the drone as the safety pilot, Xavier Paris, and Hector Garcia de Marina, and the rest of the ENAC Drone Lab for their help during test flights. Last but not least, we acknowledge *Paparazzi* community for their contributions to the autopilot system.

References

1. Roadmap for Intelligent Systems in Aerospace (2016)
2. Angelov, P.: Sense and avoid in UAS: research and applications. John Wiley & Sons (2012)
3. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory, pp. 144–152. ACM (1992)
4. Brisset, P., Drouin, A., Gorraz, M., Huard, P.S., Tyler, J.: The paparazzi solution. In: MAV 2006, 2nd US-European competition and workshop on micro air vehicles, pp. pp–xxxx (2006)
5. Chow, E., Willsky, A.: Analytical redundancy and the design of robust failure detection systems. IEEE Transactions on Automatic control **29**(7), 603–614 (1984)
6. Ducard, G.J.: Fault-tolerant flight control and guidance systems: Practical methods for small unmanned aerial vehicles. Springer Science & Business Media (2009)

7. Etherington, T.J., Kramer, L.J., Kennedy, K.D., Bailey, R.E., Last, M.C.: Quantifying pilot contribution to flight safety during dual generator failure. In: Digital Avionics Systems Conference (DASC), 2017 IEEE/AIAA 36th, pp. 1–11. IEEE (2017)
8. Introduction of a regulatory framework for the operation of unmanned aircraft systems in the 'open' and 'specific' categories (2018)
9. Verification of Adaptive Systems (2016)
10. Gui, W.h., Liu, X.y.: Fault diagnosis technologies based on artificial intelligence for complex process. Basic Automation **4**, 000 (2002)
11. Gunn, S.R., et al.: Support vector machines for classification and regression. ISIS technical report **14**, 85–86 (1998)
12. Hagenblad, A., Gustafsson, F., Klein, I.: A comparison of two methods for stochastic fault detection: the parity space approach and principal component analysis (2004)
13. Hajiyev, C., Caliskan, F.: Sensor and control surface/actuator failure detection and isolation applied to f-16 flight dynamic. Aircraft Engineering and aerospace technology **77**(2), 152–160 (2005)
14. Isermann, R.: Fault-diagnosis systems: an introduction from fault detection to fault tolerance. Springer Science & Business Media (2006)
15. Isermann, R., Ballé, P.: Trends in the application of model-based fault detection and diagnosis of technical processes. Control engineering practice **5**(5), 709–719 (1997)
16. JARUS guidelines on Specific Operations Risk Assessment(SORA) (2017)
17. Laouti, N., Sheibat-Othman, N., Othman, S.: Support vector machines for fault detection in wind turbines. IFAC Proceedings Volumes **44**(1), 7067–7072 (2011)
18. Li, M., Li, G., Zhong, M.: A data driven fault detection and isolation scheme for uav flight control system. In: Control Conference (CCC), 2016 35th Chinese, pp. 6778–6783. TCCT (2016)
19. Melody, J.W., Hillbrand, T., Başar, T., Perkins, W.R.: H_∞ parameter identification for inflight detection of aircraft icing: The time-varying case. Control Engineering Practice **9**(12), 1327–1335 (2001)
20. Platt, J.: Sequential minimal optimization: A fast algorithm for training support vector machines (1998)
21. Platt, J., et al.: Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. Advances in large margin classifiers **10**(3), 61–74 (1999)
22. Rotstein, H., Ingvalson, R., Keviczky, T., Balas, G.J.: Fault-detection design for uninhabited aerial vehicles. Journal of guidance, control, and dynamics **29**(5), 1051–1060 (2006)
23. Schlechtingen, M., Santos, I.F.: Comparative analysis of neural network and regression based condition monitoring approaches for wind turbine fault detection. Mechanical systems and signal processing **25**(5), 1849–1875 (2011)
24. Sharma, R., Aldeen, M.: Fault detection in nonlinear systems with unknown inputs using sliding mode observer. In: 2007 American Control Conference, pp. 432–437. IEEE (2007)
25. Stutz, J.: On data-centric diagnosis of aircraft systems. IEEE Transactions on Systems, Man and Cybernetics (2010)
26. Sun, X., Marquez, H.J., Chen, T., Riaz, M.: An improved pca method with application to boiler leak detection. ISA transactions **44**(3), 379–397 (2005)
27. Undertaking, S.J.: U-space blueprint. web page, June **9** (2017)
28. Vapnik, V.: The nature of statistical learning theory springer new york google scholar (1995)
29. Vapnik, V., Chervonenkis, A.: A note on one class of perceptrons. Automation and remote control **25**(1), 103 (1964)
30. Yin, S., Gao, X., Karimi, H.R., Zhu, X.: Study on support vector machine-based fault detection in tennessee eastman process. In: Abstract and Applied Analysis, vol. 2014. Hindawi Publishing Corporation (2014)