Détection & diagnostic de pannes pour les drones utilisant la machine learning

Elgiz Baskaya

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Elgiz BASKAYA

Le 16 mai 2019

Détection&diagnostic de pannes pour les drones utilisant la machine learning

---

Ecole doctorale : **AA - Aéronautique, Astronautique**

Spécialité : **Mathématiques et Applications**

Unité de recherche :
Laboratoire Communications Numeriques et Algorithmes

Thèse dirigée par
Daniel DELAHAYE et Murat BRONZ

Jury

M. Eric FERON, Rapporteur
M. Cingiz HACIYEV, Rapporteur
Mme Janset DASDEMIR, Examinatrice
M. Philippe TRUILLET, Examinateur
M. Daniel DELAHAYE, Directeur de thèse
M. Murat BRONZ, Co-directeur de thèse

Fault Detection and Diagnosis for Drones using Machine Learning

by

Elgiz Baskaya

Submitted to the Applied Mathematics
on February 22, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philisophy in Aeronautics and Astronautics

Abstract

This new era of small UAVs currently populating the airspace introduces many safety concerns, due to the absence of a pilot onboard and the less accurate nature of the sensors. This necessitates intelligent approaches to address the emergency situations that will inevitably arise for all classes of UAV operations as defined by EASA (European Aviation Safety Agency). Hardware limitations for these small vehicles point to the utilization of *analytical redundancy*, rather than to the usual practice of hardware redundancy in manned aviation. In the course of this study, machine learning practices are implemented in order 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 data 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 classification ability, with a small and inexpensive chip set and without the need to access the data from the autopilot, such as the control input information. This work addresses the faults in the control surfaces of a UAV. More specifically, the faults considered are the control surface stuck at an angle and the loss of effectiveness. First, a model of an aircraft is simulated. This model is not used for the design of Fault Detection and Diagnosis (FDD) algorithms, but is instead utilized to generate data. Simulated data are used instead of flight data in order to isolate the probable effects of the controller on the diagnosis, which may complicate a preliminary study on FDD for drones. The results show that for simulated measurements, SVM gives very accurate results on the classification of the loss of effectiveness faults on the control surfaces. These promising results call for further investigation so as to assess SVM performance on fault classification with flight data. Real flights were arranged to generate faulty flight data by manipulating the open source autopilot, *Paparazzi*. All data and the code are available in the code sharing and versioning system, *Github*. Training is held offline due to the need for labeled data and the computational burden of the tuning phase of the classifiers. Results show that from the flight data, SVM yields an F1 score of 0.98 for the classification of

control surface stuck faults. For the loss of efficiency faults, some feature engineering, involving the addition of past measurements is needed in order to attain the same classification performance. A promising result is discovered when *spinors* are used as features instead of angular velocities. Results show that by using spinors for classification, there is a vast improvement in classification accuracy, especially when the classifiers are untuned. Using spinors and a Gaussian Kernel, an untuned classifier gives an F1 score of 0.9555, which was 0.2712 when gyro measurements were used as features. In summary, this work shows that SVM gives a satisfactory performance for the classification of faults on the control surfaces of a drone using flight data.

Thesis Supervisor: Daniel Delahaye
Title: Professor

Thesis Supervisor: Murat Bronz
Title: Assistant Professor

Acknowledgments

First of all, I would like to thank to my PhD advisors, Professors Daniel Delahaye and Murat Bronz, for supporting me during these past three years. I am grateful to Daniel Delahaye, for his insight during the thesis and directed me to the very interesting topic of machine learning. I would like to thank Murat Bronz for initiating the PhD opportunity, his guidance throughout this thesis, his help for the modifications to the *Paparazzi* autopilot and being the safety pilot during the flight campaigns. I would also like to thank Stephane Puechmorel for his insight that lead to interesting results also discussed in this thesis.

The drone team in ENAC, Yannick Jestin (the best boss), Michel Gorraz, Fabian Garcia, Alexandre Bustico, Fabian Bonneval. You guys are great, it was such a pleasure.. Extra thanks for their help on flight campaigns to Torbjoern Cunis, Gauthier Hattenberger, and Michel Gorraz. Special thanks to Torbjoern for even cooking me dinner during difficult times. And to the triple sec; Guido Manfredi and Jim Sharples; I am grateful to you for the all the shared moments and your endless support. And the drone team is luckily even larger, I am so happy that we worked all together with Jean-Philippe Condomines, Xavier Paris, Titouan Verdu, Yuchen Leng, Baizura Bohari. All colleagues and friends with whom my last three years has passed. BEST team ever. Felt like home. Thank you all..

My family, my mom Nergiz and father Adiguzel, I am grateful to you most, for the most precious gift you gave me, my life. Mom, you are the reason of my freedom, my imagination and my free will. I always admired you being competent since I first saw you at work, passionately talking on your profession, eyes closed, hard to believe that it was my mother. And dad, you are the reason for my artistic spirit, which tries to burst out of me sometimes. You both always supported me on music, always supported my choices, I was lucky to be grown as a free spirit, I think this is where I am grateful to you the most.

My brother, you are really special to me. It is enough to make me happy just knowing you are there.

My grandmom, nananem. The cleverest women I have ever known, lucky to be raised by you. I wont forget that you being the person that I get along the best, never even an argument, like magic. My flees to captain's bridge has never been punished on the way along the sea to summer house, my never finishing questions never left unanswered, my passion for sea shells and flower seeds plugging the washing machines have been ignored.

My uncle Cengiz, where my logic genes have been inherited. Again, the person I have never argued even once in my lifetime, no delays in communication, or may be sometimes when something else is been processed. The best person I have ever known in my life, an angel. Too clever, too talented. And my lovely aunt Leyla, who woke me up everyday, patiently, thank you many times.

Rapunzel, my companion, you have helped me a ton, you made me graduate. Although you wont be able to read it, you won't be the first one.

Ertan, you have helped me numerous times I stopped counting, my crisis, my anxieties, my vulnerabilities. Those days I was about to quit, you made me back, not let me to leave Rapunzel, told me that I can do it, and I did it.

I also thank my friend, my sister, Hale, for providing support and friendship that I needed.

Utku, ne 3 u la, my perfect lab mate. Hope to work AI together here or there, one day..

Estelle, I am thankful for what we have shared during three years, and grateful for her help, especially during the last phase of this thesis.

Helen, thanks to her, the paperwork which I hated with a passion became much of an ease. Such a nice person, kind, and gentle. Thank you very much.

I am also grateful to Sara, Micheal, and Magda for proofreading the thesis, and Sarah for her incredible illustrations and the cover of this thesis.

This work is supported by ENGIE Ineo - Groupe ADP - Safran RPAS Chair. I am grateful for their support that enabled me to pursue three years of research on this interesting subject. I am also grateful to Catherine Ronfle-Nadaud for her support during the early stages of *Chaire Drones*.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Lately, the popularity and reachability of Unmanned Aircraft System (UAS) have risen steeply thanks to the advancements in electronic components and their decrease in cost. This accelerating trend towards small but capable flying vehicles is extending the limits of hardware and software potential in industry and academia. Regulators from the aviation community have become increasingly concerned by the number of incidences of drones flying close to civil aircraft and airports. With the advent of the new era of UAS, different institutions around the world are addressing safe integration of UAS into airspace [9], specifically the National Aeronautics and Space Administration (NASA) [56] and the Federal Aviation Administration (FAA) [62] in the US, European Aviation Safety Agency (EASA) [36] in Europe and international bases such as International Civil Aviation Organization (ICAO) [48]. The U-Space concept in Europe (UTM in the US, which aims at enabling safe integration into airspace), provides the insight by showing in their roadmaps that the level of drone automatization and the level of drone connectivity (drone to drone and drone to infrastructure) will guide the pace of the U-Space services (U1 to U4) [93]. These enablers will allow intelligent agents to share information and automate complex procedures in case of emergencies. Thus, future aviation will inevitably move towards automatization. Here we propose a concept that will contribute to the automatization of drones, to make them more intelligent and thus contribute to their safer integration into the airspace. These awareness abilities allow the mitigation of risks in accordance with

the risk assessment procedures offered by JARUS (SORA) [55] defined by [38]. The functioning abilities of a drone following an emergency should be assessed and, depending on the availability of the environment, a recovery procedure may be initiated. For cases in which recovery is not possible, safe ditching may be required in order to reduce the potential harm in the air or on the ground.

1.1 Motivation

Improving the flight reliability is considered to be one of the main concerns for integrating UAVs into civil airspace, according to the Unmanned Systems Roadmap by the US Office of the Secretary of Defense, DoD [82]. Achieving safe flight is not a straightforward task, considering the multiplexity of unknowns in the system hardware, environment, and possible system faults/failures yet to emerge. The expectation that UAVs should be less expensive than their manned counterparts effects their reliability. Cost saving measures — other than the need to support a pilot/crew onboard or a reduction in size — may lead to a decrease in system reliability.

Under the research and development programs and initiatives identified by the DoD to develop technologies and capabilities for UAS, the greatest area of control technologies is health management and adaptive control, with a budget of 74.3 m dollars. Other safety features such as the validation and verification of flight critical intelligent software are the second area, with 57.8 m dollars [82].

Systems are often susceptible to faults of differing nature. Existing irregularities in sensors, actuators or controller may be intensified due to the control system design and lead to failures. A fault may be hidden due to the control action [30].

The most widely-used method to increase reliability is to increase the use of more reliable components and/or hardware redundancy. Both require an increase in the cost of the UAS, conflicting with one of the main reasons of UAS' popularity [5]. In order to provide solutions for the different foreseen categories within the airspace, a variety of approaches should be considered. While hardware redundancy may cope with failure situations of UAVs in certified operations, it may not be suitable for UAVs

in open category operations or some operations in specific category, due to budget constraints. Analytical redundancy is another solution. This may be not as effective and straightforward as hardware redundancy, but relies on the design of intelligent methods in order to utilize the information on board the aircraft to deal with all potential circumstances.

1.2 Contribution

The integration of drones into the airspace requires the introduction of ingenious design to ensure safe solutions for drones. This necessitates intelligent approaches to address all potential emergency situations. One of these aspects is to ensure safe flight by designing fault detection and diagnosis systems using less expensive avionics, common in a vast number of drones. The hardware limitations for these small vehicles point to the utilization of *analytical redundancy* rather than the usual practice of hardware redundancy in manned aviation. For that purpose, we introduce an end-to-end design to achieve data-driven fault diagnosis for control surface faults on drones. In the course of this thesis, machine learning practices are implemented in order to diagnose faults on a small fixed-wing UAV, and to avoid the burden of accurate modeling required in model-based fault diagnosis. We aim to design a classifier via SVM to solve fault detection and diagnosis (FDD) as a classification problem for drones with actuator faults. All data and code are available in the code sharing and versioning system, *Github*.

In this thesis, fault classification simulations are investigated under two main sections: first, the classification of faults based on simulated flight measurements; and second, the classification of faults based on real flight data.

For the classification on data generated from simulations, a model of a MAKO UAV [8] is simulated. Sensor measurements (accelerometer and gyro data) have been calculated using information on the drone's motion and the specifications of the sensors. Generated data is usually more structured compared to real flight data. In this preliminary application of SVM to fault diagnosis, we started with simpler

problem and used data generated from models. Results show that the SVM classifier was accurate and fast in diagnosing faults on the control surfaces, with a classification accuracy of 99.999%.

Accurate results on the classification of faults with generated data have thus encouraged us to further investigate fault detection with real flight data. Since SVM is a supervised classification method, labeled data is necessary to train the algorithm. Real flights have therefore been arranged in order to generate faulty flight data by manipulating the open source autopilot, *Paparazzi*. Training is held offline due to the need for labeled data and the computational burden of the tuning phase of the classifiers. Two main types of faults have been investigated; the control surface stuck fault and the loss of effectiveness of the elevon. Results indicate that the control surface stuck fault can be detected relatively easily with the data from three gyros and three accelerometers, compared to the loss of effectiveness (efficiency). The results show that over the flight data, SVM yields an F1 score of 0.98 for the classification of the control surface stuck fault. The addition of features to accommodate previous measurements improve the classification performance for tuned classifiers while the performance for untuned classification deteriorates. Classification performs poorly for the loss of efficiency faults, especially for the smaller ineffectiveness values. For the loss of efficiency fault, some feature engineering — involving the addition of past measurements — is needed in order to attain the same classification performance.

A very promising result is discovered when *spinors* are used as features instead of angular velocities. Result show that by using spinors for classification, there is a significant improvement in classification accuracy, especially when the classifiers are untuned. Using spinors and a Gaussian Kernel, the untuned classifier gives an F1 score of 0.9555, which is 0.2712 when the gyro measurements are used as features.

In general, this work shows that SVM gives a satisfactory performance for the classification of faults on the control surfaces of a drone using flight data.

This thesis contributes to the literature with the following papers:

- Baskaya, E., Bronz, M., & Delahaye, D. (2017, September). Fault detection & diagnosis for small UAVs via machine learning. In Digital Avionics Systems

Conference (DASC), 2017 IEEE/AIAA 36th (pp. 1-6). IEEE,

- Baskaya, E., Bronz, M., & Delahaye, D. (2017, September). Flight Simulation of a MAKO UAV for Use in Data-Driven Fault Diagnosis. In IMAV 2017, 9th international microair vehicle conference,

- Baskaya, E., Manfredi, G., Bronz, M., & Delahaye, D. (2016, September). Flexible open architecture for UASs integration into the airspace: Paparazzi autopilot system. In Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th (pp. 1-7). IEEE.

1.3 Thesis Outline

Chapter 2 presents the state of the art on the following topics:

- the integration of drones into the airspace;
- *Paparazzi* autopilot;
- fault tolerant control systems with a focus on fault detection and diagnosis;
- machine learning and artificial intelligence in general.

The safe integration of drones into the airspace is the motivation for this thesis. To ensure this safe integration, the design of vehicle health management systems is crucial. Fault detection and diagnosis plays an important role in vehicle health management, and we provide a literature review of this topic. In this thesis, the method selected to detect and diagnose faults is Support Vector Machines (SVM), which is a machine learning method. Thus, the current state of the art for machine learning is also provided. *Paparazzi* autopilot has been used to realize flights to generate faulty data. This is also introduced in here.

Chapter 3 focuses on the equations of motion of an aircraft. Equations for translational and rotational motion are discussed separately and in detail. After the equations are derived for a generic aircraft, the calculations of forces and moments which

are specific to an individual drone are presented. Then, modeling accelerometer and gyro data is explained, followed by a discussion of modeling faults in the control surfaces of an aircraft.

Chapter 4 is dedicated to machine learning algorithms with a focus on implementation. After the general practice of machine learning applications is presented, the use of Support Vector Machines to detect and diagnose faults is discussed in more detail.

Chapter 5 focuses on fault classification results under two main sections: first, the classification of faults based on simulated flight measurements; and second, the classification of faults based on real flight data. In the first part, flight data is simulated using the mathematical equations explained in Chapter 3. The SVM algorithm is trained with the simulated data in order to classify the faulty and nominal phases of the simulated flight. The second part commences with explanations on realized faulty flight experiments. *Paparazzi* autopilot has been modified to introduce faults to control surfaces during flight. Then, the classification of the control surface stuck fault and the loss of effectiveness fault are investigated separately.

Chapter 6 concludes this thesis with a review of the contributions and results made.

Chapter 2

State of the Art

This chapter includes the current state of the art for a variety of topics. The integration of unmanned systems into airspace is the first topic addressed as it is the motivation for this thesis. The second topic addressed is the *Paparazzi Autopilot System* since it is used to realize the flight campaigns for this thesis. We also discuss how the use of the Paparazzi open source auto-pilot system can assist with the integration of unmanned aircraft system (UAS) into the airspace. Then, to achieve this safe integration, fault tolerant control systems have been examined with a focus on fault detection and diagnosis. Finally, the rise of machine learning methods in the last decade is discussed as they are utilized to diagnose faults in this study.

2.1 Integration of Drones into Airspace

Commercial advantages offered by drones are already targeted by big companies worldwide. The airspace regulatory authorities seem to be caught between the companies (that demand rapid access to airspace), and the concerns of the public about potential privacy breaches, safety and liability issues [29, 41]. Even with today's strictly regulated airspace, reported occurrences show that there are a number of hurdles to overcome before the further integration of UAS into the airspace.

With the Riga Conference[1] in 2015, Europe has started on a path towards a unified

[1]The Future of Flying, Conference on remotely piloted aircraft systems, Riga, 6 March 2015

Figure 2-1: Statue of mermaid in Riga handling airspace (for fun)

regulatory framework design in order to allocate airspace with the growing number of drones. Its diversity, innovation and international structure is offering huge potential for new jobs. EASA (European Aviation Safety Agency) has been assigned by the European Commission to develop two main aspects:

- EU Regulatory Framework for drone operations;
- proposals for the regulation of low-risk drone operations, key elements of the future Implementation Rules (IRs).

With a starting point of the Riga Declaration [73], and building on Regulation (EC) No 216/2008 (Basic Regulation) [76], the A-NPA (Advance Notice of Proposed Amendment) [36] introduces three main categories of operations and asks for public consultation. In the report, drones are grouped under two main categories; remotely piloted and autonomous. An autonomous drone is defined by the ICAO (International Civil Aviation Organization) [52] as: *A drone that does not allow pilot intervention in the management of the flight.* Aside from sounding like science fiction, one of the key reasons that EASA has switched to using the term 'drone' and categorizing as remotely piloted or autonomous — rather than using UAS or RPAS — is to ensure the fast growing development of autonomous drones.

While mentioning the efforts by different organizations to accommodate drones in airspace, we examine the present regulatory context from European and international perspectives. The ICAO is a United Nations Organization working with 191 Member States of the Chicago Convention (Convention on International Civil Aviation) and global aviation organizations. At the present time, the international circulation of drones is not allowed by Article 8 of the Chicago Convention [51]. Starting in 2003, the ICAO began a study on the Standards and Recommended Practices (SARPs) for drones, to which member states refer when developing their legally enforceable national civil aviation regulations. 2018 is the proposed year for draft SARPs with a focus on international drone operations. Until then, the work of the ICAO can be traced by studies of the UAS Study Group (set up by ICAO in 2007), which developed the Circular 328 AN/190 on Unmanned Aircraft Systems [48], an amendment to Annex 2 (Rules of the Air) [49] and Annex 7 (Aircraft Nationality and Registration Marks) [50]. From a European perspective, Basic Regulation (Regulation EC No.216/2008) [76] is that which holds at present. Article 2 and Annex II limit the regulation to apply only to drones with a maximum take off mass above 150 kg and/or except operations on military, customs, police, firefighting, search and rescue, and experimental work. Those that are not covered by Basic Regulation are handled by national aviation legislation, which are not yet harmonized and mostly designed with an assumption that small drones are operating locally. EASA is working on different aspects of drone integration into the airspace. Last year, EASA published the 'Prototype' Commission Regulation on Unmanned Aircraft Operations [33], after public consultancy with A-NPA 2015-10 (Advance Notice of Proposed Amendment) [36], and its resulting Opinion [32] with an Explanatory Note of Opinion. Another work from EASA aims to authorize general principles for the *type certification* for fixed wing and helicopter drones separately, using a policy adopted by the agency in 2009 (E.Y013-01) [31].

In the proposed regulation by EASA, an operation and risk centric categorization has been followed. *Open*, *specific*, and *certified* are the three categories proposed, corresponding to the different risk levels that they entail.

- **Open Category** encompasses low-risk operations, such as most leisure flights and some professional activities. Operations falling in this category do not require explicit authorization from civil aviation authorities. However, they are subject to strict operational limitations (e.g., no proximity to people, traffic, infrastructures; no dangerous items; one pilot per UAS; no item dropping). These operational limitations are sufficient to mitigate the low risk. Though some professional activities fit into this category, the main goal is to regulate leisure types of operation [63].

- **Specific Category** regroups medium-risk operations, such as operation beyond visual line of sight (i.e., no visual contact between pilot and UAV during flight). To facilitate the regulation process, several scenarios with specific operational limitations are designed. To operate within a scenario, the UAS needs to comply with a list of requirements. For operations outside the scope of those scenarios, a risk analysis must be carried out to show that the existing risks are properly mitigated. To facilitate this risk analysis process, JARUS has developed a framework called the Specific Operations Risk Assessment (SORA). SORA considers threats — which contribute to the risk — and barriers — which mitigate the risk — in order to evaluate the actual risk of an operation, and decide if the resulting mitigated risk is low enough to allow the operation, as illustrated in Fig. 2-2. The risk analysis needs to be validated by the authorities in order to authorize an operation not included in the standard scenarios [63].

- **Certified Category** Operations with risk that cannot be mitigated in *Specific Category* are evaluated in *Certified Category*. These represent high-risk operations, such as a large cargo delivery in urban area. These are likely to be operations with a risk close to current manned aviation. As such, the aircraft, avionics, pilot/crew and operator will need to be certified in order to fly these operations. The fact that a certification process is involved increases the complexity of the introduction of UAS in these types of operations. Indeed,

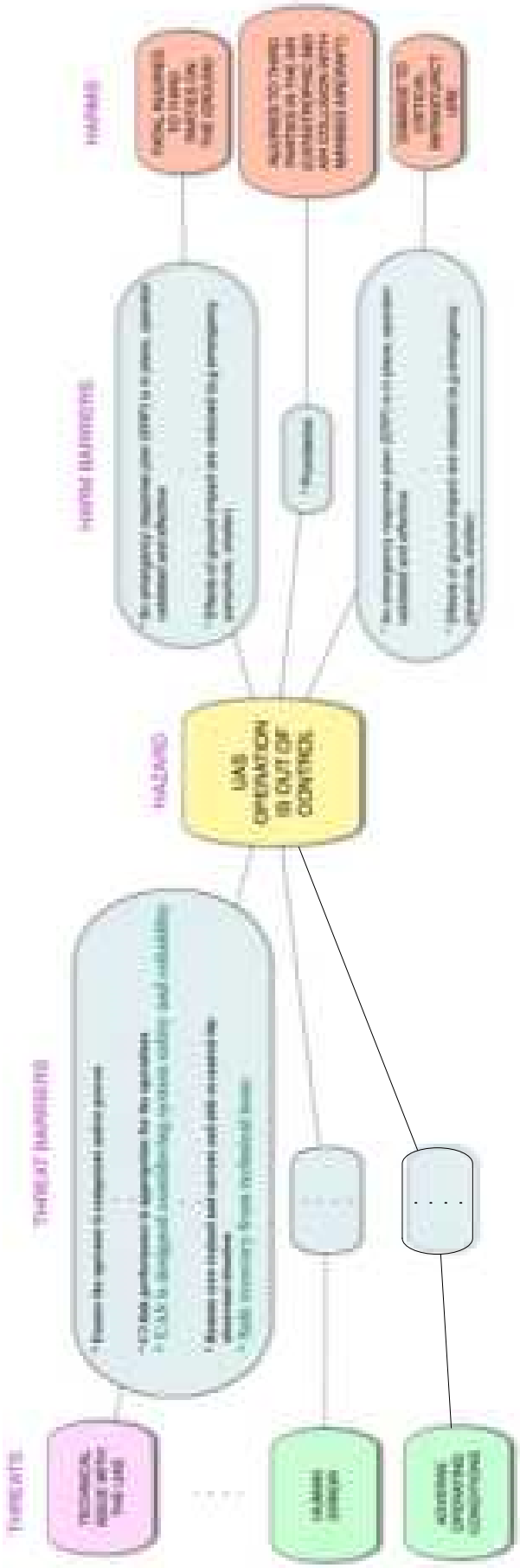

Figure 2-2: SORA structure: threats, threat barriers, hazard, harm barriers, harms

Figure 2-3: U-space illustration [20]

before being able to certify a piece of equipment, a standard must be developed for this equipment. Standardization can be best understood as the process of defining details and minimum requirements for safe and uniform operations across a diverse range of implementations. However, at present, not all of the parts required to fly a UAS have the corresponding standards, e.g., Detect And Avoid (DAA) systems, C2 links, pilot training. Thus, enabling this category of operation requires extra effort and time for the industry so as to agree on standards. For *Certified Category*, harmonization at the ICAO is planned to allow international operations [63].

Latest efforts for the integration of drones into airspace have welcomed a relatively new concept, called the 'U-Space'. This approach has been adopted for immediate plans at the European level, to accommodate drones in urban areas. While EASA continues to search for solutions for the integration of drones in a variety of presumed airspaces, Commissioner Violeta Bulc has mentioned other pillars at a high level conference[2] in Warsaw. She offers that in parallel to the efforts of EASA, there are two other pillars that need to be tackled; UTM (UAS Traffic Management) and the U-

[2]Drones as a leverage for jobs and new business opportunities [74]

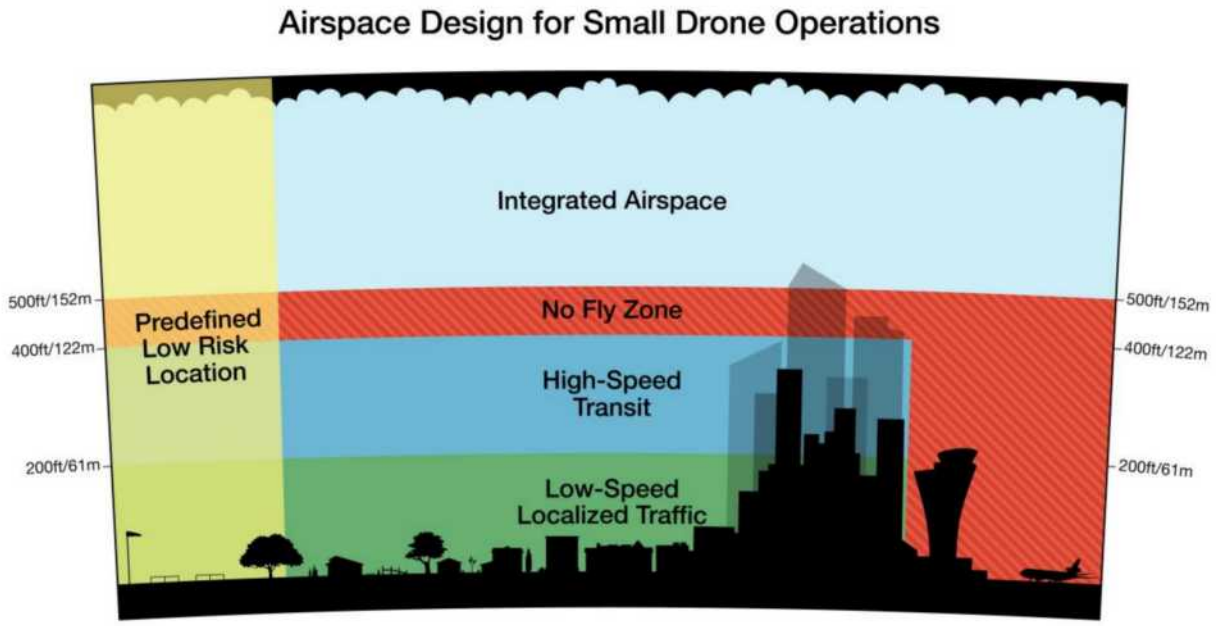

Figure 2-4: Airspace design for small drone operations by Amazon [4]

Space. U-Space covers the usage of drones especially in 'U-rban' areas with an equal share of airspace, in which case '*U*' stands for '*you*'. To illustrate its availability to everyone, the services will be available through a mobile phone, as shown in Fig. 2-3. The goals are highly challenging [20]:

- "Safe: safety at low altitude levels will be just as good as that for traditional manned aviation. The concept is to develop a system similar to that of Air Traffic Management for manned aviation."
- "Automated: the system will provide information for highly automated or autonomous drones to fly safely and avoid obstacles or collisions."
- "Up and running by 2019: for the basic services like registration, e-identification and geo-fencing. However, further U-Space services and their corresponding standards will need to be developed in the future."

EASA regulations already cover the airspace issued in U-Space, either under spe-

cific or certified categories (depending on the risk of the operation). Using U-Space, a faster integration is expected in urban areas. We do not appear to be far from the images of flying vehicles populating the sky, as in science fiction — but even more advanced — without drivers, and more automated.

In the US, in order to tackle the safety challenges and help the development of regulation, NASA is currently carrying out a four-year research program (up to 2019) to enable unmanned aircraft traffic management solutions which are structured and flexible when needed. To ensure safety, this integration needs to be achieved through airspace management and UAS reliability. In addition, preliminary airspace designs, such as that proposed by Amazon shown in Fig. 2-4, identify different zones depending on the UAS capabilities, population density and altitude.

United Arab Emirates (UAE) has accelerated the safe integration of drones into urban airspace, by settling an agreement for UAS Traffic Management (UTM) with Nokia, as part of its 'Next generation network for mission-critical and smart city services' [71]. Aiming at tracking and managing all drones in the airspace, Nokia is offering solutions for automated flight permissions, no-fly zone regulation, beyond visual line of sight (BVLOS) safety operations utilizing LTE protocols for low latency, reliable and resilient communication. For compatibility with this system, drones will be equipped with LTE dongles and access modules for telemetry data access.

2.2 Paparazzi Autopilot System

An open-source autopilot, *Paparazzi* [14], has been adapted to generate faulty control inputs during the flight. In this section, an introduction to *Paparazzi* open-source autopilot system is given. Next, we discuss the properties of open-source systems that would best serve the integration of drones into the airspace. To highlight the advantages of *Paparazzi*, the best practices that are offered by EASA in its A-NPA [36] are referred to, and the properties of *Paparazzi* to achieve this are discussed.

2.2.1 Open-source Autopilots for UAS

With the new era of First Person View (FPV) flights [40], especially for multi-rotor UAVs, there has been an exponential increase in the hardware and software of the open-source autopilots. A brief comparison of the popular current open-source and commercial autopilots is available in Table 2.1. Usually, the trend is to make the hardware as inexpensive as possible for recreational consumers. This reality is damaging to the reputation of open-source autopilots as they are considered as not reliable or robust, because they are being used without experience. Indeed, there exists a difference in the quality of sensors used in the open-source autopilot systems compared to commercial autopilots, and this is confirmed by the price of the units. This in turn makes a difference to the flight quality of the vehicles, however, with new on-board processing power, more complex estimation algorithms and filters are being used in order to overcome this problem [9].

2.2.2 Introduction to *Paparazzi Autopilot System*

The *Paparazzi* Autopilot System provides a software and hardware solution for low-cost mini and micro unmanned air vehicles. A fleet of *Paparazzi* equipped drones can be seen in Fig. 2-5.

This began as a personal project in 2003 by Pascal Brisset and Antoine Drouin, and afterwards gained the support of ENAC in 2005. Being one of the first (if not *the* first) open-source autopilot system in the world, *Paparazzi* has attracted much attention, and has led others to start new branches and systems. The software is originally packaged for Debian/Ubuntu but can be manually installed on any GNU/Linux operating system, even including MacOS-X. However, it is not compatible with Windows which automatically eliminates 90 % of the possible user community, and therefore it is not as popular as other existing autopilot systems currently on the user market.

Table 2.1: Comparison of popular open-source and commercial autopilots [9].

Open-Source	Vehicle type	Hardware	Multi UAV	Flight Plan	Path Definition	Geofencing	Collision Avoid-ance	Latest stable re-lease
Paparazzi	Fully configurable	varied	yes	scriptable	Formula Based	3-D	UAV TCAS	21-06-16
PixHawk	Fully configurable	specific	yes	scriptable	Circles Lines Patterns	3-D	none	06-08-16
Ardupilot	Copter Plane Rover	varied	alpha	scriptable	Circles Lines Patterns	2-D	none	22-06-16
OpenPilot	multicopter	specific	none	NA	NA	none	none	15-05-15
AeroQuad	multicopter	varied	none	NA	NA	none	none	31-01-13
Commercial								
Picollo	Copter Plane	specific	none	dynamic	Way Point	NC	NC	NC
MicroPilot	Copter Plane Rover Blimp	specific	yes	dynamic	Way Point	NC	NC	NC

Figure 2-5: Fleet of *Paparazzi* equipped drones from ENAC UAV laboratory in a flight campaign with the *Pyrenees* in the background. Photo by Alexandre Bustico.

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, as shown in Fig. 2-6. The ground software of *Paparazzi* is mainly written in Ocaml, with some additional parts in Python and C. 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. Airborne software is written in C, however there is an on-going effort to support C++ [9].

One of the distinguishing features of *Paparazzi* is to support multi-UAV flights (Figure 2-7). Several projects have been using *Paparazzi* Autopilot System because of this additional feature. The communication with each vehicle is established by the ground control station, however air-to-air communication between flying vehicles and the internal relay of information to the ground control station is being implemented (for a future version).

Modules are the easiest and most flexible way of adding new code into *Paparazzi*.

Figure 2-6: Paparazzi Autopilot System overview

There are over 130 modules written for *Paparazzi* by several developers and researchers, including meteorology, imagery, surveillance, advance navigation, formation flight, and collision avoidance, etc. A photo taken during a *Paparazzi* mission in a campaign for meteorological studies by *MeteoFrance* is provided in Fig. 2-8. MeteoFrance is French national meteorological service, and uses drones as tools to conduct research for meteorological studies.

Paparazzi has its own complete flight plan language, where the user 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.

A real-time operating system based on ChibiOS has been used since 2015. The first implementation was mainly for adding a separate thread in order to make on-board logging at a high frequency. On going work is to divide all autopilot tasks into individual threads and manage everything according to priorities which will increase

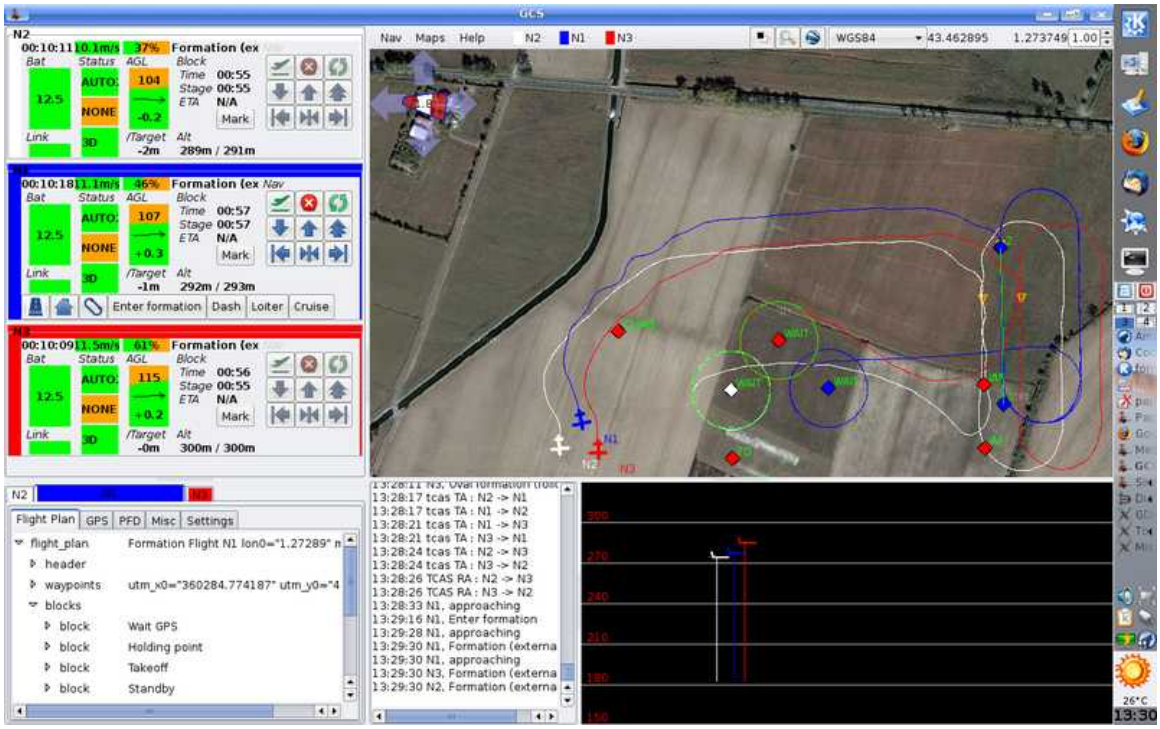

Figure 2-7: Ground Control Station showing trajectories of 2 UAVs using the TCAS system

Figure 2-8: Drone equipped with *Paparazzi* on a mission for meteorological studies by *Meteofrance*. Photo taken by Alexandre Bustico.

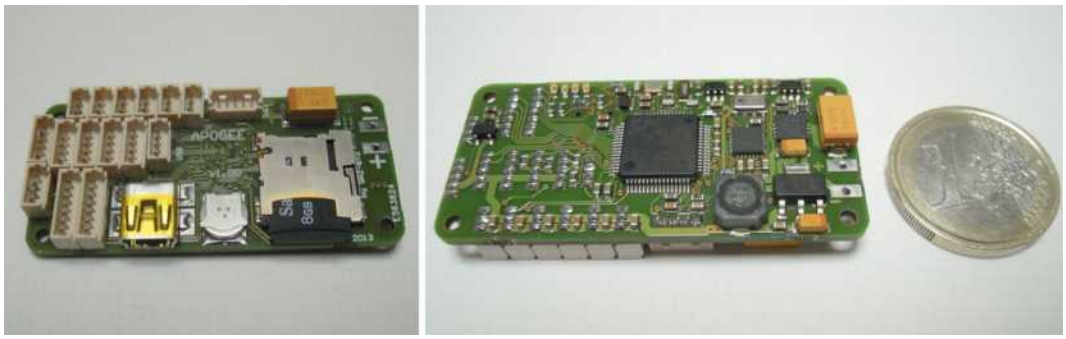

Figure 2-9: Front and back view of *Apogee* autopilot

the safety and reliability.

There exists over 20 different autopilot boards capable of running *Paparazzi*. Additionally, several mission-based custom sensor boards have been designed under the *Paparazzi* project, such as *Meteo-Stick*[3]. In this thesis, flights have been realized with an *Apogee*[4] which is shown in Fig. 2-9.

For the faulty flight data gathering necessary for this thesis, some modifications to the *Paparazzi* autopilot were necessary: first, injecting the real-time faults from GCS; and second, editing the controller on board so that the sent faulty input values configure the servos when changed from the GCS.

The rest of this section highlights the capacity of *Paparazzi* capacity to tackle issues encountered during the integration of Unmanned Aircraft System (UAS)s into the airspace. These issues have been grouped into three categories: modularity, congestion management and reliability. For each category, *Paparazzi* offers a unique set of features to deal with these issues.

2.2.3 Open-Source Systems in Relation to the New Regulatory Context

In this section, we present the *Paparazzi* open-source auto-pilot system and its features, as a tool towards the safe integration of low altitude UAS into the National

[3]http://wiki.paparazziuav.org/wiki/MeteoStick_v1.10
[4]https://wiki.paparazziuav.org/wiki/Apogee/v1.00

Air Space (NAS). *Paparazzi* is favored thanks to its modular design, its support for congestion management and evolving reliability. We believe that the flexibility required for such solutions calls for open architecture. To ensure safety, this integration needs to be achieved through airspace management and UAS reliability. Preliminary airspace designs (for example, Amazon's design) identify different zones depending on the UAS capabilities, population density and altitude. Furthermore, the evolution of national rules increases the push to cope variety of requirements. Open-source and modular architectures are keys to successfully adapting to these requirements. From UTM point of view, *Paparazzi* provides features to ease congestion management, such as dynamic geofencing, trajectory communication and collision avoidance. Concerning reliability, current regulations focus on flight constraints but may be expected to involve regulations on software and hardware components as well. In such a case, the increased cost will be inevitable for the demands of certification. In the *Paparazzi* software case, parts of the code have been formally proved and stable versions have thousands of flight hours. Such heritage may ease the certification process for smaller companies. In addition to its flexibility and reliability, *Paparazzi* offers a unique set of features as an open-source software, so as to achieve safe integration of low altitude UAS into the airspace.

2.2.4 Modularity

The current evolving nature of regulations, and the variety of organizations in charge of the airspace rules, demands flexible solutions to cope with this unique environments. *Paparazzi*, as an open-source autopilot system, is easily modifiable through its modules to be able to offer such flexible solutions.

Airspace Categorization

The UAS in the National Air Space (NAS) project points to a performance-based routine access to all segments of the national airspace for all unmanned aircraft system classes, after the safety and technical issues are addressed thoroughly. Initially,

National Aeronautics and Space Administration (NASA) and Federal Aviation Administration (FAA) seem to have a short-term goal to integrate UAS in low-altitude airspace as quickly as possible. They further aim to accommodate increased demand safely and efficiently in the long term. NASA and FAA appear to manage the airspace above 500 feet and that below separately. European Aviation Safety Agency (EASA), tasked by the European Union, is planning a risk-based approach, accommodating the UAS in the airspace under three different categories: low risk; specific; and high risk [36]. Both regulators seem to categorize the airspace and scale regulatory needs according to certain criteria. To respond to the different needs of the different categories, flexibility given by the high level of modularity of open-source autopilot systems will be critical. Customizability of the software and hardware depending on the airspace gives a chance for the larger airspace community to utilize *Paparazzi*, scaled for their specific needs. A larger community using the same system would lead to a natural evolution of the system towards a better design.

National Regulations

International UAS flight is somewhat hampered by the Chicago convention unless an agreement holds between Contracting States [36]. The International Civil Aviation Organization (ICAO) is aiming to develop international standards and recommended practices to which the member states can refer when developing their national civil aviation regulations. Although member states are planning to refer to the same standards and national aviation legislations, those will not be exactly the same because of the different expectations of nations towards UAS aviation. Thanks to the modular nature of the *Paparazzi* software and hardware suite, the functionality of the system could be enhanced according to the specific regulations held in the area of utilization.

Accommodating evolution of regulations

Prescriptive rules seem to cause some difficulties since the technical area of the UAS systems develops rapidly [36]. Innovations, both on the aircraft and on the type of operation of the UAS, will accelerate especially after the regulations are set. Thus,

regulatory bodies call for refinable operational requirements and systems architecture to evolve into a safer and more efficient integration of UAS into the airspace. The systems to cope with the regulations should also be modular and flexible in order to not be superseded by innovations in the area. Thus, the aviation regulatory bodies aim to achieve designs with flexibility where possible, and structure where needed. Having flexible hardware and software increases modularity, which is for the most part supported via open-source systems.

2.2.5 Congestion Management

According to *UAV Factory*, one of the large European UAS companies, "The future of the UAV industry is likely to be shaped by airspace congestion" [79]. Indeed, high level airspaces are becoming crowded, and large-scale solutions, such as NextGen (US) or SESAR-JU (EU), are necessary to increase airspace capacity while maintaining the current safety levels. However, there is no such existing management solution for Very Low Level (VLL). Yet, large projects like Amazon's Prime Air and Google's Project Wing are already ready to populate the VLL airspace.

Part of the congestion management problem is to avoid conflicts (and more importantly, collisions), between the UAS through strategic deconfliction and safety nets. Another purpose of the congestion management system is to make sure that UAS do not go where they are not supposed to go, thus requiring geofencing. In order to implement the previously mentioned systems, the UAS autopilot needs to be able to perform complex operations, for example, static waypoints following is likely to be insufficient. In the following, we divide these issues into four topics of interest: 4-D trajectory management; geofencing; safety nets; and complex operations, and show how the *Paparazzi* system addresses them.

4-D Trajectory Management

As noted in [37], 4-D trajectories will be central in future airspace management methods. The principle of 4-D trajectory management is to have every UAS broadcasting

its trajectory up to some time horizon, and to receive Unmanned Aircraft System Traffic Management (UTM) clearances in the form of trajectories. The trajectory information contains a path, a series of points through which the UAS will pass, and will at times be associated with each of these points. Thanks to this information, the idea is to perform pro-active deconfliction, as explained by Thomas et al. in [92]. It implies that UTM detects future conflicts along the trajectories of all UAS and deconflicts them as safely and as quickly as possible. This kind of approach requires the autopilot to represent UAS motions with trajectories and to be able to transmit them, which is the case in *Paparazzi*.

Paparazzi originally supports a basic description of trajectories based on circles and straight lines, but recent updates allow it to process advanced trajectories as well. Indeed, *Paparazzi* can represent trajectories as functions in 2-D (x, y), 3-D (x, y, z) or 4-D (x, y, z, t). The only requirement is for the function to be differentiable at least two times at every point. The function and its closed form derivatives are computed offline and can then be quickly evaluated online. Moreover, gains can be calculated by tuning to adjust the convergence speed from the UAS starting point to the given trajectory. These gains influence the convergence path to a given trajectory and the resulting trajectory (convergence + commanded) can be computed offline, given initial conditions and the commanded trajectory. This is of particular interest for UTM as it allows the UAS to provide not only its trajectory over time but also how it will reach this trajectory from its starting point.

It is important to note that UTM will have to solve conflicts between manned and unmanned aircraft. Thus, an air traffic controller is needed as part of the process with an appropriate display for 4-D trajectories and a communication link with both manned and unmanned aircraft.

Safety Nets

Trajectory deconfliction is the first step to manage congestion, however safety nets are also part of congestion management. Indeed, safety nets such as self-separation and collision avoidance allow UAS to fly close to each other while preserving an

acceptable safety level. Though *Paparazzi* does not include self-separation algorithms, it does contain a light version of the TCAS II collision avoidance system. It considers intruders one at a time and is capable of coordinated avoidance maneuvers.

Values such as distance thresholds have been adapted to suit the performance of small UAS. Keeping in mind that the *Parapazzi* philosophy is to be configurable, so these parameters can be easily changed from a configuration file.

A module has been recently developed to input data from traffic services and display them into *Paparazzi*'s ground control station, thus providing situational awareness to the remote pilot. Preliminary tests have been conducted using opensky-network to display traffic around a given area. Based on this information, the remote pilot can effectively perform conflict resolution.

Geofencing

Keeping UAS away from each other is important, but keeping them out of forbidden areas is critical. Geofencing determines no-fly zones where the UAS should not enter. To accommodate land owners while managing traffic and limiting congestion, Foina et al. [39] have proposed a participative dynamical airspace management method: the air-parcel model. This allows land owners to authorize/forbid flights over their land through a web interface. However, this type of approach requires the UAS to be able to handle dynamical geofencing. Furthermore, this model considers only cuboid parcels; the need for more precise airspace shapes may emerge making 3-D geofencing a fundamental need.

This type of model can be handled by *Paparazzi* by defining geofenced zones manually or as a function. Zones are defined as a simple geometrical shapes (e.g., a circle, a polygon, etc.), plus altitude limits, thus effectively creating 3-D geofencing. These zones can be static, activated dynamically from the ground station or from the flight plan with associated conditions.

Complex Operations

Having 4-D trajectory management, safety nets and geofencing are useless if the UAS cannot follow instructions from the UTM regarding these tools. Indeed, new UTM paradigms suggest being able to change flight plans dynamically to answer to UTM demands. In [98] two types of complex operation examples are mentioned: space transition corridors; and temporary flight restrictions. Both methods require that the UAS can modify its flight plan according to the new UTM instructions.

Modifying the flight plan dynamically is not possible, and not desirable, in *Paparazzi*. However, an appropriate response to these complex operations can be provided through conditional flight plans. This enables the UAS to follow different courses of action depending on given parameters. These parameters are defined offline by the user and can then be modified online by values coming from sensors or from UTM. This allows an intelligent reaction to exterior instructions; in particular, it can be used by the user to give UTM control on portions of the flight.

2.2.6 Reliability

The improvement of the reliability of the flight is considered to be one of the main goals for integrating military UAVs into civil airspace according to the unmanned systems roadmap of US Office of the Secretary of Defense, DoD [82]. Compared to manned counterparts, civil UAVs experienced more failures with a frequency of two orders of magnitude compared with military UAVs. Although this has changed in the last few years with technological improvements, making the UAVs as reliable as early manned military aircraft does not appear to be sufficient from the DoD perspective. This can be realized by checking the biggest portion of control technologies' budget for research and development, which is health management and adaptive control.

To achieve a safe flight is not an easy task considering the unknowns of the systems hardware, environment and possible system faults and failures yet to emerge. Also, increasing demand on cost-effective systems, resulting in smaller sensors and actuators with less accuracy, impose the need for the software to achieve even more. The

expectation that UAVs should be less expensive than their manned counterparts might have a hit on reliability of the systems.

Small and Medium Enterprise (SME) and Certification Costs

Using drones for quicker and cheaper deliveries could be rewarding for SMEs. Being an early bird might put the SMEs in an advantageous position, considering the increase in the capabilities of the drones with ongoing research activities and their widened application areas. Nevertheless, inexpensive access to drones and their relatively cheap utilization cost does not seem to be enough to put them to air at present, due to the heavy cost of certification and regulatory hurdles [81]. In this regard, capable open-source solutions may be an effective way to loosen this restriction; otherwise, SMEs may not survive. Even worse, they may operate them without relevant the permissions, sacrifying a substantial fine as reported by Civil Aviation Authority (CAA). This will compromise confidence in the system contradicting the hopes for reliable the integration of drones into the airspace. As an open-source autopilot system, *Paparazzi* is a known platform for a computer scientist who wants to test the viability of complex software systems. Thus, parts of *Paparazzi* code (Ground Control Station (GCS)) have been formally proven [94] and a stable version has thousands of flight heritage. Furthermore, although *Paparazzi* is not certified, to be able to have access to the certified airspace, users can build up their configuration on the readily and freely available *Paparazzi* system, which may reduce the cost of having a certified system reaching specific airspaces.

Individuals and Education

Individuals, as well as SMEs, suffer from the same budget constraints. Personal UAV usage counts for a substantial amount of the drone ecosystem. Both US and European authorities mention the importance of individuals in the utilizations of drones. There is a community with a passion for aviation and its potential, thought it may likely be inexperienced. *Paparazzi* offers a whole community to help and educate these beginners through its forums enhanced by a rising number of users.

A rich documentation is available through the Wikipedia page, encouraging users to self-teach.

2.2.7 Flight Heritage for Risk Assessment

As the drone industry is extremely innovative, technical developments may supersede the prescriptive rules as regulations. Thus, a solution may be to follow a risk-based approach rather than to have strict rules to comply with. Predicted regulations in Europe seem to evolve under different categories dedicated to specific operation risks. Flight heritage and occurrence reporting is expected to be an inevitable part of safety risk assessment in achieving reliable flight. The wide utilization of *Paparazzi* and real-time connection to the ethernet could offer reliable and practical solutions to be able report occurrences.

Support for real-time planning and onboard vehicle automation

To access low-altitude airspace with the use of small unmanned aircraft safely, an important ability could be to implement real-time planning and on-board vehicle automation. Amazon states that this approach will allow some flexibility to adapt to different situations, such as weather changes, severe winds or any other emergency needs. *Paparazzi*, has a real-time planning ability already implemented, allowing the user to change the trajectory in real-time through its ground control station via different strategies. The user could switch between trajectories already available or even drag the waypoints to his/her preference. The automation of the vehicles is handled in different stages. For now, the most used modes of autonomy is no autonomy, assisted mode and fully autonomous mode. No autonomy mode, or manual mode, gives whole responsibility of the aircraft dynamics to the pilot through the remote control link. The assisted mode closes some attitude control loops, giving some stability to the aircraft. This will ease the challenges of piloting which could be a significant advantage, especially for unexperienced pilots. The fully autonomous mode handles both heading and navigation through selected trajectories. In case of

an emergency, the pilot takes control of the aircraft at any time, through the RC link. Features already implemented on *Paparazzi*, such as geofencing, go home, and its ease in adding and smodifying thresholds to various variables, support the safety procedures.

2.2.8 Future Evolutions

As is the same for many open-source projects, *Paparazzi* is in constant evolution through regular contributions. With its large community, it is difficult to keep track of all ongoing projects. In this section, we present three features being currently explored and developed at ENAC's labs.

Though there is currently no life at stake, avoiding UAS collisions is desirable for safety and operational reasons. *Paparazzi*'s current TCAS-like collision avoidance allows the operation of numerous UAVs with little risk of collision between them. However, future missions will include more and more UAVs in bounded size airspaces. In this context, an efficient collision avoidance algorithm is desirable so as to allow closer operations with an acceptable number of nuisance alerts. Because it relies on advanced logics and can be adapted to different types of performances, we have chosen to implement the ACAS Xu algorithm. Though this standard's definition has just started, the baseline is already determined and may allow the development ofs a simplified version for micro UAVs.

Regarding geofencing, most current methods use boundaries in the horizontal plane into which the UAV cannot enter. However, the utilization of the VLL airspace is likely to demand more sophisticated methods to handle 3-D geofencing where a 3-D no-fly shape can be described. Though *Paparazzi* can emulate 3-D geofencing, it cannot yet use full 3-D models. Work is underway to extend the current geofencing to allow the loading of terrain models, and padding them with arbitrarily shaped no-fly zones.

Finally, conventional procedures to handle failures onboard could be one of the three approaches in order to achieve safe flight. The first is the fail operational systems, which are made insensitive to any single component failure. The second

approach is the failsafe systems, where a controlled shut down to a safe state is practiced whenever a critical fault is pointed out by a sensor. The third approach is the fault tolerant control systems, in which components of the system are monitored and actions taken in whenever needed. The key strategy is most likely to keep the system availability and accept reduced performance. For the *Paparazzi* autopilot system, there is ongoing research to implement the ingenious fault detection and diagnosis module to enable reconfigurable controller loops.

2.3 Fault Tolerant Control Systems

The improvement of the reliability of the flight is considered to be one of the main obstacles to integrating UAVs into civil airspace. To achieve a safe flight is not an easy task considering the unknowns of the systems, environment and possible system faults/failures to emerge.

2.3.1 Loss of Control in Aviation

One of the biggest contributors to fatal accidents is considered to be aircraft *loss of control* (LOC). The Joint Safety Analysis Team (JSAT) defines LOC as a "significant, unintended departure of the aircraft from controlled flight, the operational flight envelope, or usual flight attitudes, including ground even". Here, *significant* refers to events resulting in an incidence or accident [87]. Control failures, inappropriate pilot action (inaction) in a healthy aircraft, and vehicle impairment are examples of LOC events [83].

LOC-I (loss of control in-flight) is defined as "an extreme manifestation of a deviation from intended flight path" and is the most deadly accident type, with 37 fatal accidents per year (for fixed-wings) [35]. Although LOC-I is the cause behind many fatal accidents, manned aviation has a very limited use of LOC prevention and recovery [10]. Having no single action to prevent LOC events, technical limits in LOC simulations, such as full stall or failure simulations, constitute some of the challenges to LOC event prevention and recovery solutions.

To offer prevention strategies, the causes of LOC accidents and incidents should be thoroughly studied. LOC events have been categorized by Ref.[57] under five main topics, two of them being the most common. Aerodynamic stall and flight control system are the biggest factors that led to LOC accidents and upsets in 15 years log of LOC accidents in transport airplanes. Autopilot commands and control surface failures are considered under the flight control failures category, the second most fatal LOC of all types [57]. The same study also shows that the percentage of flight control system upset incidents have risen from 9% to 22% from 1993 (pre-1993 compared to 1993-2007). This could be caused by the increased complexity of onboard systems suppressing the technological advancements in fault detection and recovery. Thus, addressing onboard fault detection and recovery could contribute to the reduction likelihood of LOC accidents and/or fatalities. The importance of these emergency recovery measures are magnified for unmanned systems, considering the increase in the number of drones, and the difficulties inherent in unmanned systems.

Unmanned systems are more susceptible to LOC events; they are less robust to disturbances, and recovery must be held by the onboard computer or a remote pilot [83]. Studies for drone regulations increase the pace for the assessment of risk for drone operations. A recently published *Annual Safety Review 2017* [34] discusses aviation accidents in detail, with a chapter focusing on drones. This report by EASA involves data from European Central Repository (ECR), experienced by EASA member states. With an increase in the number of drones, and possibly raising consciousness on reporting occurrences, the numbers of non-fatal accidents increased by 470% in 2016 relative to the 2011-2015 average, fortunanetly maintaining zero fatalities. Most of the time, it is the commercial airliner pilot who reports the occurrences, and rarely the UAS pilot. The prior key risk areas have been investigated and aircraft upsets are, by far, the most common cause of occurrence, and are established as the first key risk to address for the safe integration of drones into the airspace. 50% of RPAS accidents fall in this case which often results in the damage or destruction of UAS, following a loss of control of the drone by the pilot. The second key risk area is airborne collision although it is rarely encountered at present. The risk is expected

to increase due to the probable increase in the number of drones. Obstacle collision is the third risk area which is likely to increase with the integration of drones, especially in urban areas.

Designing recovery measures for unmanned systems has further challenges due to the lack of redundancies utilized in manned aviation and the use of cheaper and less accurate components. Introducing intelligent algorithms to reduce the risk of harm is usually adopted for those systems which are limited in hardware. Fault tolerant control systems (FTCS) are designed to provide solutions to systems which are under fault/failure. There are a wide range of different strategies offered for this solution such as passive or active FTCS, where the latter requires a fault detection and diagnosis (FDD) phase [30]. FDD methods are used to discover if there is any fault/failure in the system and to determine the type of the fault.

After the fault is known, an assessment of the current abilities of the drone is vital. The severity of the situation must be evaluated to decide if a recovery is possible. If so, the current situation of the drone should be considered during the design of recovery control methodologies. In the case a recovery is likely to fail, a safe ditch maneuver can abruptly decrease the number of fatalities. Maps pointing to zones with no (or minimum) population could be uploaded onboard and the safest region to ditch can be selected. Since these situations are handled by the pilot for manned aviation, the number of works to assess and plan ditch maneuvers is few for unmanned systems. NASA is offering *Safe2Ditch* [2] to offer autonomous crash management but it is only currently at the design stage.

2.3.2 FTCS Terminology

Systems are often sensitive to faults of differing nature. Existing irregularities in sensors, actuators, or controller may be amplified due to the control system design and lead to failures (Fig. 2-10). A fault may be hidden thanks to the control action [30].

Since fault tolerant control is comprised of a set of different disciplines and is a relatively new topic, the terminology is not firmly established. “FDI” could be a

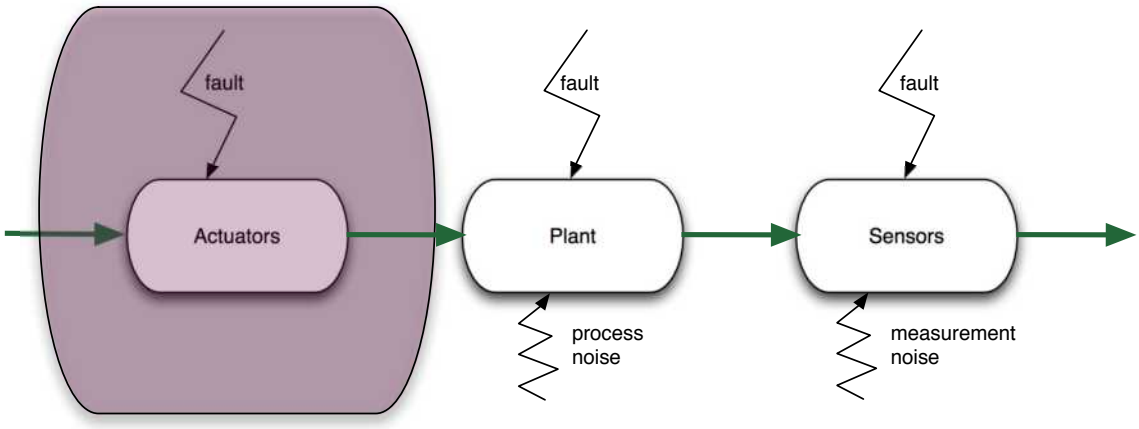

Figure 2-10: Faults altering the system

proper example of this ambiguity. In some works, it stands for Fault Detection and Isolation while in others it stands for Fault Detection and Identification (which could also be named after Fault Detection and Diagnosis), meaning that identification is added to Fault Detection and Isolation [102].

One of the first attempts to unify the terminology was carried out by the IFAC SAFEPROCESS technical committee in 1996 and published by [54]. Fault, failure and the methodology to handle them, such as fault detection, fault isolation, fault identification, fault diagnosis, and supervision terms are explained separately to avoid the ongoing ambiguity in this field. Although fault detection methods are clearer, the difference between the methods for the two steps of fault diagnosis (namely the fault isolation and fault identification) is not very obvious.

2.3.3 Conventions for a Safe Flight

The most widely-used method to increase reliability is to use more reliable components and/or hardware redundancy. Both require an increase in the cost of the UAS [5]. To offer solutions for all different foreseen categories of airspace, a variety of approaches should be considered. While hardware redundancy may cope with the failure situations of UAVs in the certified airspace, it may not be suitable for UAVs in open air or some subsets of specific categories due to budget constraints. Analytical

redundancy is another solution, but it may be not as effective and simple as hardware redundancy, and it relies on the design of intelligent methods to utilize all information on board an aircraft to wisely deal with all eventualities.

There are three approaches to achieve safe FTC in standard flight conventions. The first is the fail operational systems, which are made insensitive to any single point component failure. The second approach is the fail safe systems, where a controlled shut down to a safe state is practiced whenever a critical fault is noted by a sensor. The level of degradation ensures the switch to robust (alternate) or direct (minimal level of stability augmentation independent of the nature of the fault) mode. Switching from nominal mode to the robust and direct modes leads to a decrease in the available GNC functions. This causes a degradation in ease of piloting. Furthermore, some optimality conditions may have been compromised. The third approach is the fault tolerant control systems, in which redundancy in the plant and the automation system are used to design software that monitors the components and takes in actions whenever needed. This strategy is the most probable to try to keep plant availability and to accept reduced performance [12].

RECONFIGURE project of FP7 [42] aims to address the problem of piloting degradation and optimality compromise by Flight Parameter Estimation (FPE), which is the online estimation of aircraft parameters, FDD and FTC in case of off-nominal events [80]. They utilize a black box nonlinear model of aircraft, and the project uses some outputs of a previous FP 7 project ADDSAFE leaded by Deimos Space [1].

2.3.4 Methods for Fault Tolerant Control Systems

Among the different categorizations of fault tolerance, there are options to handle faults on-line or off-line. Employing fault diagnosis schemes on-line is a way to achieve fault tolerance. In this case, as soon as a fault is detected, a supervisory agent is informed via a discrete event signal. Then, accommodation of the faults is handled with either the selection of a predetermined controller for the specific fault case, or by designing the action online with real-time analysis and optimization [12].

Another common categorization of FTCS is passive and active FTCS. In passive

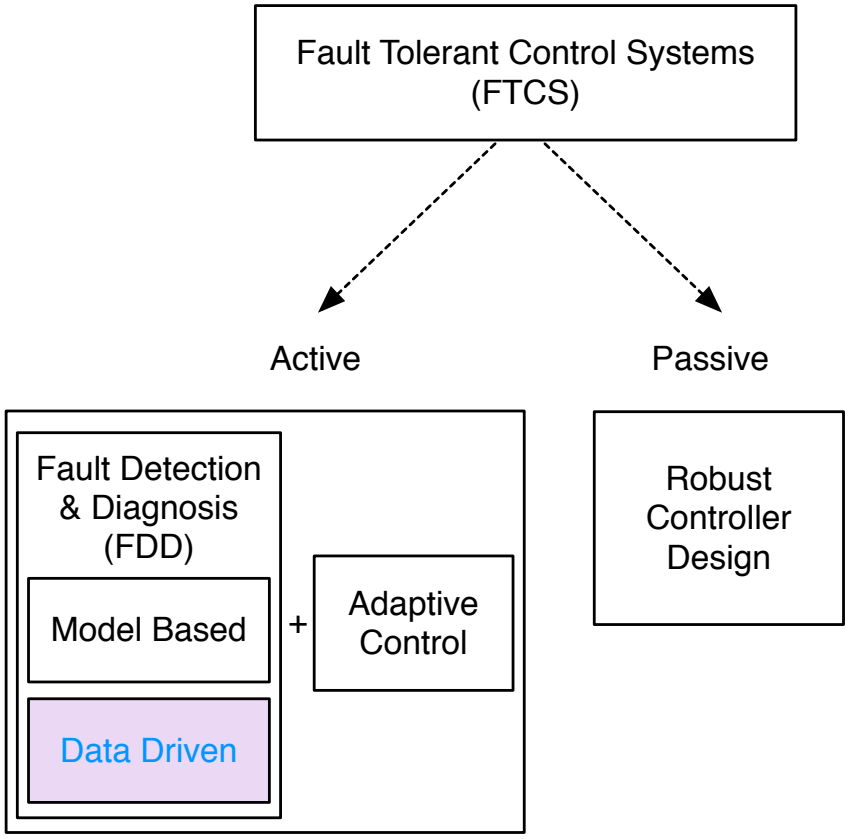

Figure 2-11: Variations of fault tolerant control systems

FTCS, the flight controller is designed in such a way to accommodate not only the disturbances but also the faults. Active FTCS first distinguishes the fault via the fault detection and diagnosis module, and then switches between the designed controllers specific to the fault case or designs a new one online [5]. While active FTCS requires more tools to handle faults, as seen in Fig. 2-11, for faults not predicted and not counted for during the design of the robust controller, this method most probably fails.

Even with a long list of available methods, the aerospace industry has not implemented FTC widely, except for some space systems, because of the evolving nature of the methods, the nonlinear nature of the problem, design complexity and the high possibility of wrong alarms in case of large disturbances and/or modeling uncertainties. Thus, the hardware redundancy is now the preferred way because of its ease and

maturity being implemented on various critical missions with human lives.

2.3.5 Fault Detection and Diagnosis

FDD is handled in two main steps; fault detection and fault diagnosis. Fault diagnosis encapsulates fault isolation and fault identification. FDD should not only be sensitive to faults but must also be robust to the model uncertainties and external disturbances.

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 relies on observational data, and it prefers dense, redundant data with a frequency larger than the failure rate.

Model-Based Methods

In model based approaches, relationships between measurements and estimated states are exploited to detect possible dysfunctions. The most common ways to implement a model-based approach is to estimate the states, the model parameters. The accuracy of the results depends on the type of faults (additive or multiplicative). Additive faults affect the variables of the process by a summation whereas the multiplicative faults affect the variables by a multiplication. When only the output signal can be measured, signal model-based methods can be used for fault detection, such as Bandpass filters, Spectral analysis(FFT) and maximum entropy estimation. For the case where both the input and output signals are available, the methods used for fault detection are called process based methods such as: state and output observers(estimators), parity equations, and identification and parameter estimation. They generate residuals for state variables or output variables. The most widely-used technique for sensor and actuator faults are the state and output observers (estimators) and for process faults, identification and parameter estimation [54].

The output of the model-based fault detection methods is the stochastic behavior with mean values and variances. With the use of change detection methods,

deviations from the normal behavior can be detected. For that purpose, three available methods are considered: mean and variance estimation, likelihood-ratio-test and Bayes decision, run-sum test and two-probe t-test. Fault detection is only supported by simple threshold logic or hypothesis testing in most of the applications [54].

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 [46]. Parameter identification via H_∞ filter is used to indicate icing in [64]. Another method to detect and isolate actuator/sensor faults is the multiple model adaptive estimation (MMAE) method [30]. In MMAE [61], multiple Kalman filters are used as shown in Fig. 2-12. In this method, each Kalman filter k uses a different system model to calculate the state estimates $\hat{x}_k$. This multiple Kalman Filters can be run in parallel, since each filter k is not dependent on other Kalman Filters. The difference between the predicted measurements (calculated using the estimated states) and the real measurements z, gives the residual r_k. Residuals are used in the hypothesis conditional probability computation since they represent each model's closeness to the real model. Finally, the conditional probability p_k for each model k is calculated. Then, based on the conditional probabilities, state estimates are blended through a probability weighted average, that results in the final state estimate $\hat{x}$ [65].

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.

A fairly old study from 1984 investigates FDI systems robust to uncertainties. One of the two steps of FDI (residual generation and decision-making) is targeted. They offer the ability to handle model uncertainties, by designing a robust residual generation process [19]. Another study deals with model uncertainties by determining the threshold of the residual in a novel way with an application to detect aileron

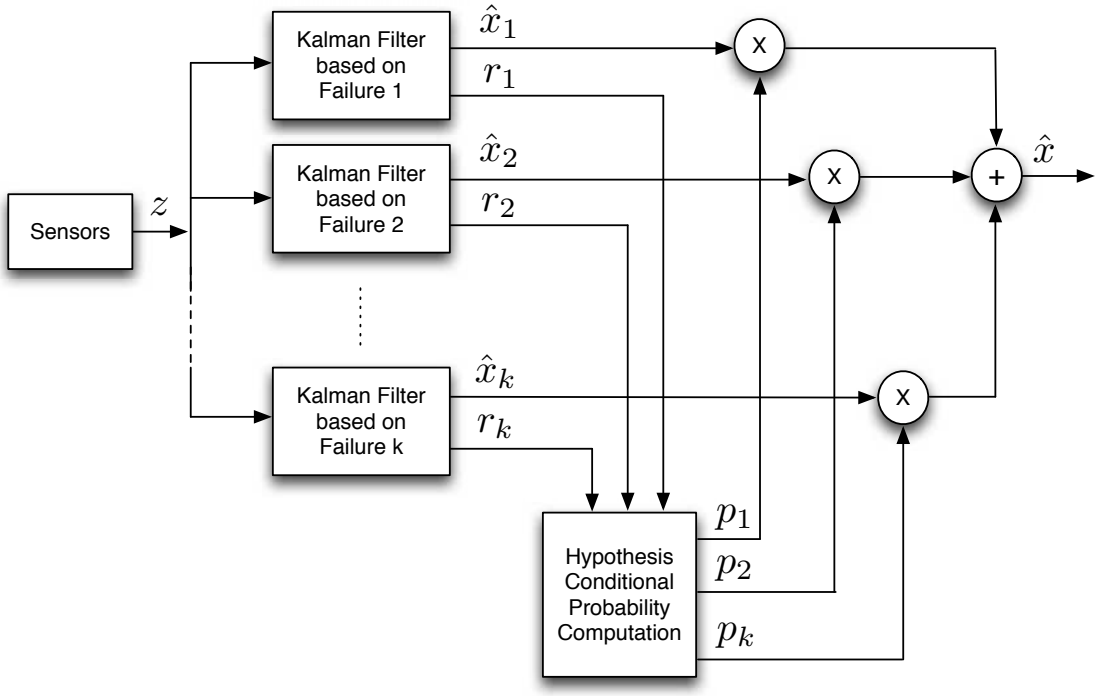

Figure 2-12: Multiple model adaptive estimation algorithm [65]

actuator fault [85]. [88] uses two cascade sliding mode observers state estimation and fault detection to guarantee staying in sliding manifold in the presence of unknown disturbances and faults.

Data-Driven Methods

Model-based approaches have various successful applications up until now; most of them assuming an accurate model is available on-board. With the new era of UAVs, the airspace is expected to be populated by a strong increase in the number of UAVs. The variety of UAVs, the expense of accurate modeling practices, the difficulty in modeling the behavior of UAV in case of failures,all call for alternative approaches for the quite challenging problem of FDD. The increased efficiency of on-board sensors, 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, 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. 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 the further integration of this fault into the diagnosis scheme [91].

[43] argues artificial intelligent methods for the fault detection of complex systems. Comparison between Principle Component Analysis (PCA) and model based stochastic parity space approaches is given in [45]. In [60], the authors argue the use of dynamic PCA since UAV flight controls is a dynamic system itself and the Dynamic Principle Component Analysis (DPCA) can reflect unknown disturbances, while model-based approaches can only model typical disturbance.

2.4 Machines on the Rise

Since machine learning methods are selected to detect and diagnose actuator faults in this thesis, a short introduction to the topic is now provided.

The mathematical analysis of learning processes goes back to the1960s when F. Rosenblatt [84] suggested the *perceptron*: the first learning machine [96] (see Fig. 2-13). Inspired by a simplified model of a biological neuron, the idea of the perceptron was discussed for many years in the neurophysiological literature [96]. The contributions of Rosenblatt were to describe the perceptron model as a program for computers, and to show that the model could be generalized via experiments [96]. The perceptron was first implemented on an IBM 704 as an algorithm. Later, a custom-built hardware implementation of the perceptron was delivered (as shown in Fig. 2-14).

The interest in perceptron came to a halt when Minsky and Papert proved that a single layer perceptron cannot learn an XOR function [66]. This led to a wrong assumption that multi-layer networks would not be able produce an XOR function.

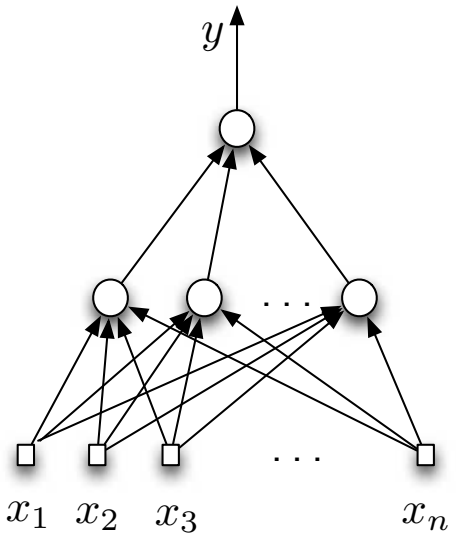

Figure 2-13: Schematics of perceptron. The perceptron is a composition of several neurons [96]. x_1 to x_n are the features and y is the output.

Although Grossberg published that multi-layer perceptrons are capable of producing XOR functions, the declined interest has persisted for some years until the research on perceptron was revived.

Back-propagation method, proposed by several authors [58, 86] independently in 1986, is considered to be the second birth of the perceptron. It is originally an older method (1963) [17] and was used in solving control problems [96]. The application of the method to calculate the weight simultaneously for many neurons changed the developmental path of learning machines [96]. In 1980s, the terminology was also changed due to an application-oriented approach in research on learning (with the introduction of powerful computers) [96], and multi-layer perceptrons were renamed as neural networks.

Although 1974-1980 was not very fruitful for neural networks, and was known as the AI winter, the development of statistical learning theory endured. A novel inductive principle, *Structural Risk Minimization (SRM) inductive principle*, was introduced in 1974 [97]. In the SRM inductive principle, the risk in a given set of functions is minimized by controlling two factors: the value of the empirical risk; and the value of the confidence interval [96].

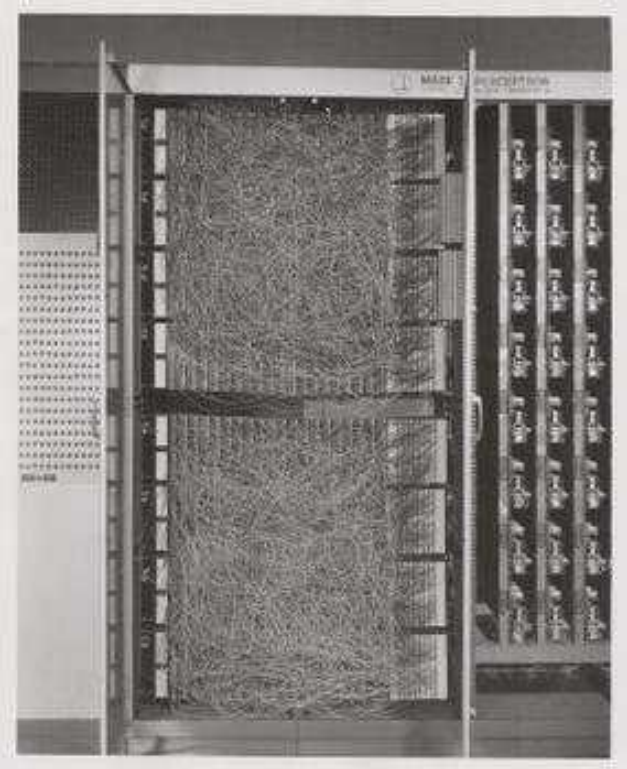

Figure 2-14: Mark 1 perceptron: the first custom-built hardware implementation of the *perceptron*. It is designed for image recognition with an array of 400 photocells connected to the "neurons" randomly. Potentiometers are used to encode the weights, and electric motors update the weights during learning [11].

Support Vector Machines [97, 95] are introduced as an application of SRM by keeping the value of the empirical risk fixed (say equal to zero), and minimizing the confidence interval [96], as opposed to keeping the confidence interval fixed (by choosing an appropriate construction of machine) and minimizing the empirical risk (as in Neural Networks). Their applicability expanded thanks to its extension to problems that require nonlinear classifiers [13] using *kernel trick*, [3] and also to problems with non-separable classes [25] using *soft margin separating hyperplanes*.

Almost two decades after IBM's Deep Blue beat the chess champion Kasparov [47], Google DeepMind's Artificial Intelligence(AI) AlphaGo [26, 18] defeating 9-dan Go professional player raises the question: what will the next victory of AI look like? European Parliament published a draft report with recommendations to the European Commission on Civil Law Rules on Robotics [28], including a list of concerns for a possible rise of the learning machines. Not only the singularity (artificial super intelligence resulting in deep changes in human civilization) but also more inevitable outcomes are discussed, such as AI's effects on the workforce, ethical and legal issues inherent to automatized systems including drones.

The winner of the Go contest, the AI, is based on a method named *Reinforcement Learning* (RL). This method is used in many of the latest ground-breaking applications of AI. In reinforcement learning, the selection of actions results in consequences. Thus, the agent interacts with its environment, by taking actions and observing the consequences via the reward function. The goal of the agent is to calculate actions that will maximize the reward. An example of this functioning can be demonstrated on a robot locomotion problem. Fig. 2-15 shows a humanoid model and an ant robot model from *Google DeepMind* [27, 72, 53]. The robot models have virtual sensors which give information about their states (angle and position of the joints [72]) and surrounding objects [53]. The objective for the robots is to move from point A to point B. Although the robots were not explicitly programmed to walk or jump, and were not given any information about what walking or jumping looks like, they learnt to move in a way very similar to walking and jumping.

In recent times, we have witness very promising application results in Neural

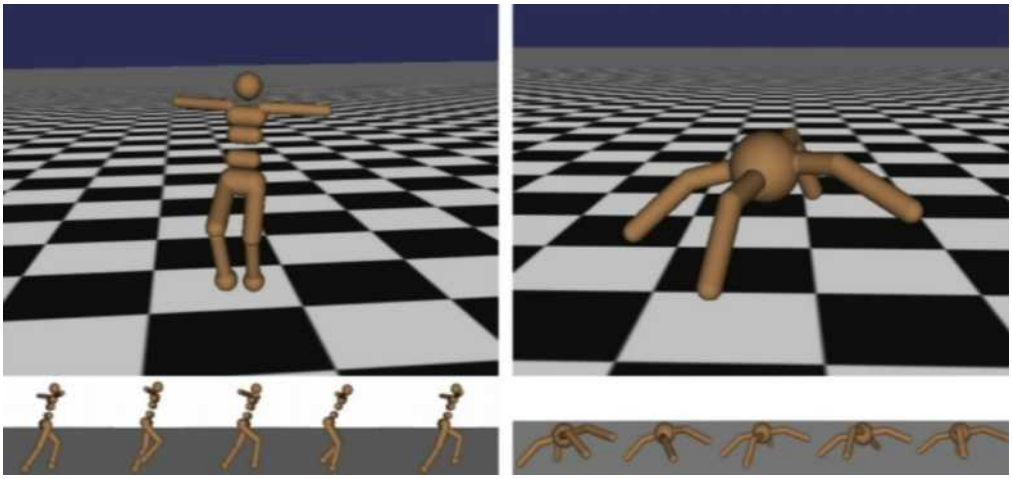

Figure 2-15: Humanoid and ant robot models [72] exhibiting walking and jumping motions respectively using reinforcement learning [27]. The robots are given objectives to go from one point to the other, but not given any information about walking or jumping [53].

Networks in its various forms. Convolutional neural networks (CNN) applied to image processing problems, and sequence models (Recurrent Neural Networks (RNN and Long short term memory (LSTM)) applied to natural language processing have produced remarkable results, raising interest in machine learning and AI in various fields. Such an example is Google's *Duplex* AI [59], making phone calls to restaurants or hair salons, to book a table or arrange a meeting. AI talks to the person on the phone and adapts to the responses of the human on the phone, and achieves its task without being exposed that it is actually an AI.

An application of RL in aviation is demonstrated with Stanford University's autonomous helicopter [70]. Thanks to RL, the helicopter's autopilot performs acrobatic maneuvers, which is difficult to do with a hardcoded controller design. A photo taken during its inverted flight can be seen in Fig. 2-16.

Machine learning has already started to take part in aviation. Operational improvements in aviation is one of the preliminary fields in which it has appeared. Recently, in an AGE sponsored competition, data scientists designed a routing algorithm using real flight data, and achieved a 12% improvement in fuel consumption efficiency. Some other operational problems issued by machine learning are accurate arrival time

Figure 2-16: Autonomous helicopter from Stanford University learns to fly acrobatic maneuvers using RL in autopilot [70]

estimation and calculating optimal parameters for take off, mostly with the aim of reducing airline costs and increasing customer satisfaction. Another application is to deal with the expected shortage of pilots, which may be inevitable in years to come, thanks to the increasing demand for air travel. Therefore, a prospective utilization of machine learning is to assess the competency of pilots with unsupervised learning models in order to reduce the long duration of pilot training, hence contributing to reducing pilot shortage. The use of data-driven approaches for pilot training could be used not only for the assessment of the pilots but also to increase the efficiency of training by offering trainee-focused personal training, and adapting the curriculum to their individual skills and needs. To overcome the pilot shortage, another way to handle the problem may be to reduce the need for one.

AI has actually already made its way into the cockpit. Garmin implemented voice recognition features garnished with some other functions, such as changing radio channels, to assist the pilots with its audio panels GMA 350 and GMA 35, called *Telligence*. One of the goals of the autonomy researchers is to look for ways in which to implement artificial intelligence to one of the core features of an aircraft: the autopilot. Hence, the ability to use them remains challenging due to the inherent nature of the methods, that is, non-determinism. EASA A-NPAs offer different categories for

drones operations, depending on the risk of the operation of concern. The difficulties waiting for the machine learning research are not currently well defined or widely argued in aviation.

2.5 Conclusion

In this chapter, we discuss the current status of drone regulation, an introduction to *Paparazzi Autopilot* which is used in this thesis and the advantages of open-source systems to aid drone integration into the airspace. Then, fault tolerant control systems are introduced with a focus on fault detection and diagnosis. Two main methods, model-based and data-driven, are discussed via reference to the literature. Then, general advancements in artificial intelligence (AI) are addressed, since one of its techniques — machine learning — is used as the methodology to detect the faults in this thesis.

In Section 3, nonlinear aircraft dynamics will be discussed in order to provide insights about flight motion, and to simulate the states of an aircraft. These states can then be used to simulate sensor measurements on board a drone, and are necessary to design data-driven fault detection and diagnosis algorithms.

Chapter 3

Nonlinear Aircraft Model

The movement of any object can be represented by changes in its location (translation) or changes in its attitude (rotation) or a combination of both. The motion of an aircraft usually involves both translation and rotation. Studying aircraft motion is complicated as these two motions are coupled; for example a rotation might cause a change in aerodynamic forces which then affects the translation. Thus, to ease the process, the motion is broken into easier problems utilizing some assumptions. Such an assumption for the aircraft motion is to assume that the aircraft is a point mass; all of its mass is collected at its center of gravity, so it translates from one point to the other. Then, the aircraft's rotation is investigated by no longer assuming it as a point mass but a rigid body in space.

In this chapter, modeling both of these motions (translation and rotation motion) is presented in detail. Equations of rotation and translation motion are driven for generic aircraft. Then, calculation of forces and moments, which are required to solve those equations, are given for two types of drones (the aerodynamic force derivatives and stability derivatives are specific for each drone). These two examples of drones are by ETH Zurich and MAKO (used in ENAC UAV LAB).

In this thesis, drone motion is simulated to generate data. The accelerometer and gyro data are generated for MAKO model, considering the specifications of IMU *InvenSense MPU-9250 Nine-axis* used in *Paparazzi Autopilot Apogee* onboard. The models derived here are not used in detection and diagnosis algorithms. The detection

and diagnosis algorithms implemented in this thesis only use data.

If the reader is not interested in detection and diagnosis via simulated data but interested in detection and diagnosis via real data, it is not essential to read this chapter as it explains the models that are used to simulate measurements. Nonetheless, having background information on the physics of the system may help to understand the features (translational acceleration and angular velocities) used in both model-based and data-driven fault diagnosis.

3.1 Attitude motion modeling

Rotation is a change in an object's attitude. A change in attitude is modeled using rotations about the center of gravity. This section derives the equations for rotation (attitude motion) in detail. Note that the equations for attitude kinematics and attitude dynamics can be found in Equations 3.35 and 3.51, respectively.

Rotations are directly affected by external torques and moments while translations are directly affected by external forces. The attitude of an aircraft during translation also affects the aerodynamic forces causing changes in translation.

A force applied at a distance from the center of mass causes a rotation. A very common approach in aircraft control is to balance those rotations, by trimming the aircraft, such that the aircraft will not rotate.

In this section, attitude motion is represented using kinematic and dynamics equations. First different parametrization of attitude, such as Euler angles and quaternions, are discussed. Then, kinematic and dynamic equations of attitude motion are derived for a general rigid body. These equations are later specified for the aircraft as the rigid body of interest.

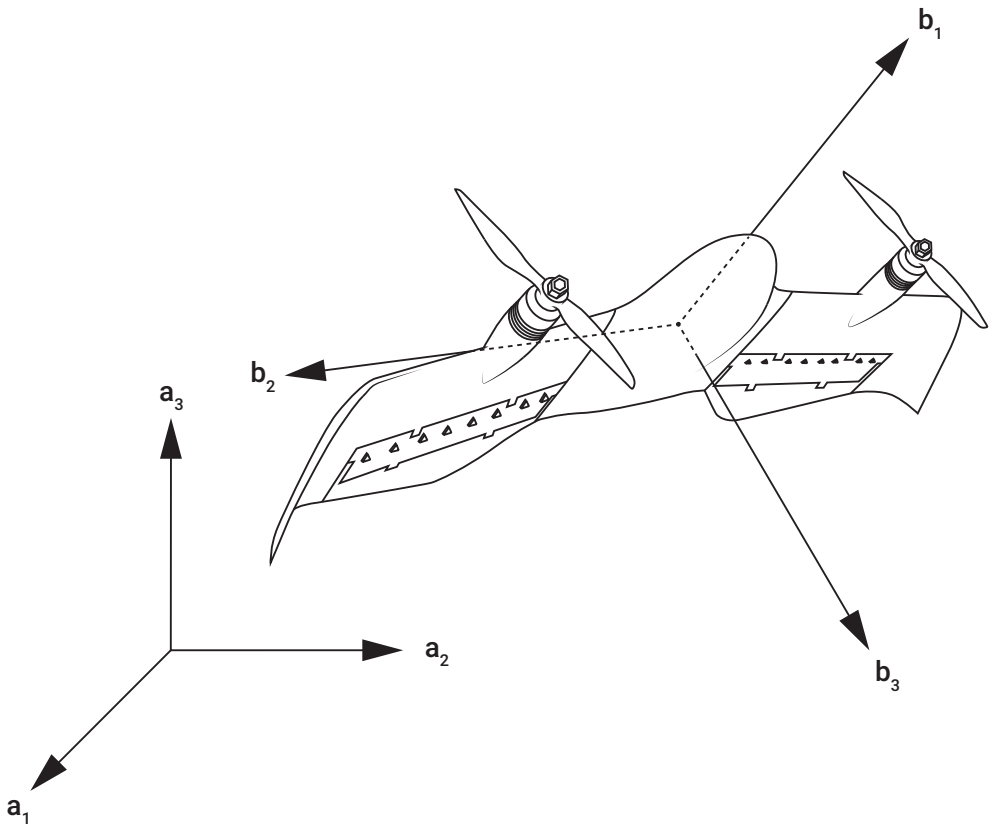

Figure 3-1: Attitude representation is simply specifying the orientation of aircraft body axes $\boldsymbol{b_1}, \boldsymbol{b_2}, \boldsymbol{b_3}$ in the reference frame A

3.1.1 Attitude representations

Let $\boldsymbol{b_1}, \boldsymbol{b_2}, \boldsymbol{b_3}$ be a triplet of unit vectors, representing an orthogonal coordinate system attached to a rigid body such that:

$$\boldsymbol{b_1} \times \boldsymbol{b_2} = \boldsymbol{b_3} \tag{3.1}$$

The rigid body is composed of points, which do not experience any distance change between themselves during the motion of the body. The problem in representing attitude can simply be thought of specifying the orientation of this triplet with respect to some reference frame A, such as in Fig. 3-1.

Expressing the basis vectors b_1, b_2, b_3 of B in terms of basis vectors a_1, a_2, a_3 of A is given by :

$$\begin{aligned}
\boldsymbol{b_1} &= C_{11}\boldsymbol{a_1} + C_{12}\boldsymbol{a_2} + C_{13}\boldsymbol{a_3}, \\
\boldsymbol{b_2} &= C_{21}\boldsymbol{a_1} + C_{22}\boldsymbol{a_2} + C_{23}\boldsymbol{a_3}, \\
\boldsymbol{b_3} &= C_{33}\boldsymbol{a_1} + C_{32}\boldsymbol{a_2} + C_{33}\boldsymbol{a_3}.
\end{aligned} \tag{3.2}$$

where $C_{ij} \equiv \boldsymbol{b_i} \cdot \boldsymbol{a_j}$ is called *direction cosine* as it corresponds to the cosine of the angle between $\boldsymbol{b_i}$ and $\boldsymbol{a_j}$. When the previous equation set is written in matrix form, we have:

$$\begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{11} & C_{12} & C_{13} \\ C_{11} & C_{12} & C_{13} \end{bmatrix} \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix} = \boldsymbol{C}^{B/A} \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix} \tag{3.3}$$

Here $\boldsymbol{C}^{B/A}$ is called the *direction cosine matrix*, also known as the *rotation matrix* or *coordinate transformation matrix* from A to B[100]. The direction cosines, the elements of the direction cosine matrix, are not all independent [99].

The direction cosine matrix is an orthonormal matrix as the basis vectors of each reference frames are orthogonal, so:

$$\boldsymbol{C}^{-1} = \boldsymbol{C}^{\mathrm{T}} \tag{3.4}$$

and:

$$\boldsymbol{C}\boldsymbol{C}^{\mathrm{T}} = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C} = \mathbb{1} \tag{3.5}$$

where $\mathbb{1}$ is the identity matrix. When the orientation is preserved, an additional condition occurs:

$$\Big|C\Big| = \mathbb{1} \tag{3.6}$$

Matrices satisfying the last two properties belong to the special orthogonal group SO(3). The following relations are valid between $\boldsymbol{C}^{B/A}$ — the direction cosine matrix

of B relative to A or the direction cosine matrix from A to B — and $\boldsymbol{C}^{A/B}$ — the direction cosine matrix of A relative to B, or the direction cosine matrix from B to A — :

$$\begin{aligned}[\boldsymbol{C}^{A/B}]^{-1} &= [\boldsymbol{C}^{A/B}]^{\mathrm{T}} = \boldsymbol{C}^{B/A} \\ [\boldsymbol{C}^{B/A}]^{-1} &= [\boldsymbol{C}^{B/A}]^{\mathrm{T}} = \boldsymbol{C}^{A/B}\end{aligned} \tag{3.7}$$

The direction cosine maps the vectors from reference frame to body frame. Let us write an arbitrary vector $\boldsymbol{H}$ in the reference frame A and in the body frame B:

$$\begin{aligned}\boldsymbol{H} &= H_1\boldsymbol{a_1} + H_2\boldsymbol{a_2} + H_3\boldsymbol{a_3} \\ &= H_1^{'}\boldsymbol{b_1} + H_2^{'}\boldsymbol{b_2} + H_3^{'}\boldsymbol{b_3}\end{aligned} \tag{3.8}$$

Using the direction cosine matrix in the following equation, components of the arbitrary vector $\boldsymbol{H}$ are transformed from A to B:

$$\begin{bmatrix} H_1^{'} \\ H_2^{'} \\ H_3^{'} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b_1}\cdot\boldsymbol{a_1} & \boldsymbol{b_1}\cdot\boldsymbol{a_2} & \boldsymbol{b_1}\cdot\boldsymbol{a_3} \\ \boldsymbol{b_2}\cdot\boldsymbol{a_1} & \boldsymbol{b_2}\cdot\boldsymbol{a_2} & \boldsymbol{b_2}\cdot\boldsymbol{a_3} \\ \boldsymbol{b_3}\cdot\boldsymbol{a_1} & \boldsymbol{b_3}\cdot\boldsymbol{a_2} & \boldsymbol{b_3}\cdot\boldsymbol{a_3} \end{bmatrix} \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} = \boldsymbol{C}^{B/A} \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} \tag{3.9}$$

Euler Angles

One of the approaches used to represent the attitude is the use of Euler angles. It is a procedure to rotate three times in succession about one axis of the rotated body fixed reference frame. The first rotation is about any of the fixed body axes. The second one is about any of the other two axes which have not been used in the first rotation. The third one is about one of the axes which has not been used in the second rotation. The result is a combination of 12 sets of rotation types. A sequence of rotations about three different axes of reference frame A, describing the orientation of the body frame B with respect to reference frame A, can be represented as:

$$
\begin{aligned}
&\boldsymbol{C}_3(\theta_3) : A^{'} \leftarrow A, \\
&\boldsymbol{C}_2(\theta_2) : A^{''} \leftarrow A^{'}, \\
&\boldsymbol{C}_1(\theta_1) : B \leftarrow A^{''}.
\end{aligned}
\tag{3.10}
$$

The direction cosine matrix B relative to A can be given as:

$$
\boldsymbol{C}^{B/A} = \boldsymbol{C}_1(\theta_1)\boldsymbol{C}_2(\theta_2)\boldsymbol{C}_3(\theta_3) \tag{3.11}
$$

where θ_1, θ_2, θ_3 are the Euler angles. $\boldsymbol{C}_i(\theta_i)$ denotes a rotation of angle θ_i, about the i^{th} axis of the body frame. The orientation of B with respect to A is given as:

$$
\begin{aligned}
\boldsymbol{C}^{B/A} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 \\ 0 & -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} \cos\theta_3 & \sin\theta_3 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta_2\cos\theta_3 & \cos\theta_2\sin\theta_3 & -\sin\theta_2 \\ \sin\theta_1\sin\theta_2\cos\theta_3 - \cos\theta_1\sin\theta_3 & \sin\theta_1\sin\theta_2\cos\theta_3 + \cos\theta_1\cos\theta_1\cos\theta_3 & \sin\theta_1\cos\theta_2 \\ \cos\theta_1\sin\theta_2\cos\theta_3 + \sin\theta_1\sin\theta_3 & \cos\theta_1\sin\theta_2\sin\theta_3 - \sin\theta_1\cos\theta_3 & \cos\theta_1\cos\theta_2 \end{bmatrix}
\end{aligned}
\tag{3.12}
$$

For aircrafts, the Euler angles $(\theta_1, \theta_2, \theta_3)$ are called roll, pitch and yaw, respectively, and can be seen in Fig. 3-2.

The rotation sequence can be selected in different ways depending on the needs of the problem. An example would be selecting a rotation sequence for a transitioning vehicle. The assumption that the pitch angle is constrained to $0 < \theta < 90^\circ$ for a conventional drone to avoid singularity, is not really feasible for a transitioning vehicle which encounters $-90 < \theta < 90$ pitch during the whole flight envelope. For such a problem, selecting the sequence as yaw-roll-pitch, rather than the conventional yaw-pitch-roll sequence, is useful.

Thus, if the set of rotations are selected in a different way, such as:

$$\begin{aligned} \boldsymbol{C}_2(\theta_2) &: A^{'} \leftarrow A, \\ \boldsymbol{C}_3(\theta_3) &: A^{''} \leftarrow A^{'}, \\ \boldsymbol{C}_1(\theta_1) &: B \leftarrow A^{''}. \end{aligned} \tag{3.13}$$

then the direction cosine matrix would differ from Eq. 3.12, and be given as:

$$\boldsymbol{C}^{B/A} = \begin{bmatrix} \cos\theta_2 \cos\theta_3 & \sin\theta_2 & -\cos\theta_2 \sin\theta_3 \\ -\cos\theta_1 \sin\theta_2 \cos\theta_3 + \sin\theta_1 \sin\theta_3 & \cos\theta_1 \cos\theta_2 & \cos\theta_1 \sin\theta_2 \sin\theta_3 + \sin\theta_2 \cos\theta_3 \\ \sin\theta_1 \sin\theta_2 \cos\theta_3 + \cos\theta_1 \sin\theta_3 & -\sin\theta_1 \cos\theta_2 & -\sin\theta_1 \sin\theta_2 \sin\theta_3 + \cos\theta_1 \cos\theta_3 \end{bmatrix} \tag{3.14}$$

Quaternions

The idea behind this representation is the Euler's *eigen-axis rotation* theorem. According to Euler, "the most general displacement of a rigid body with one point fixed is a rotation about some axis." In other words, it states that there exists a unit vector $\boldsymbol{e}$, with the property:

$$\boldsymbol{C}\boldsymbol{e} = \boldsymbol{e} \tag{3.15}$$

The $\boldsymbol{e}$ vector has the same components in body and reference frames :

$$\begin{aligned} \boldsymbol{e} &= e_1\boldsymbol{a_1} + e_2\boldsymbol{a_2} + e_3\boldsymbol{a_3} \\ &= e_1\boldsymbol{b_1} + e_2\boldsymbol{b_2} + e_3\boldsymbol{b_3} \end{aligned} \tag{3.16}$$

By rotating a rigid body about this axis $\boldsymbol{e}$, a rotation from any given orientation to any other orientation can be achieved. Such an axis is called an *Euler axis*, after the Swiss mathematician and theoretical physicist, Leonard Euler (1707-1783).

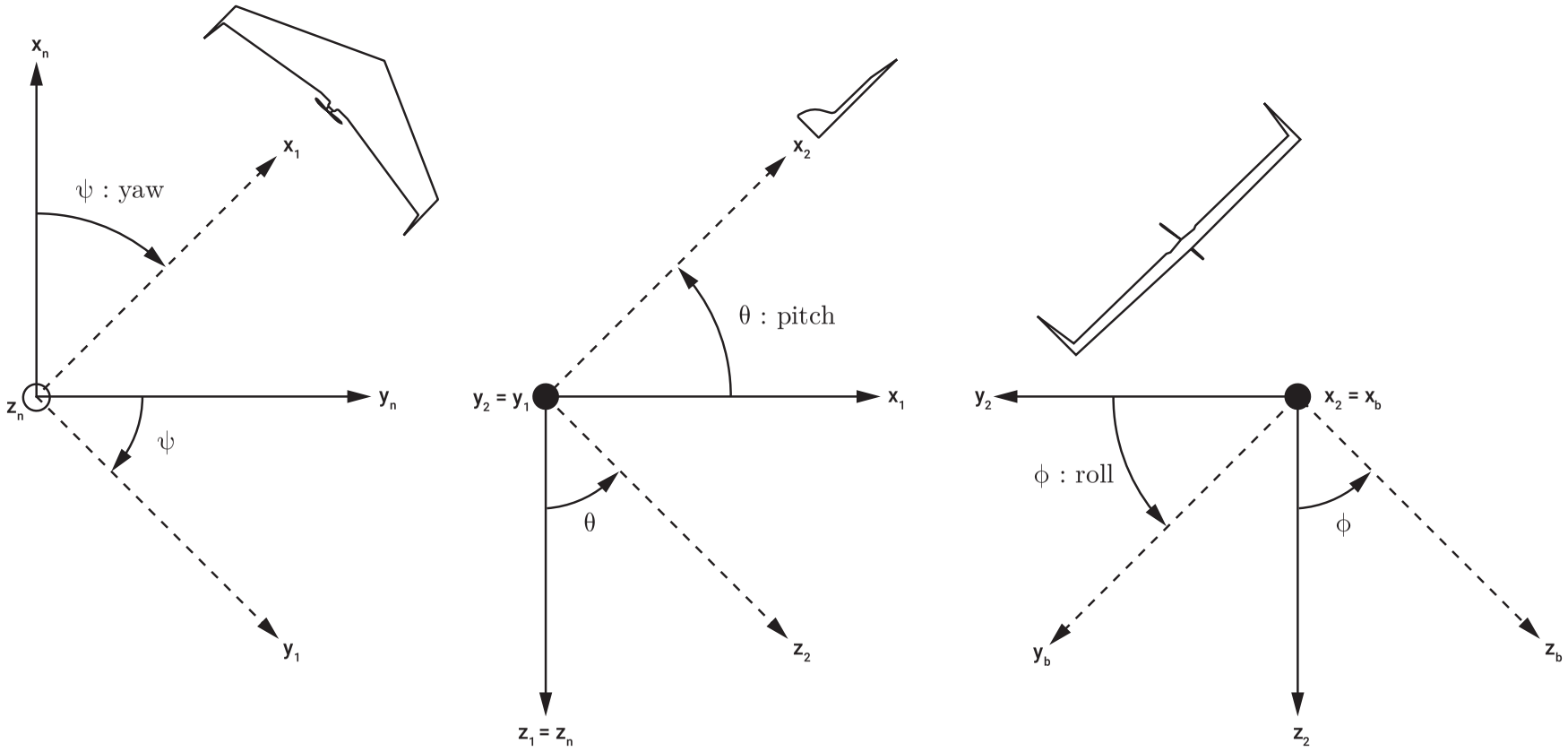

Figure 3-2: Euler angle sequence [30]

Euler symmetric parameters, also known as *quaternions*, are defined as:

$$\boldsymbol{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\phi/2) \\ e_1 \sin(\phi/2) \\ e_2 \sin(\phi/2) \\ e_3 \sin(\phi/2) \end{bmatrix} \tag{3.17}$$

where ϕ is the *Euler rotation angle*. Hamilton (1805-1865) is considered to be the first to mention quaternions. For ease of the mathematical representations, we define a vector as :

$$\boldsymbol{q} = \boldsymbol{e} \sin(\phi/2) \tag{3.18}$$

Euler symmetric parameters are not independent, and satisfy the constraint:

$$\boldsymbol{q}^{\mathrm{T}}\boldsymbol{q} + q_0^2 = q_0^2 + q_1^2 + q_1^2 + q_3^2 = 1 \tag{3.19}$$

The direction cosine matrix can also be written in terms of quaternions as below

$$\begin{aligned} C(\boldsymbol{q}) &= \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 + q_3q_0) & 2(q_1q_3 - q_2q_0) \\ 2(q_1q_2 - q_3q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 + q_1q_0) \\ 2(q_1q_3 + q_2q_0) & 2(q_2q_3 - q_1q_0) & -q_1^2 - q_2^2 + q_3^2 + q_0^2 \end{bmatrix} \\ &= (q_0^2 - \boldsymbol{q}^{\mathrm{T}}\boldsymbol{q})\mathbb{1} + 2\boldsymbol{q}\boldsymbol{q}^{\mathrm{T}} - 2q_0\boldsymbol{Q} \end{aligned} \tag{3.20}$$

where:

$$\boldsymbol{Q} = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{3.21}$$

Given the direction cosine matrix, quaternions can be calculated as:

$$\boldsymbol{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \frac{1}{4q_0} \begin{bmatrix} C_{23} - C_{32} \\ C_{31} - C_{13} \\ C_{12} - C_{21} \end{bmatrix} \tag{3.22}$$
$$q_0 = \pm\frac{1}{2}(1 + C_{11} + C_{22} + C_{33})^{\frac{1}{2}}$$

Attitude parametrization selection

In 1776, Euler showed that SO(3) has three dimensions [90]. Representations with more than three parameters are subject to constraints. Also, [90] states that no parameter set can be both global and nonsingular. So, we are faced with choosing the representation parameters as either singular or redundant. Euler angle representation has its advantages, such as having a clear physical interpretation and minimum parameter set achieving no redundancy. However, due to their important disadvantage, that is, the possibility of having singularities when describing motion, Euler angles are not selected to represent attitude in this study. Another attitude parametrization, the Gibbs vector, is not often used, and can be thought of as an interval step on the way to quaternion parametrization. There are other representation types, which are not often preferred, such as the axis-azimuth representation, which will not be discussed in this thesis [7]. Quaternion (Euler symmetric parameters) representation will be used in this study due to its advantageous nature in simulations, and having no singularities. Another advantage of quaternions is that the kinematic equations are linear in terms of quaternions. Also, quaternion multiplication offers a useful way to express composite rotations. With all of these preferable properties, quaternions are the choice for attitude representations for many attitude control missions. Table 3.1 shows a brief comparison of attitude representations [7].

In the previous sections, a variety of parameters to represent rotations are identified and the inherent properties of each technique have been examined. Now we examine how to represent attitude depending on time; namely the equations of motion. Equations of motion are generally presented in two sections: the kinematic equations

Table 3.1: Attitude parametrizations of the rotation group SO(3) [7]

Representation	Number of parameter set	Properties
Euler angles	3	Minimal set Clear physical interpretation Often computed directly Trigonometric functions in rotation matrix No simple composition rule Singular for certain rotations Trigonometric functions in kinematic relation
Quaternions	4	Easy orthogonality of rotation matrix Bilinear composition rule Not singular at any rotation Linear kinematic equations No clear physical interpretation One redundant parameter Simple kinematic relation
Gibbs vector	4	Minimal set Clear composition rule Singular for certain rotations Simple kinematic relation

and dynamic equations. The kinematic equations provide the relationships between the time derivative of the attitude representation and the angular velocity, while the dynamic (or kinetic) equations describe the development of angular velocities under the influence of external moments.

3.1.2 Attitude kinematics

The time evolution of attitude is identified by a set of first order differential equations called the kinematic equations. The angular velocity of a reference frame B with respect to a reference frame A given in the B frame, can be written as follows:

$$\boldsymbol{\omega}_B^{B/A} = \omega_1 \boldsymbol{b_1} + \omega_2 \boldsymbol{b_2} + \omega_3 \boldsymbol{b_3} \tag{3.23}$$

Recalling Eq. 3.3 and orthonomality property in Eq. 3.4:

$$\begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix} = \left[\boldsymbol{C}_B^{B/A}\right]^{-1} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} \tag{3.24}$$

Taking the time derivative with respect to frame A:

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix}\Bigg|_A &= \left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} + \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \dot{\boldsymbol{b_1}} \\ \dot{\boldsymbol{b_2}} \\ \dot{\boldsymbol{b_3}} \end{bmatrix} = \left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} + \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\omega} \times \boldsymbol{b_1} \\ \boldsymbol{\omega} \times \boldsymbol{b_2} \\ \boldsymbol{\omega} \times \boldsymbol{b_3} \end{bmatrix} \\ &= \left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} + \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} \end{aligned} \tag{3.25}$$

If the skew-symmetric matrix can be represented as $\boldsymbol{\Omega}$:

$$\boldsymbol{\Omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{3.26}$$

and if we rearrange the terms as such:

$$\left[\left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} - \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \boldsymbol{\Omega}\right] \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.27}$$

implies:

$$\left[\left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} - \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \boldsymbol{\Omega}\right] = \boldsymbol{0} \tag{3.28}$$

Taking the transpose and using the relationship $\Omega^{\mathrm{T}} = -\Omega$, the kinematic differential equation for the direction cosine matrix can be written as:

$$\dot{\boldsymbol{C}}_B^{B/A} + \boldsymbol{\Omega}\boldsymbol{C}_B^{B/A} = \mathbf{0} \tag{3.29}$$

Angular velocity components, given in the B reference frame, can be written as:

$$\begin{aligned}
\omega_1 &= \dot{C}_{21}C_{31} + \dot{C}_{22}C_{32} + \dot{C}_{23}C_{33} \\
\omega_2 &= \dot{C}_{31}C_{11} + \dot{C}_{32}C_{12} + \dot{C}_{33}C_{13} \\
\omega_3 &= \dot{C}_{11}C_{21} + \dot{C}_{12}C_{22} + \dot{C}_{13}C_{23}
\end{aligned} \tag{3.30}$$

From that point, derivation depends on the attitude parameters used. When the direction cosines and their derivatives are substituted with their equivalents in terms of Euler angles, the result will give the time dependence of Euler angles. In this study however, due to reasons already discussed, quaternions are selected to represent the attitude, so the direction cosines will be written in terms of quaternions:

$$\begin{aligned}
\omega_1 &= 2(\dot{q}_1 q_0 + \dot{q}_2 q_3 - \dot{q}_3 q_2 - \dot{q}_0 q_1) \\
\omega_2 &= 2(\dot{q}_2 q_0 + \dot{q}_3 q_1 - \dot{q}_1 q_3 - \dot{q}_0 q_2) \\
\omega_3 &= 2(\dot{q}_3 q_0 + \dot{q}_1 q_2 - \dot{q}_2 q_1 - \dot{q}_2 q_3)
\end{aligned} \tag{3.31}$$

The fourth equation comes from the differentiation of the quaternion norm constraint:

$$0 = 2(\dot{q}_0 q_0 + \dot{q}_1 q_1 + \dot{q}_2 q_2 + \dot{q}_3 q_3) \tag{3.32}$$

Writing in matrix form:

$$\begin{bmatrix} 0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = 2 \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \\ -q_1 & q_0 & q_3 & -q_2 \\ -q_2 & -q_3 & q_0 & q_1 \\ -q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \tag{3.33}$$

As the matrix, which is composed of quaternions, is an orthonormal matrix, the kinematic equations of motion in terms of quaternions can be given as [100]:

$$\begin{aligned} \begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & -q_3 & q_2 \\ q_2 & q_3 & q_0 & -q_1 \\ q_3 & -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} 0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \\ &= \frac{1}{2} \begin{bmatrix} 0 & -\omega_1 & -\omega_2 & -\omega_3 \\ \omega_1 & 0 & \omega_3 & -\omega_2 \\ \omega_2 & -\omega_3 & 0 & \omega_1 \\ \omega_3 & \omega_2 & -\omega_1 & 0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \end{aligned} \tag{3.34}$$

In a more compact form:

$$\boxed{\begin{aligned} \dot{q}_0 &= -\frac{1}{2} \boldsymbol{q}_\nu^T \boldsymbol{\omega}_B^{B/A} \\ \dot{\boldsymbol{q}}_\nu &= \frac{1}{2} \Big(\boldsymbol{q}_\nu^\times + q_0 \boldsymbol{I}_3 \Big) \boldsymbol{\omega}_B^{B/A} \end{aligned}} \tag{3.35}$$

where:

$$\boldsymbol{x}^\times = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{3.36}$$

Finally the kinematic equations of motion for the aircraft in Eq. 3.34 (first equation

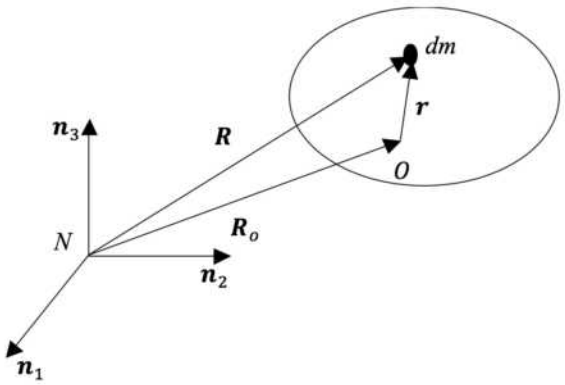

Figure 3-3: Rigid body rotating about an arbitrary point O, given in the N frame

row) can be rewritten as in Eq. 3.37, since the first column of the matrix on the right side of the equation (first equation row in Eq. 3.34) is multiplied by zero:

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3.37}$$

3.1.3 Attitude dynamics

The equation describing the rotational motion of a rigid body moving relative to an inertial frame can written as [100]:

$$\int \boldsymbol{r} \times \ddot{\boldsymbol{R}} dm = \boldsymbol{M_0} \tag{3.38}$$

Here, the rotation of the rigid body takes place about an arbitrary point O. Let us take an infinitesimal mass element dm. In Fig.3-3, $\boldsymbol{r}$ is the position vector of dm relative to O, $\boldsymbol{R}$ is the position vector of dm relative to the origin of the inertial frame, $\ddot{\boldsymbol{R}}$ is the acceleration of dm, and $\boldsymbol{M_0}$ is the total external torque about point O.

Now, let us take a body fixed frame B with the origin at the center of mass of the rigid body, as shown in Fig. 3-4. In Fig. 3-4, $\boldsymbol{\rho}$ is the position vector of dm mass

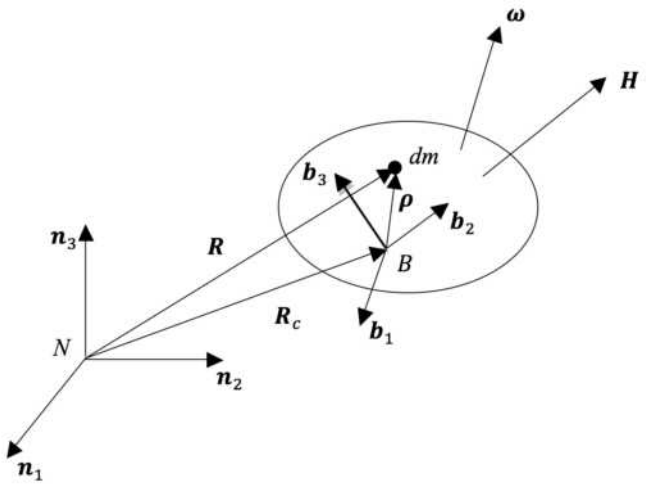

Figure 3-4: Body frame B attached to the rigid body, given in the inertial frame N

with respect to the center of mass of the rigid body, $\boldsymbol{R}_c$ is the position vector of the center of mass of the rigid body with respect to the origin of the inertial frame N, and $\boldsymbol{R}$ is the position vector of dm with respect to the origin of the inertial frame N.

The angular velocity of the rigid body in the inertial frame N is denoted as $\boldsymbol{\omega} \equiv \boldsymbol{\omega}_B^{B/N}$. It represents the angular velocity of the body frame B with respect to inertial frame N given in the body frame B. Angular momentum vector $\boldsymbol{H}$ of a rigid body about its center of mass is given by :

$$\boldsymbol{H} = \int \boldsymbol{\rho} \times \dot{\boldsymbol{R}} dm \tag{3.39}$$

Since $\boldsymbol{R}_C$ is constant:

$$\dot{\boldsymbol{R}} = \dot{\boldsymbol{R}}_C + \dot{\boldsymbol{\rho}} = \dot{\boldsymbol{\rho}} \tag{3.40}$$

and from rigidity of the body:

$$\dot{\boldsymbol{\rho}} \equiv \left\{ \frac{d\boldsymbol{\rho}}{dt} \right\}_N = \cancel{\left\{ \frac{d\boldsymbol{\rho}}{dt} \right\}_B} + \boldsymbol{\omega} \times \boldsymbol{\rho} = \boldsymbol{\omega} \times \boldsymbol{\rho} \tag{3.41}$$

Then, the angular momentum is given as:

$$\boldsymbol{H} = \int \boldsymbol{\rho} \times \dot{\boldsymbol{R}} dm = \int \boldsymbol{\rho} \times \dot{\boldsymbol{\rho}} dm = \int \boldsymbol{\rho} \times (\boldsymbol{\omega} \times \boldsymbol{\rho}) dm \tag{3.42}$$

The components of $\boldsymbol{\rho}$ and $\boldsymbol{\omega}$ in the body frame B are written as:

$$\begin{aligned} \boldsymbol{\rho} &= \rho_1 \boldsymbol{b}_1 + \rho_2 \boldsymbol{b}_2 + \rho_3 \boldsymbol{b}_3 \\ \boldsymbol{\omega} &= \omega_1 \boldsymbol{b}_1 + \omega_2 \boldsymbol{b}_2 + \omega_3 \boldsymbol{b}_3 \end{aligned} \tag{3.43}$$

Then, the angular momentum can be written as:

$$\boldsymbol{H} = H_1 \boldsymbol{b}_1 + H_2 \boldsymbol{b}_2 + H_3 \boldsymbol{b}_3 \tag{3.44}$$

where

$$\begin{aligned} H_1 &= I_{11}\omega_1 + I_{12}\omega_2 + I_{13}\omega_3 \\ H_2 &= I_{21}\omega_1 + I_{22}\omega_2 + I_{23}\omega_3 \\ H_3 &= I_{31}\omega_1 + I_{32}\omega_2 + I_{33}\omega_3 \end{aligned} \tag{3.45}$$

Writing in matrix form gives:

$$\begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} = \begin{bmatrix} I_{11} & I_{12} & I_{13} \\ I_{21} & I_{22} & I_{23} \\ I_{31} & I_{32} & I_{33} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3.46}$$

and in a compact form:

$$\boldsymbol{H} = \boldsymbol{I}\boldsymbol{\omega} \tag{3.47}$$

where $\boldsymbol{I}$ is called the inertia matrix of the rigid body about a body fixed reference frame B with the origin at the center of the mass of the rigid body.

Let us write a set of axes to achieve all the products of inertia, or all of the

elements of the inertia matrix except diagonal elements, are zero. This set is called the principal axes and the moments of inertia are called the principal moments of inertia. Assuming the axes of the body reference frame B are the principal axes, then the equation for the angular momentum becomes:

$$\begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} = \begin{bmatrix} I_{11} & 0 & 0 \\ 0 & I_{22} & 0 \\ 0 & 0 & I_{33} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \tag{3.48}$$

Now, we express the rotational equations of motion — also known as Euler's equations of motion — for a rigid body. Remember the angular momentum equation:

$$\boldsymbol{M} = \dot{\boldsymbol{H}} \tag{3.49}$$

where $\boldsymbol{H}$ is the angular momentum vector of the rigid body about its center of mass and $\boldsymbol{M}$ is the external moment acting on the rigid body about its center of mass. We can also write it as:

$$\dot{\boldsymbol{H}} = \left\{ \frac{d\boldsymbol{H}}{dt} \right\}_N = \left\{ \frac{d\boldsymbol{H}}{dt} \right\}_B + \boldsymbol{\omega}_B^{B/N} \times \boldsymbol{H} = \boldsymbol{M} \tag{3.50}$$

By taking the time derivative of Eq. 3.47, assuming the inertia is not dependent on time, and evaluating in Eq. 3.50, Euler's rotational equation of motion can be written as:

$$\boxed{\dot{\boldsymbol{\omega}}_B^{B/N} = \boldsymbol{I}_B^{-1}(\boldsymbol{M}_B - \boldsymbol{\omega}_B^{B/N} \times \boldsymbol{I}_B \boldsymbol{\omega}_B^{B/N})} \tag{3.51}$$

3.2 Translation modeling

The movement of any object can be represented by changes in its location (translation) or changes in its attitude (rotation), or a combination of both. Studying aircraft motion is complicated since those two motions are coupled; for example, a rotation might cause a change in aerodynamic forces which affects the translation. Thus, to

ease the modeling process, the aircraft is assumed to be a point mass, with all of its mass collected at its center of gravity, while it translates from one point to the other. Then, the motion of the that point (at its center of gravity) is described by Newton's laws of motion. The translations are in direct response to external forces, namely the lift, drag, thrust, and weight. Unfortunately, some of these forces depend on the attitude of the aircraft.

3.2.1 Translational kinematics

The time change of the position of the aircraft $\boldsymbol{x}_N$ expressed in the navigation frame can be written in terms of translational velocity $\boldsymbol{v}_B$, expressed in the body frame as:

$$\begin{aligned} \dot{\boldsymbol{x}}_N &= \frac{d}{dt}(\boldsymbol{x}_N) \\ &= \frac{d}{dt}(\boldsymbol{C}_B^N \boldsymbol{x}_B) \\ &= \dot{\boldsymbol{C}}_B^N \boldsymbol{x}_B + \boldsymbol{C}_B^N \dot{\boldsymbol{x}}_B \\ &= \boldsymbol{C}_B^N \boldsymbol{v}_B \end{aligned} \tag{3.52}$$

using $\boldsymbol{x}_B = 0$. Eq. 3.52 can also be written as :

$$\boxed{\begin{bmatrix} \dot{x}_n \\ \dot{x}_e \\ \dot{x}_d \end{bmatrix} = \boldsymbol{C}_B^N \begin{bmatrix} u \\ v \\ w \end{bmatrix}} \tag{3.53}$$

where, $\boldsymbol{v}_B = [u \quad v \quad w]^T$ is the inertial velocity of the center of mass of the body expressed in the body frame B and $\dot{\boldsymbol{x}}_N = [\dot{x}_n \quad \dot{x}_e \quad \dot{x}_d]^T$ is the ground speed vector expressed in the navigation frame N. N is assumed to be a local inertial frame. $\boldsymbol{C}_B^N$ is the direction cosine matrix which transforms a vector expressed in the body frame to its equivalent expressed in the navigation frame N.

3.2.2 Translational dynamics

The aircraft is assumed to be a point mass, with all of its mass collected at its center of gravity while it translates from one point to an other. Then, the motion of that point is described by Newton's laws of motion. The translations are in direct response to external forces, namely the lift, drag, thrust, and weight. From now on, the aircraft will be assumed to be flying over a small region compared to size of the Earth, so that the Earth is locally flat to neglect centripetal acceleration (due to the Earth's curvature). Another assumption is that the frame attached to the Earth is an inertial frame, by ignoring Coriolis acceleration, so that Newton's laws apply:

$$\sum_j \boldsymbol{F}_j = \frac{d}{dt}\big(m\boldsymbol{v}\big)\bigg|_I \tag{3.54}$$

Unfortunately, those forces that directly effect the translation depend on the attitude of the aircraft, complicating the dynamics.

Here, the subscript I represents the frame in which the time derivation occurs, which is an inertial frame in this case. To represent time derivation in the inertial frame in terms of time derivation in body frame, the relative rotation of the body frame with respect to the inertial frame should be included, such that:

$$\frac{d}{dt}\big(m\boldsymbol{v}\big)\bigg|_I = \frac{d}{dt}\big(m\boldsymbol{v}\big)\bigg|_B + \boldsymbol{\omega}^{B/I} \times \big(m\boldsymbol{v}\big) \tag{3.55}$$

Substituting Eq. 3.55 in Eq. 3.54 and then projecting the vector variables in the body frame, as well as assuming that mass is constant, gives:

$$\frac{1}{m}\bigg(\sum_j \boldsymbol{F}_{B_j}\bigg) = \frac{d\big(\boldsymbol{v}_B\big)}{dt}\bigg|_B + \boldsymbol{\omega}_B^{B/I} \times \boldsymbol{v}_B \tag{3.56}$$

Writing the summation of forces in terms of the forces acting on the aircraft :

$$\frac{1}{m}\Big(m\boldsymbol{g}_B + \boldsymbol{F}_{thrust_B} + \boldsymbol{F}_{aero_B}\Big) = \begin{bmatrix}\dot{u}\\ \dot{v}\\ \dot{w}\end{bmatrix} + \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times \begin{bmatrix}u\\ v\\ w\end{bmatrix} \tag{3.57}$$

Rearranging terms and writing the forces in more detail gives :

$$\boxed{\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} -g\sin\theta \\ g\sin\phi\cos\theta \\ g\cos\phi\cos\theta \end{bmatrix} + \frac{1}{m}\begin{bmatrix} F_{thrust} \\ 0 \\ 0 \end{bmatrix} + \frac{1}{m}\begin{bmatrix} X^b \\ Y^b \\ Z^b \end{bmatrix} - \begin{bmatrix} qw - rv \\ ru - pw \\ pv - qu \end{bmatrix}} \tag{3.58}$$

which are the equations of motion of the aircraft.

3.3 Drone model

The calculation of aerodynamic forces and moments that are necessary to solve the equations of the motion of a drone are given in this section. During the course of this research, two kind of drones have been simulated : a drone from ETH university [30] and MAKO (used in ENAC drone lab) given in Fig. 3-6. ETH drone has two ailerons, two elevators and a rudder as its control surfaces, while MAKO has only two control surfaces: elevons. An example of an elevon can be seen in the schematic of Zagi given in Fig. 3-5. Changed in the same direction, elevons are used as an elevators (which change the pitch), while when changed in the reverse direction they act as ailerons (they change the roll).

First, to calculate the moments of inertia of MAKO, it is hanged from two strings, at different orientations, as shown in Fig. 3-7, and the measurements are performed by timing the oscillation period for each axis. The resultant moment of inertias are given Table 3.2.

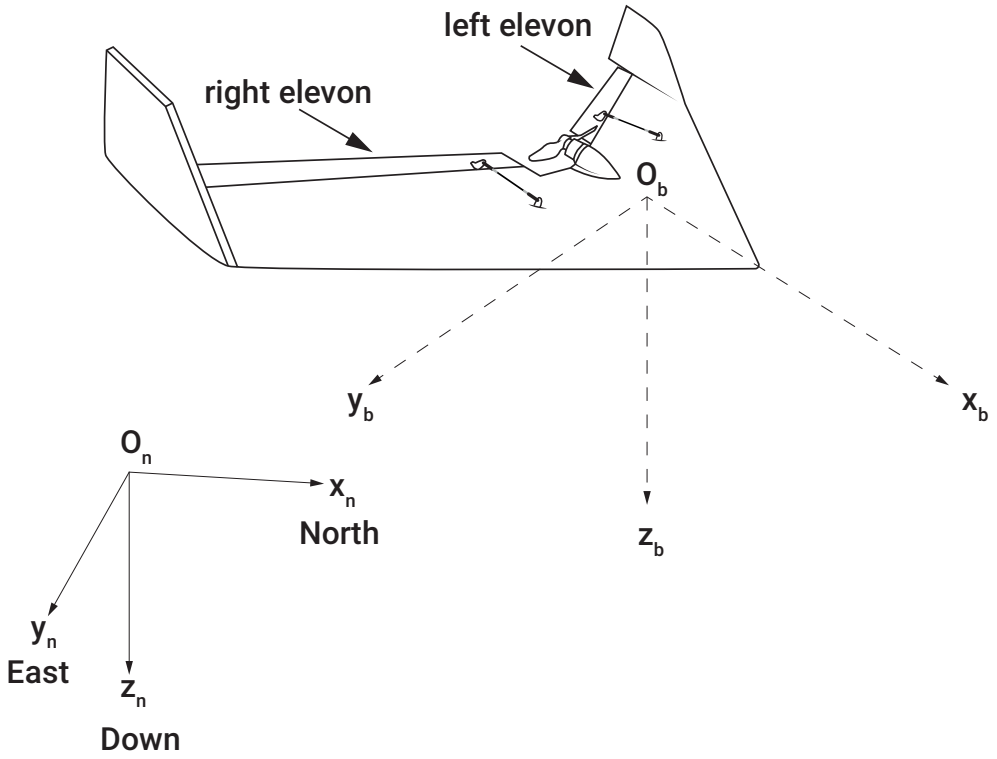

Figure 3-5: Body fixed frame and North East Down (NED) frame representations

Table 3.2: General specifications of MAKO [16]

Parameter	**Value**	**Definition**
Wing span (b)	1.288	$[m]$
Wing surface area (S)	0.27	$[m^2]$
Mean aero chord ($\bar{c}$)	0.21	$[m]$
Take-off mass (m)	$0.7 - 2.0$	$[kg]$
Flight velocity	$10 - 25$	$[m/s]$
I_{xx}	0.02471284	$[kg \cdot m^2]$
I_{yy}	0.015835159	$[kg \cdot m^2]$
I_{zz}	0.037424499	$[kg \cdot m^2]$

Figure 3-6: MAKO

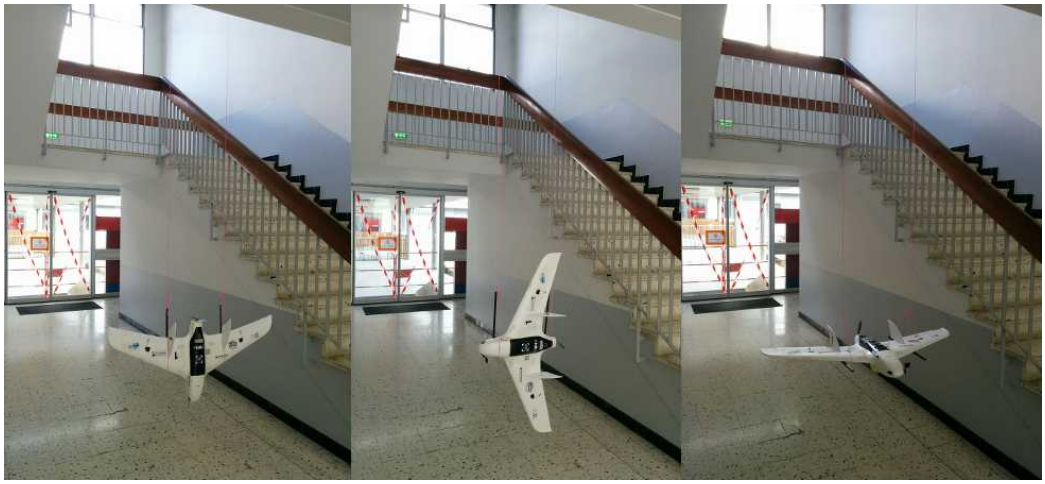

Figure 3-7: Moments of inertia measurements for each axis, I_{xx}, I_{yy}, I_{zz}.

Table 3.3: Parameters of ETH drone [30]

Parameter	**Value**	**Definition**
Wing span (b)	3.1	$[m]$
Wing surface area (S)	1.80	$[m^2]$
Mean aero chord ($\bar{c}$)	0.58	$[m]$
Take-off mass (m)	28	$[kg]$
Propeller diameter (D)	28	$[m]$
Time constant of the engine (τ_n)	28	$[m]$
I_{xx}	2.56	$[kg \cdot m^2]$
I_{yy}	10.9	$[kg \cdot m^2]$
I_{zz}	11.3	$[kg \cdot m^2]$
I_{zx}	0.5	$[kg \cdot m^2]$
I_{xz}	0.5	$[kg \cdot m^2]$

3.3.1 Modeling of aerodynamic moments

The attitude of the aircraft changes with the torques applied to the airframe:

$$\boldsymbol{M}_B = \begin{bmatrix} L_b \\ M_b \\ N_b \end{bmatrix} \tag{3.59}$$

Here, L_b is the roll torque, M_b is the pitch torque, and N_b is the yaw torque given in body frame shown in Fig. 3-5.

The stability derivatives required to calculate the moments are given under Table 3.4 for ETH drone and under Table 3.5 for MAKO. The values for the ETH drone are taken from [30] while for MAKO they are calculated via AVL. AVL is an open source program developed at MIT, and uses the vortex-lattice method for the aerodynamic and stability calculations. The output of the program is linearized at a selected trim condition, therefore all of the coefficients are calculated around the equilibrium point at $14m/s$ cruise flight condition. The center of gravity is located at $X_{CG} = 0.295\,m$, which corresponds to $8\,\%$ of the positive static margin that has been flight tested.

Roll torque

Roll torque is given as a multiplication of dynamic pressure $\bar{q}$, wing surface area S, wingspan b, and dimensionless roll torque C_L as :

$$L_B = \bar{q}\, S\, b\, C_L \tag{3.60}$$

Here, the dynamic pressure is calculated as :

$$\bar{q} = \frac{\rho V_T^2}{2} \tag{3.61}$$

while the air density ρ is given by the international standard atmosphere model for low altitude (<11000m) as :

$$\rho = \frac{p_0 \Big[1 + \frac{ah}{T_0}\Big]^{5.2561}}{R\,T} \tag{3.62}$$

where the temperature T is given as:

$$T = T_0 \Big[1 + \frac{ah}{T_0}\Big] \tag{3.63}$$

with $T_0 = 288.15K$, $a = -6.5 \times 10^{-3}\, K/m$, $R = 287.3\ m^2K^{-1}s^{-2}$ and $p_0 = 1013 \times 10^2\ Nm^{-2}$.

Next, we calculate the dimensionless roll torque in Equ. 3.60. For ETH drone, it is given by [89, 30, 67] :

$$C_L = C_{L_{a1}}\, \delta_{a1} + C_{L_{a2}}\, \delta_{a2} + C_{L_{e1}}\, \delta_{e1} + C_{L_{e2}}\, \delta_{e2} + C_{L_{\tilde{p}}}\, \tilde{p} + C_{L_{\tilde{r}}}\, \tilde{r} + C_{L_\beta}\, \beta \tag{3.64}$$

For MAKO, the dimensionless roll torque is given by :

$$C_L = C_{L_a}\, \delta_a + C_{L_{\tilde{p}}}\, \tilde{p} + C_{L_{\tilde{r}}}\, \tilde{r} + C_{L_\beta}\, \beta \tag{3.65}$$

where δ_{a1}, δ_{a2} are the aileron deflections, δ_{e1}, δ_{e2} are the elevator deflections, δ_a

Table 3.4: Stability derivatives for ETH UAV [30]

Parameter	**Value**	**Definition**
$C_{L_{a1}} = -C_{L_{a2}}$	-3.395×10^{-2}	roll derivative
$C_{L_{e1}} = -C_{L_{e2}}$	-0.485×10^{-2}	roll derivative
$C_{L_{\tilde{p}}}$	-1.92×10^{-1}	roll derivative
$C_{L_{\tilde{r}}}$	3.61×10^{-2}	roll derivative
$C_{L_{\beta}}$	-1.30×10^{-2}	roll derivative
C_{M_1}	2.08×10^{-2}	pitch derivative
$C_{M_{a1}} = C_{M_{a2}}$	0.389×10^{-1}	pitch derivative
$C_{M_{e1}} = C_{M_{e2}}$	2.725×10^{-1}	pitch derivative
$C_{M_{\tilde{q}}}$	-9.83	pitch derivative
$C_{M_{\alpha}}$	-9.03×10^{-2}	pitch derivative
$C_{N_{\delta r}}$	5.34×10^{-2}	yaw derivative
$C_{N_{\tilde{r}}}$	-2.14×10^{-1}	yaw derivative
$C_{N_{\beta}}$	8.67×10^{-2}	yaw derivative

Table 3.5: Stability derivatives for MAKO extracted from AVL program at $14m/s$ equilibrium cruise speed [16]

Parameter	**Value**	**Definition**
C_{L_a}	-0.1956×10^{-2}	roll derivative
$C_{L_{\tilde{p}}}$	-4.095×10^{-1}	roll derivative
$C_{L_{\tilde{r}}}$	6.203×10^{-2}	roll derivative
$C_{L_{\beta}}$	3.319×10^{-2}	roll derivative
C_{M_0}	0	pitch derivative
C_{M_e}	-0.076×10^{-1}	pitch derivative
$C_{M_{\tilde{q}}}$	-1.6834	pitch derivative
$C_{M_{\alpha}}$	-32.34×10^{-2}	pitch derivative
C_{N_0}	0	yaw derivative
C_{N_a}	-0.0126×10^{-2}	yaw derivative
$C_{N_{\tilde{p}}}$	-4.139×10^{-2}	yaw derivative
$C_{N_{\tilde{r}}}$	-0.1002×10^{-1}	yaw derivative
$C_{N_{\beta}}$	2.28×10^{-2}	yaw derivative

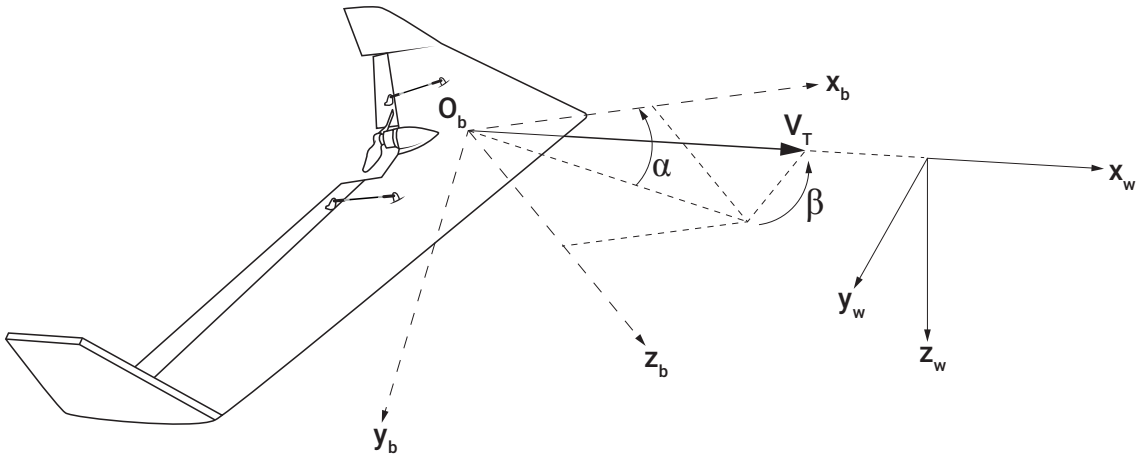

Figure 3-8: Wind frame, airspeed vector $\boldsymbol{V}_T$, angle of attack α and side slip angle β representation [30]

is the aileron deflection for MAKO, β is the sideslip angle representing the angle between airspeed vector $\boldsymbol{V_T}$ and the projection of $\boldsymbol{V_T}$ onto $x_b - z_b$ plane as shown in Fig. 3-8 and given as :

$$\beta = \arcsin \frac{v_T}{V_T} \tag{3.66}$$

and $\tilde{p}, \tilde{r}$ are dimensionless angular rates given by:

$$\tilde{p} = \frac{bp}{2V_T} \qquad \tilde{r} = \frac{br}{2V_T} \tag{3.67}$$

Here, b is the wingspan, p, q are angular rates and V_T is the norm of the airspeed vector such as :

$$V_T = \sqrt{u_T^2 + v_T^2 + w_T^2} \tag{3.68}$$

Airflow acting on the aircraft is described by airspeed vector $\boldsymbol{V_T}$ (see Fig. 3-8). The wind frame is shown in Fig. 3-8 with axes (x_w, y_w, z_w) where x_w points along the airspeed vector $\boldsymbol{V}_T$. Airspeed vector, in terms of its components in body frame and in wind frame, can be given as:

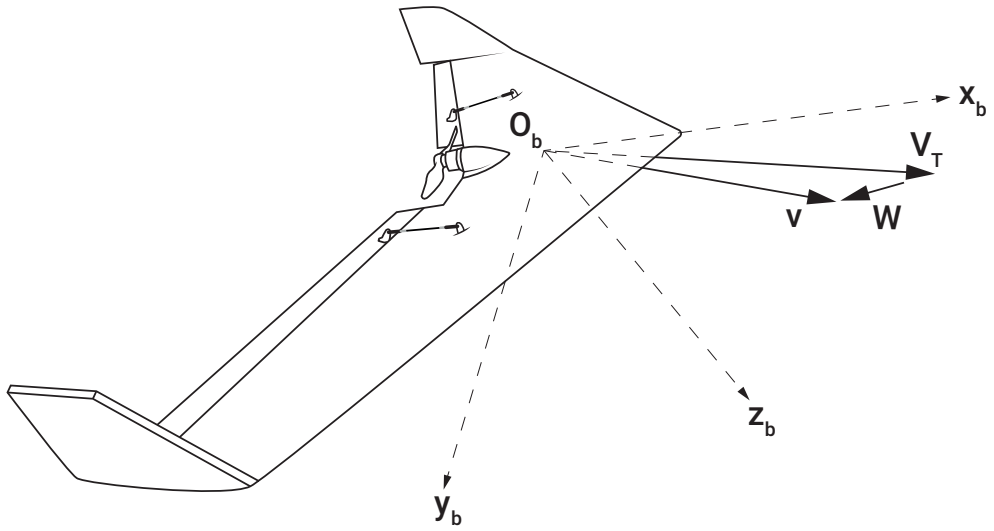

Figure 3-9: Relation revealed between the inertial velocity vector $\boldsymbol{v}$, airspeed vector $\boldsymbol{V}_T$ and wind disturbance $\boldsymbol{W}$ [30]

$$\boldsymbol{V_{T_B}} = \begin{bmatrix} u_T \\ v_T \\ w_T \end{bmatrix} \qquad \boldsymbol{V_{T_W}} = \begin{bmatrix} V_T \\ 0 \\ 0 \end{bmatrix} \tag{3.69}$$

The relationship between $\boldsymbol{V_{T_B}}$ and $\boldsymbol{V_{T_W}}$ can be written as [30]:

$$\boldsymbol{V_{T_B}} = \boldsymbol{C_W^B V_{T_W}} \tag{3.70}$$

Note that the inertial velocity of the aircraft $\boldsymbol{v}_B = [u \quad v \quad w]^T$ is different to the airspeed vector $\boldsymbol{V_{T_B}} = [u_T \quad v_T \quad w_T]^T$. Those two vectors are related to wind velocity $\boldsymbol{W}$ by :

$$\boldsymbol{v} = \boldsymbol{V_T} + \boldsymbol{W} \tag{3.71}$$

When these vectors are projected on to body frame, the relation becomes:

$$\boldsymbol{v}_B = \boldsymbol{V_{T_B}} + \boldsymbol{C_N^B W}_N \tag{3.72}$$

and with components :

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} u_T \\ v_T \\ w_T \end{bmatrix} + \boldsymbol{C_N^B} \begin{bmatrix} W_n \\ W_e \\ W_d \end{bmatrix} \tag{3.73}$$

Pitch torque

Pitch torque is given as a multiplication of dynamic pressure $\bar{q}$, wing surface area S, mean aero chord $\bar{c}$, and dimensionless pitch torque C_M as :

$$M_B = \bar{q}\, S\, \bar{c}\, C_M \tag{3.74}$$

where the dimensionless pitch torque is given as for the ETH drone as :

$$C_M = C_{M_1} + C_{M_{a1}}\, \delta_{a1} + C_{M_{a2}}\, \delta_{a2} + C_{M_{e1}}\, \delta_{e1} + C_{M_{e2}}\, \delta_{e2} + C_{M_{\tilde{q}}}\, \tilde{q} + C_{M_\alpha}\, \alpha \tag{3.75}$$

and for MAKO :

$$C_M = C_{M_0} + C_{M_e}\, \delta_e + C_{M_{\tilde{q}}}\, \tilde{q} + C_{M_\alpha}\, \alpha \tag{3.76}$$

where the variables not introduced up to now are the dimensionless roll rate $\tilde{q}$, elevator deflection δ_e and angle of attack α. Angle of attack α is defined as the angle between the projection of the airspeed vector $\boldsymbol{V_T}$ onto the $x_b - z_b$ plane and x_b axis as can be seen in Fig. 3-8, and calculated as :

$$\alpha = \arctan\left(\frac{w_T}{u_T}\right) \tag{3.77}$$

Yaw torque

Yaw torque can be given as:

$$N_B = \bar{q}\, S\, b\, C_N \tag{3.78}$$

with a dimensionless yaw torque for the ETH drone :

$$C_N = C_{N_{\delta r}}\,\delta_r + C_{N_{\tilde{r}}}\,\tilde{r} + C_{N_\beta}\,\beta \tag{3.79}$$

and for the MAKO :

$$C_N = C_{N_0} + C_{N_a}\,\delta_a + C_{N_{\tilde{p}}}\,\tilde{p} + C_{N_{\tilde{r}}}\,\tilde{r} + C_{N_\beta}\,\beta \tag{3.80}$$

where $\bar{q}$ is the dynamic pressure, V_T is the total airspeed of the aircraft, ρ is the air density, S is the wing total surface, b is the wing span, and $\bar{c}$ mean aerodynamic wing chord.

3.3.2 Modeling of aerodynamic forces

The position of the aircraft changes with the forces applied to the airframe. The calculation of aerodynamic forces, lift force, drag force, lateral force, and thrust force are given below.

Lift force

The lift force is given in the wind frame as :

$$Z^w = \bar{q}\,S\,C_Z(\alpha) \tag{3.81}$$

where the dimensionless lift coefficient is calculated as:

$$C_Z(\alpha) = C_{Z_0} + C_{Z_\alpha}\alpha \tag{3.82}$$

Drag force

Drag force in the wind frame is calculated as a multiplication of the dynamic pressure, wing surface area, and the drag coefficient as given in Equ. 3.83 :

$$X^w = \bar{q}\,S\,C_X(\alpha, \beta) \tag{3.83}$$

where the dimensionless drag coefficient for ETH drone is given as :

$$C_X(\alpha, \beta) = C_{X_1} + C_{X_\alpha}\alpha + C_{X_{\alpha 2}}\alpha^2 + C_{X_{\beta 2}}\beta^2 \tag{3.84}$$

While for MAKO, the drag force is given by [16] :

$$X^w = \bar{q}\, S\, C_Z(\alpha, \beta) \tag{3.85}$$

and the dimensionless drag coefficient for MAKO [16] :

$$C_X(\alpha) = C_{X_0} + C_{X_k}\, C_Z^2 = C_{X_0} + C_{X_k}\,(C_{Z_0} + C_{Z_\alpha}\alpha)^2 \tag{3.86}$$

Lateral force

The lateral force is given as :

$$Y^w = \bar{q}\, S\, C_Y(\beta) \tag{3.87}$$

where the dimensionless lateral coefficient for ETH drone is given as :

$$C_Y(\beta) = C_{Y_\beta}\beta \tag{3.88}$$

and for MAKO :

$$C_Y(\beta,\, \tilde{p},\, \tilde{r},\, \delta_a) = C_{Y_\beta}\beta + C_{Y_{\tilde{p}}}\,\tilde{p} + C_{Y_{\tilde{r}}}\,\tilde{r} + C_{Y_a}\,\delta_a \tag{3.89}$$

where the coefficients can be seen in Tables 3.6 and 3.7.

Thrust force model

The thrust generated by the propeller can be written as :

$$F_T = \rho n^2 D^4 C_{F_T} \tag{3.90}$$

Table 3.6: Aerodynamic force derivatives for ETH UAV [30]

Parameter	**Value**	**Definition**
C_{Z_0}	1.29×10^{-2}	lift derivative
C_{Z_α}	-3.25	lift derivative
C_{X_1}	-2.12×10^{-2}	drag derivative
C_{X_α}	-2.66×10^{-2}	drag derivative
$C_{X_{\alpha 2}}$	-1.55	drag derivative
$C_{X_{\beta 2}}$	-4.01×10^{-1}	drag derivative
C_{Y_β}	-3.79×10^{-1}	side force derivative

Table 3.7: Aerodynamic force derivatives for MAKO extracted from AVL program at $14m/s$ equilibrium cruise speed[16]

Parameter	**Value**	**Definition**
C_{Z_0}	-8.53×10^{-2}	lift derivative
C_{Z_α}	3.9444	lift derivative
C_{Z_q}	4.8198	lift derivative
C_{Z_e}	1.6558×10^{-2}	lift derivative
C_{X_0}	2.313×10^{-2}	drag derivative
C_{X_k}	1.897×10^{-1}	drag derivative
C_{Y_β}	-2.708×10^{-1}	side force derivative
$C_{Y_{\tilde{p}}}$	1.695×10^{-2}	side force derivative
$C_{Y_{\tilde{r}}}$	5.003×10^{-2}	side force derivative
C_{Y_a}	0.0254×10^{-2}	side force derivative

Table 3.8: Thrust force coefficients for propeller ETH UAV [30]

Parameter	**Value**	**Definition**
$C_{F_{T1}}$	8.42×10^{-2}	thrust derivative
$C_{F_{T2}}$	-1.36×10^{-1}	thrust derivative
$C_{F_{T3}}$	-9.28×10^{-1}	thrust derivative
D	$0.79\,m$	propeller diameter

Table 3.9: Thrust force coefficients for propeller APC SF 9×6 from wind tunnel experiments [15]

Parameter	**Value**	**Definition**
$C_{F_{T1}}$	1.342×10^{-1}	thrust derivative
$C_{F_{T2}}$	-1.975×10^{-1}	thrust derivative
$C_{F_{T_{rpm}}}$	7.048×10^{-6}	thrust derivative
D	$0.228\,m$	propeller diameter

where the dimensionless thrust coefficient for ETH drone is calculated as :

$$C_{F_T} = C_{F_{T1}} + C_{F_{T2}} J + C_{F_{T3}} J^2 \tag{3.91}$$

Here, J is the advance ratio, and for ETH drone, it is given by:

$$J = \frac{V_T}{n\pi D} \tag{3.92}$$

The dimensionless thrust coefficient for MAKO is given by [15] :

$$C_{F_T} = C_{F_{T1}} + C_{F_{T2}} \cdot J' + C_{F_{T_{rpm}}} \cdot n \cdot 60 \tag{3.93}$$

with advance ratio :

$$J' = \frac{V_T}{nD} \tag{3.94}$$

3.3.3 Shortcut to modeling

After the derivation of the aircraft flight kinematics and dynamics for translational and attitude motion, the system of first order differential equations can be summarized as follows :

$$\boxed{\begin{aligned}
\dot{\boldsymbol{x}}_N &= \boldsymbol{C}_B^N \boldsymbol{v}_B \\
\dot{\boldsymbol{v}}_B &= \frac{1}{m}\big[m\boldsymbol{g}_B + \boldsymbol{F}_{t_B} + \boldsymbol{F}_{a_B}\big] - \big[\boldsymbol{\omega}_B^{B/N}\big]^{\times}\boldsymbol{v}_B \\
\dot{q}_0 &= -\frac{1}{2}\boldsymbol{q}_\nu^T\boldsymbol{\omega}_B^{B/N} \\
\dot{\boldsymbol{q}}_\nu &= \frac{1}{2}\Big(\boldsymbol{q}_\nu^{\times} + q_0\boldsymbol{I}_3\Big)\boldsymbol{\omega}_B^{B/N} \\
\boldsymbol{J}\dot{\boldsymbol{\omega}}_B^{B/N} &= \boldsymbol{M}_B - \big[\boldsymbol{\omega}_B^{B/N}\big]^{\times}\boldsymbol{J}\boldsymbol{\omega}_B^{B/N}
\end{aligned}} \tag{3.95}$$

where $\boldsymbol{x}_N \in \mathbb{R}^3$ is the position of the center of mass of UAV in the navigation frame N, $\boldsymbol{v}_B$ is the velocity of the center of mass of UAV in the body frame B, $\boldsymbol{q} = [q_0, \boldsymbol{q}_v^T]^T \in \mathbb{R}^3 \times \mathbb{R}$ is the unit quaternion representing the attitude of the body frame B with respect to navigation frame N expressed in the body frame B, $\boldsymbol{\omega}_B^{B/N}$ is the angular velocity of the body frame B with respect to navigation frame N expressed in the body frame B, $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is the positive definite inertia matrix of the drone, $\boldsymbol{M}_B \in \mathbb{R}^3$ represents the moments acting on the drone, $\boldsymbol{C}_B^N$ is the direction cosine matrix which transforms a vector expressed in the body frame to its equivalent expressed in the navigation frame N, $\boldsymbol{I}_3 \in \mathbb{R}^{3\times3}$ is the identity matrix, $\boldsymbol{F}_{t_B} \in \mathbb{R}^3$ is the thrust force expressed in the body frame and, $\boldsymbol{F}_{a_B} \in \mathbb{R}^3$ are the aerodynamic forces given in the body frame B. The navigation frame is assumed to be a local inertial frame in which Newton's Laws apply. The notation $\boldsymbol{x}^{\times}$ for a vector $x = [x_1 \quad x_2 \quad x_3]^{\mathrm{T}}$ represents the skew-symmetric matrix, as given in Equ. 3.36.

3.3.4 Sensor Models

Accelerometer and gyro measurements are simulated using the angular velocity $\boldsymbol{\omega}_B^{B/N}$, and translational acceleration $\dot{\boldsymbol{v}}_B$ given by the system of equations of a drone summarized in Equ. 3.95 and the specifications of the hardware used in *Apogee Autopilot* of

Paparazzi Autopilot System. The sensor suit simulated is the InvenSense MPU-9250 Nine-axis (Gyro + Accelerometer + Compass) MEMS MotionTracking Device.

The accelerometer and gyro data is simulated as:

$$\boldsymbol{z}_{gyro} = \boldsymbol{k}_{gyro}\boldsymbol{\omega}^{B}_{B/I} + \boldsymbol{\beta}_{gyro} + \boldsymbol{\eta}_{gyro} \tag{3.96}$$

$$\boldsymbol{z}_{acc} = \boldsymbol{k}_{acc}\boldsymbol{\omega}^{B}_{B/I} + \boldsymbol{\beta}_{acc} + \boldsymbol{\eta}_{acc} \tag{3.97}$$

where $\boldsymbol{\beta}$ is the bias, and $\boldsymbol{\eta}$ is the zero mean Gaussian process with $\boldsymbol{\sigma}^2$ variance with values given in Table 3.10.

Table 3.10: Specifications of the sensor suit InvenSense MPU-9250 Nine-axis (Gyro + Accelerometer + Compass) MEMS MotionTracking Device[21]

Measurement	$\boldsymbol{\beta}$	$\boldsymbol{\sigma}$
$\boldsymbol{z}_{acc_x}$	0.142	0.0319
$\boldsymbol{z}_{acc_y}$	-0.3	0.0985
$\boldsymbol{z}_{acc_z}$	0.19	0.049
$\boldsymbol{z}_{gyro_x}$	-1.55	0.0825
$\boldsymbol{z}_{gyro_y}$	-1.13	0.1673
$\boldsymbol{z}_{gyro_z}$	-1.7	0.2214

3.3.5 Fault Models

Probable faults in the control surfaces can be grouped under two main categories [103]:

- A total control loss of the control surface actuators. The actuator does not respond to the control signals at all. Lock-in-place, hard-over and floating around trim are such failures (see Fig. 3-10).;
- A partial control loss of the control surface actuators. The actuator does respond to the control signals but does so in an abnormal way. Loss of effectiveness failure is shown in Fig. 3-10.

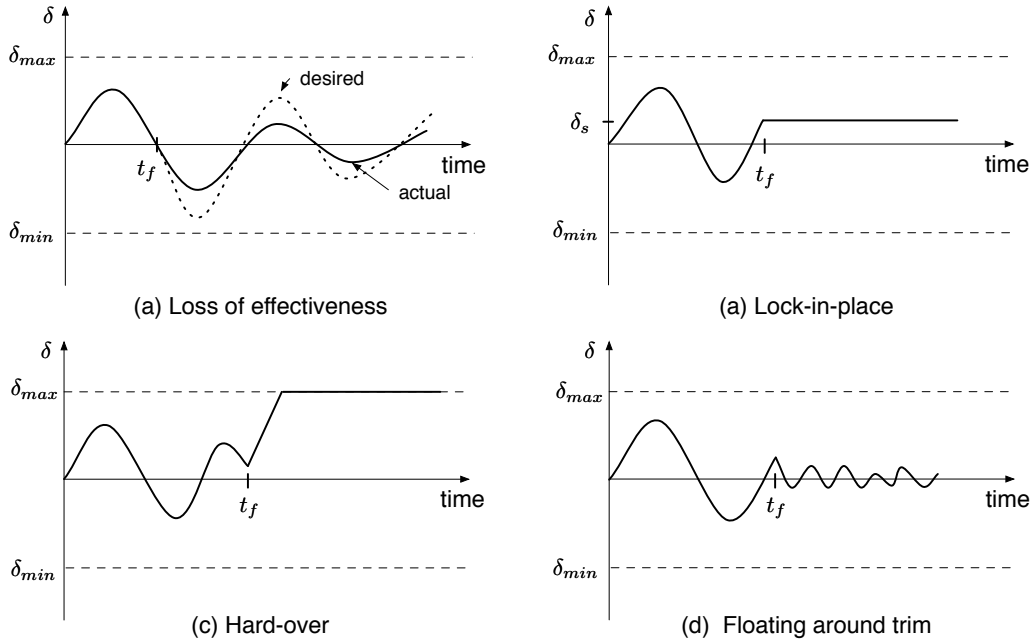

Figure 3-10: Probable actuator faults [30]. (a) Loss of effectiveness: The actuator does respond to the control signals but does so in an abnormal way such as low actuation level or low response time. (b) Lock-in-place: A total control loss of the control surface actuators. The actuator freezes at a particular position. (c) Hard-over: A total control loss of the control surface actuators. The actuator freezes at the minimum or maximum position limit. (d) A total control loss of the control surface actuators. Actuator does not contribute to the control authority.

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:

$$\boldsymbol{u}(t) = \boldsymbol{E}\boldsymbol{u}_c + \boldsymbol{u}_f \tag{3.98}$$

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

3.4 Conclusion

In this chapter, equations of motion of an aircraft are given. The motion of an aircraft usually involves both translation and rotation. Here, equations for translational and rotational motion are discussed separately and in detail. After the equations are derived for a generic aircraft, the calculation of forces and moments, which are specific to an individual drone, are presented (aerodynamic force derivatives and stability derivatives are specific for each drone). For two different drone examples, a drone from ETH Zurich and MAKO (used in ENAC UAV LAB), and the calculation of forces and moments are given. Those forces and moments are inputs to the dynamic equations of motion.

The models derived here are not used in detection and diagnosis algorithms. The detection and diagnosis algorithms used in this thesis use data only. The need for data is the driving factor to simulate the drone motion, thus, data is simulated using the MAKO drone model and specifications of IMU *InvenSense MPU-9250 Nine-axis* used in *Paparazzi Autopilot Apogee* onboard.

In this thesis, the diagnosis is implemented on two types of data: first, simulated data, and second, flight data. If the reader is not interested in detection and diagnosis via simulated data, it is not essential to read this chapter since it explains the models that are used to simulate measurements. Despite this, having background information on the physics of the system may help to understand the features (translational acceleration and angular velocities) used in both model-based and data-driven fault diagnosis.

Since the data is generated and now available for implementation, the methodology used in this thesis to diagnose faults is discussed in the next chapter. In this thesis, *Support Vector Machines* (SVM) are used as a classification method to diagnose faults on board a drone. Since SVM is a machine learning method, an introduction to machine learning methods is provided from an implementation perspective. We start with an introduction to the terminology for machine learning methods, and provide a general explanation to prepare the inputs to the machine learning algorithm. Next,

many application details are discussed to assist the user to apply these methods to their own problems. Finally, SVM is introduced, plus the three phases encountered during its implementation — *training*, *tuning* and *evaluation* — are explained.

Chapter 4

Methodology

In this chapter, an introduction to machine learning methods is presented. Machine learning methods is discussed with a focus on applications. Finally, a supervised learning method, Support Vector Machines (SVM), is explained as it is the method used for classification in this thesis.

4.1 Machine Learning

The aim of machine learning methods is to train a model with a given data set in order to predict the output values corresponding to a new input. According to Arthur Samuel, who coined the term machine learning, "Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed". Another definition offered by Tom Mitchell in 1998, relatively new compared to the definition by Samuel in 1959, is stated as: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E".

4.1.1 Introduction

Machine learning methods are covered under two main categories: supervised and unsupervised learning, as shown in Fig. 4-1.

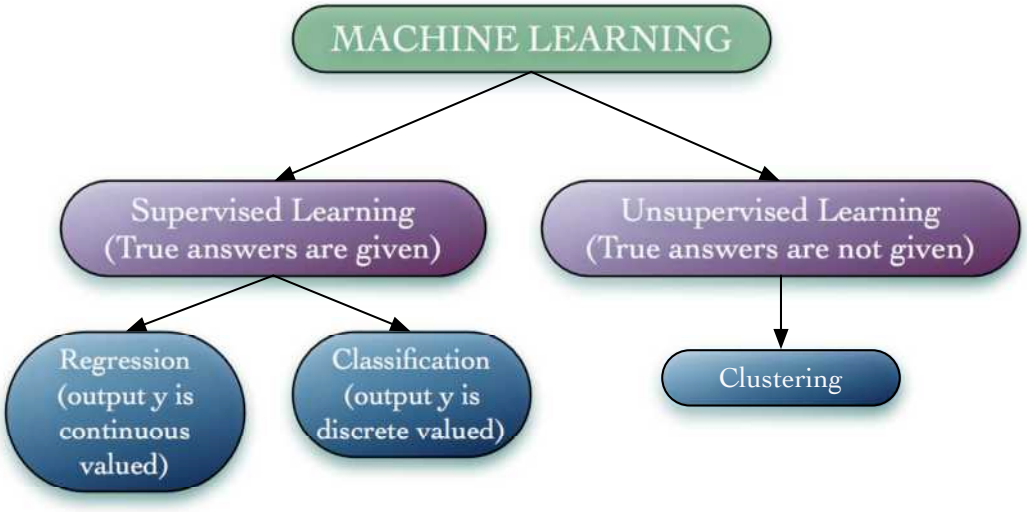

Figure 4-1: Common machine learning methodologies

In supervised learning, *"right answers"* are available for each training input in the data set. Supervised learning methods are made up of two phases; the learning and the prediction, as shown in Fig. 4-2. The first phase is comprised of learning from the available data to understand how the system behaves. In the second phase, the idea is to predict what the output of the system for a given input will be, depending on the knowledge about the system that is gained via the learning phase.

The most common types of supervised learning are the regression and classification problems. Since these are both supervised learning types, the *"right answer"* (right values of the output y) for each example is assumed to be known.

In unsupervised learning, the *"right answers"* corresponding to the training input are not available. There are other machine learning algorithms, such as reinforcement learning, that are a combination of supervised and unsupervised learning.

4.1.2 Terminology

A basic introduction to frequently used terms in machine learning is provided in this section. Table 4.1 shows representations of commonly used variables in machine learning. Although representations may differ from one reference to the other, we follow one coherent representation throughout this paper (see Table 4.1).

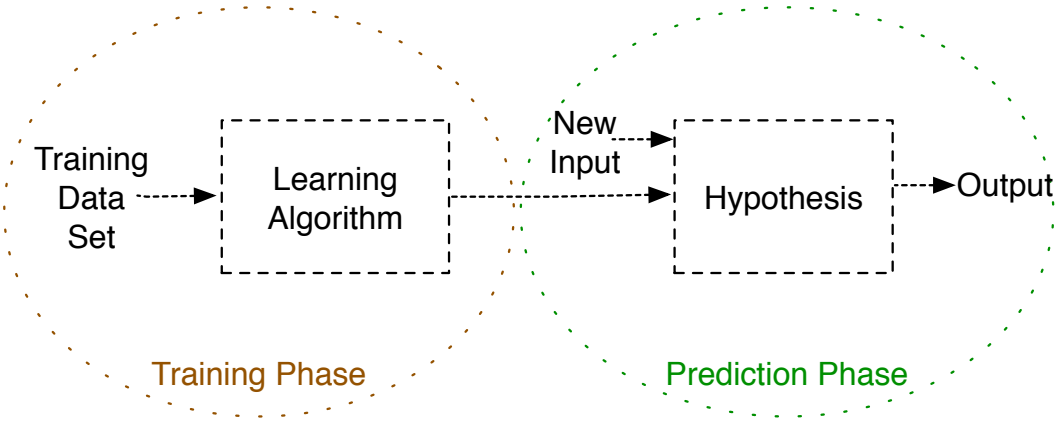

Figure 4-2: Supervised learning basics

Table 4.1: Machine learning terminology

x	input variable or feature
y	output variable or target variable
$\mathbf{X}$	input variable vector or feature vector
$\mathbf{y}$	output variable vector or target variable vector
m	number of training examples
n	number of features
i	index of training examples
j	index of features
$\mathbf{x}_j^{(i)}$	i^{th} training example for feature j
$\mathbf{y}^{(i)}$	i^{th} training output
$\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x})$	hypothesis function (model)
$\boldsymbol{\theta}$	parameter set
$\mathbf{J}(\boldsymbol{\theta})$	cost function

Table 4.2: Training set $(\mathbf{x}, \mathbf{y})$ of housing prices — one-feature example

training example index (i)	**Size in** $feet^2$ **(x)**	**Price in** $1000\$s$ **(y)**
1	2104	460
2	1416	232
3	1534	315
4	852	178
$\vdots$	$x^{(i)}$	$y^{(i)}$
m	$x^{(m)}$	$y^{(m)}$

Many of the machine learning tools, such as *MATLAB Statistics and Machine Learning Toolbox* or *Tensorflow*, have default settings which make them easier for beginners to use. Although they would most probably not achieve optimized results (as would be the case in the hands of an experienced user), a beginner can easily start using these tools by sending input and output vectors as arguments to the machine learning functions.

The configurations of input and output vectors to feed the learning algorithms may change from one to the other, but there is still a common convention that would be compatible with many. In this representation, each instance is given in different rows of the input vector. An example of predicting housing prices taken from Ref. [68] is provided to show the way in which to constitute the input and output matrix. Table 4.2 uses a supervised learning regression example to show input and output vector representations. The aim of the problem in this example is to predict the prices of houses for a given the surface area. For that, known examples of houses with a surface area and corresponding price are given. Since the aim of the problem is to predict the price of a house given the surface area, the price of the house is the output of the problem. To predict the price, the available information that is known to effect the price of the house is the surface area of the house, making it the input variable. So, in Table 4.2, each row corresponds to a different house, the second column (surface area of house) constitutes the input vector and the last row (price in $1000s) corresponds to the output vector.

In this problem, there is an assumption that the price of a house is only dependent

Table 4.3: Training set $(\mathbf{x}, \mathbf{y})$ of housing prices - multi-feature example

training example index (i)	**Size in** $feet^2$ $(\mathbf{x}_1^{(i)})$	**Number of bedrooms** $(\mathbf{x}_2^{(i)})$	**Feature j** $(\mathbf{x}_j^{(i)})$	**Feature n** $(\mathbf{x}_n^{(i)})$	**Price in** $1000\$s$ $(\mathbf{y})$
1	2104	5	$\mathbf{x}_j^{(1)}$	$\mathbf{x}_n^{(1)}$	460
2	1416	3	$\mathbf{x}_j^{(2)}$	$\mathbf{x}_n^{(2)}$	232
3	1534	3	$\mathbf{x}_j^{(3)}$	$\mathbf{x}_n^{(3)}$	315
4	852	2	$\mathbf{x}_j^{(4)}$	$\mathbf{x}_n^{(4)}$	178
$\vdots$	$\mathbf{x}_1^{(i)}$	$\mathbf{x}_2^{(i)}$	$\mathbf{x}_j^{(i)}$	$\mathbf{x}_n^{(i)}$	$\mathbf{y}^{(i)}$
m	$\mathbf{x}_1^{(m)}$	$\mathbf{x}_2^{(m)}$	$\mathbf{x}_2 j^{(m)}$	$\mathbf{x}_n^{(m)}$	$\mathbf{y}^{(m)}$

on the surface area which does not hold in the real case. In real problems, it is more common that the output would not depend on only one variable but on many variables. For this reason, the input vector in a realistic example would more likely be an input matrix having different features in its columns, as given in Table 4.3. Here, in this example, the output is still the price of the house, but now the features which correspond to each column of an input matrix (or sometimes called the feature matrix) are the surface area, the number of bedrooms, and can be enlarged up to a chosen number of features. It should be kept in mind that the selection of features that would lead to a better prediction of the output is a challenging problem, and would usually require some considerable experience on the system of interest.

4.1.3 Steps towards the learning machine

Visualizing the data

A preliminary investigation of data may give the designer an idea about the ways in which to tackle a problem, since it is the designer who selects the features, the model and the type of cost function. It is true that the algorithms are optimizing some parameters of the model and the cost function but the structure itself is still usually supplied by a human.

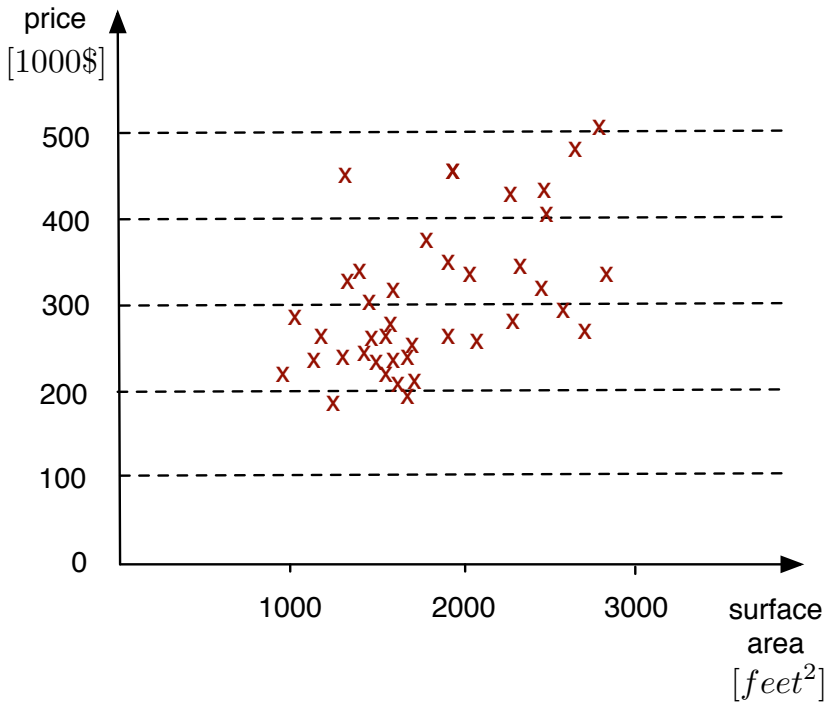

Figure 4-3: Regression example - Housing prices as a function of surface area of the house [69]

Fig. 4-3 shows the housing price depending on the surface area corresponding to the data given in Table 4.2. In this example, house prices are the output (target) variable given in y-axis and size (surface area in $feet^2$) is the feature (input) variable given in x-axis. Since the output y is the price which is a continuous value (thus not a label representing a class), this problem is called a regression problem. Plotting the given data, shows that the problem could be modeled by a linear model (a linear line to fit the data given). Then, linear regression can be applied.

Since an example of a regression problem has been given above, we now take a preliminary look at the classification problem. Fig. 4-4 presents a classification problem, since the aim is to distinguish the class that a new input instance will belong to. Here, the vertical dimension is not the output, as in the previous example of linear regression, but is another feature (x_2 in this example).

The information about the output is represented by the colors of the samples in the feature space. Fig. 4-5 shows the output vs. inputs of a classification problem.

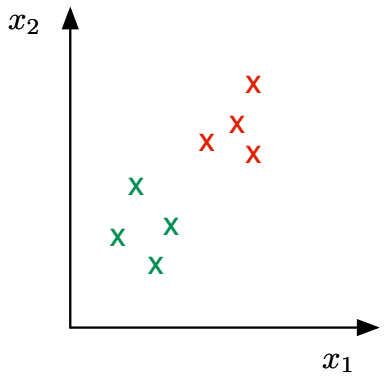

Figure 4-4: Classification example

This is simply to show the difference between regression and classification problem figures. Usually in regression problems the y-axis represents the output. Here, there is a direct analogy by representing a classification problem as a regression problem, with the x-axis representing the input and the y-axis representing the output. However, usually in classification problems, the value of y (which class it corresponds to) is represented by discrete values representing the different classes, as shown in Fig. 4-4.

By plotting the samples, as in this example, it seems that logistic regression could yield satisfactory results, since the data seems to be separated by a linear decision boundary. A decision boundary is a curve that separates the training data examples which belong to different classes.

It is possible to choose the model structure that would represent the problem satisfactorily via visualizing the data set when the number of features is not large. With an increasing number of features, determining the degree of the polynomial via visualizing the data to represent the input/output relationship could be more complicated. In this case, the user should refer to model selection techniques rather than the insights gained by data visualization.

Feature Mapping

Depending on the results from visualizing data, it may be necessary to map the features, since a straightforward implementation of linear/logistic regression may result

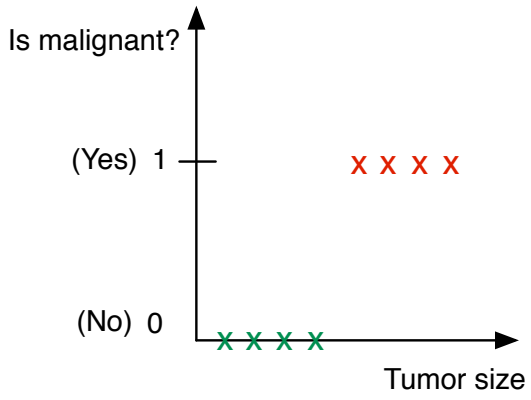

Figure 4-5: Classification example - output vs input

in linear model and linear decision boundary, respectively. For the cases in which a linear model will not be satisfactory to describe the training data, feature mapping should be applied.

The housing price prediction problem can be revisited here to provide an example of feature mapping. The data given in Table 4.2 and plotted in Fig. 4-3 is a regression problem with one feature (surface area of house). To fit more accurately to the training data given, new features can be artificially added to the system as below, where the new features will be functions of the original feature:

$$\begin{aligned} h_\theta(\boldsymbol{x}) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\ &= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3 \end{aligned} \tag{4.1}$$

where:

$$\begin{aligned} x_1 &= size \\ x_2 &= size^2 \\ x_3 &= size^3 \end{aligned} \tag{4.2}$$

By changing the model via adding features, it is now possible to fit a nonlinear

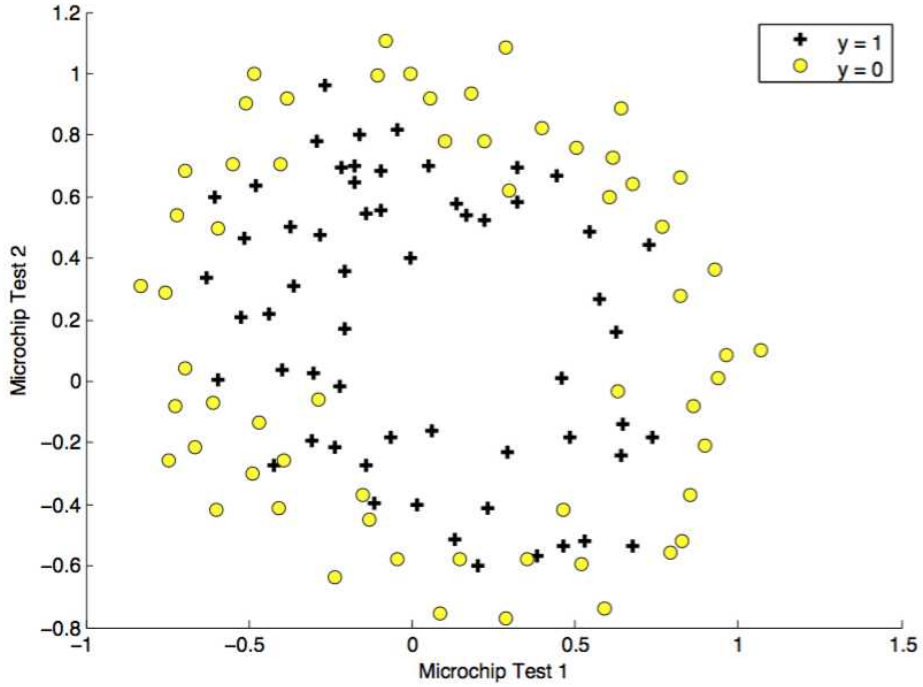

Figure 4-6: Classification example of complex decision boundaries [69]

curve to the training data, as given in Fig. 4-13.

Another example that requires a nonlinear model to fit the data set is the microchip assessment problem. This example is taken from the lecture notes of a machine learning course by Andrew Ng [68]. This is a two-class classification problem, where one class of microchips is faulty while the other class is functional. It is seen from the Fig. 4-6 that a linear decision boundary will not appropriately serve to classify the data. So, in order to apply logistic regression for this problem, feature mapping is necessary. An example of a mapped feature vector that could provide a satisfactory decision boundary to classify the two classes could be given as:

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 \cdots & x_1x_2^5 & x_2^6 \end{bmatrix}^T \tag{4.3}$$

Mapping of the features could be done in various ways. We now take another example of a binary (two-class) classification problem with 100 features, which needs a nonlinear decision boundary to achieve a satisfactory classification. One way to map

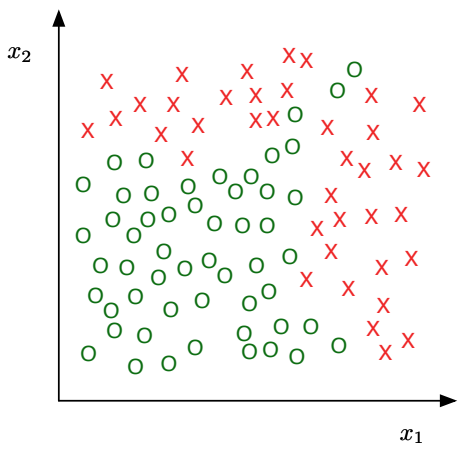

Figure 4-7: Classification example of complex decision boundaries

the feature set could be such that only the quadratic terms are taken into account, which would result in a new feature set such as:

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} x_1^2 & x_1x_2 & x_1x_3 & \cdots & x_1x_{100} & x_2^2 & x_2x_3 & \cdots & x_3^2 & x_3x_4\cdots \end{bmatrix}^T \tag{4.4}$$

with a complexity $O(n^2) \sim \frac{n^2}{2}$.

If 3$^{\text{rd}}$ order terms are considered, the new feature set becomes:

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} x_1^3 & x_1x_2x_3 & x_1^2x_2 & \cdots & x_{11}x_{13}x_{17} & \cdots \end{bmatrix}^T \tag{4.5}$$

It is important to point out that, with a higher dimensional feature vector, the model will tend to fit more accurately with the training set but may generalize poorly to new input. This is called overfitting, which is a common problem for many machine learning applications. Further explanation about overfitting and ways to overcome it will be offered in the regularization section. Increasing the dimensionality of the features, the model might also become computationally more expensive.

To fit a model or decision boundary of a circular or elliptical shape, only a subset

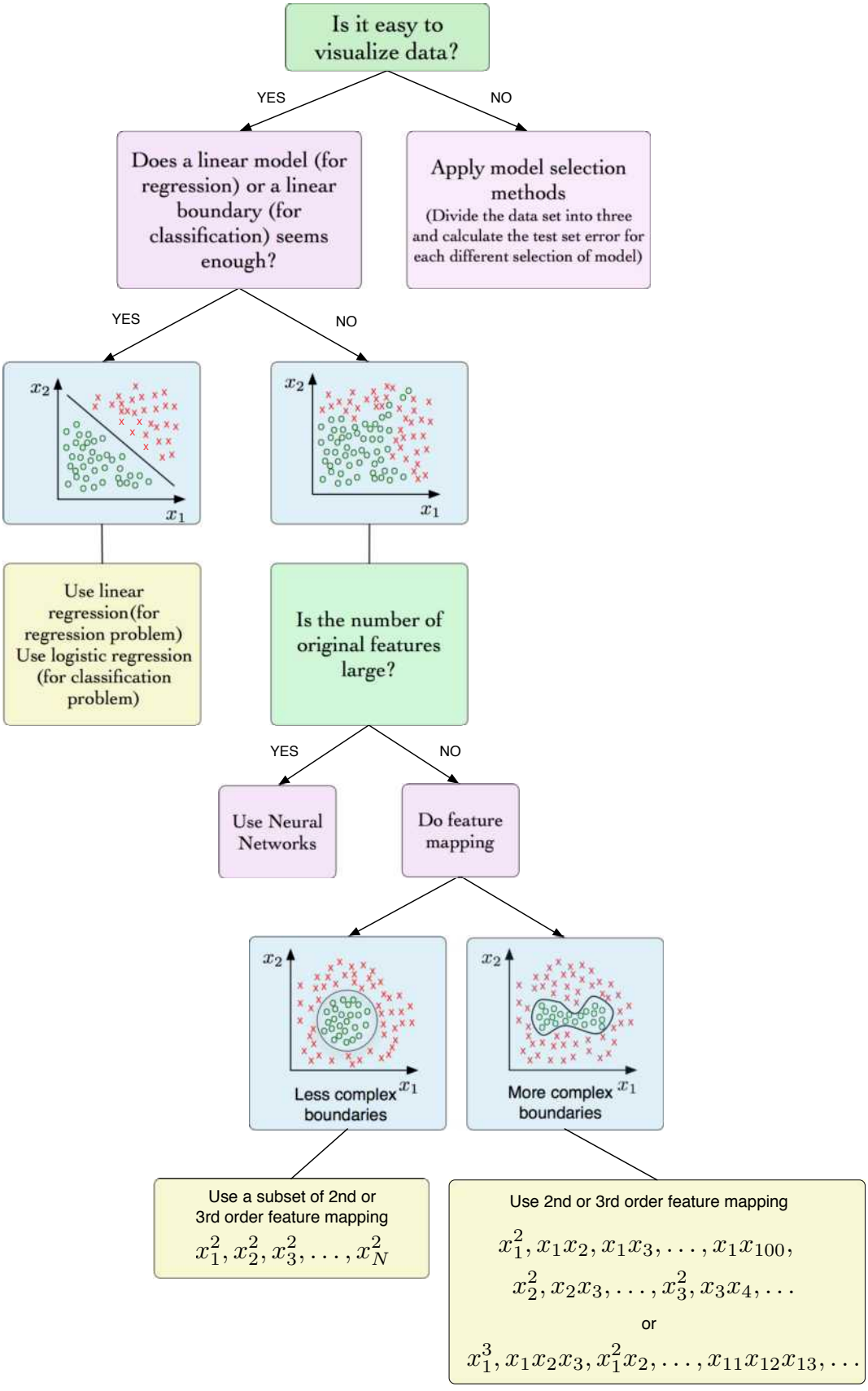

Figure 4-8: Simplified guide for feature mapping

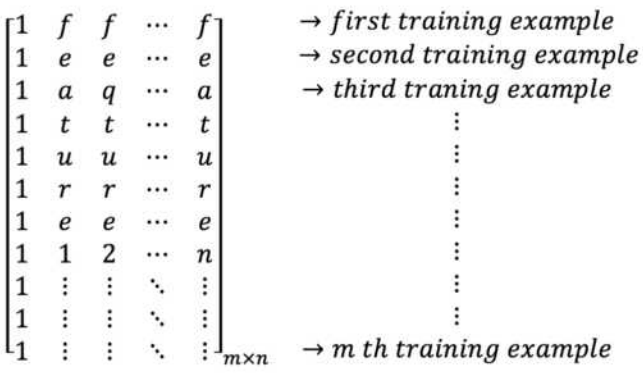

Figure 4-9: Adding an artificial feature of 1s.

of features can be considered, such as:

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 & \cdots & x_N^2 \end{bmatrix}^T \tag{4.6}$$

Finally, when the number of features are small, but cannot fit complex models or boundaries, such as in Fig. 4-7, then the addition of higher terms such as those given in Equ. 4.4 or the use of Neural Networks should be considered. A scheme to help select the strategy for mapping is given in Fig. 4-8.

Vectorized form of the problem

The model can be written in a vectorized form in order to accommodate the larger number of features in a more compact form. To write in compact form, an extra feature $\boldsymbol{x}_0 = 1$ is added to the feature vector, such as:

$$\boldsymbol{x} = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \tag{4.7}$$

Fig. 4-9 shows the addition of the artificial feature in the feature matrix.

Also, parameters of the model are given in vectorial form, such as:

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_n \end{bmatrix}^T \tag{4.8}$$

So, for the linear regression problem with a single feature, the model is given as:

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x_1 \tag{4.9}$$

and the model for n features:

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{4.10}$$

Introducing the vectorial forms of $\boldsymbol{x}$ and $\boldsymbol{\theta}$ enables the model to be written in a compact generic form, such as:

$$h_{\boldsymbol{\theta}}(x) = \boldsymbol{\theta}^{\intercal}\boldsymbol{x} \tag{4.11}$$

Cost Function

To calculate $\boldsymbol{\theta}$ that will fit a better model with the training data, optimization algorithms are used to minimize the cost function $J(\boldsymbol{\theta})$. Usually, the cost function and its gradient are given to the optimization algorithm as an input. Here, widely-used cost functions for linear regression (regression problem) and logistic regression (classification problem) are given.

The cost function for linear regression can be given as:

$$J(\boldsymbol{\theta}) = \frac{1}{2m}\sum_{i=1}^{m}\Big(h_{\boldsymbol{\theta}}(x^{(i)}) - (y^{(i)})\Big)^2 \tag{4.12}$$

and the cost function for logistic regression (classification) is given for a two-class classification problem (e.g $y \in \{0, 1\}$) as:

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m}\Big[-y^{(i)}log(h_{\boldsymbol{\theta}}(x^{(i)})) - (1-(y^{(i)}))log(1-h_{\boldsymbol{\theta}}(x^{(i)}))\Big] \tag{4.13}$$

Selecting the model

With a larger set of features, it may be difficult to have a sense of which model would be a better fit for a particular problem. In these cases, model selection techniques could be applied to decide which model will yield better accuracy.

To explain the procedure, first assume a model structure is already selected or given. In such a problem, during the training process, training data is used to calculate the best model parameter vector $\boldsymbol{\theta}$ that would minimize the objective function, $J_{tranining}(\boldsymbol{\theta})$. The the trained model is then assessed with a new independent data set J_{test} to evaluate the overfitting biases favoring the training data and avoid an over-optimistic view of the abilities of the learning algorithm.

In the model selection technique, each model is represented by a number d as shown below:

$$
\begin{aligned}
d=1, \quad & h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x && \longrightarrow \theta^{(1)} && \longrightarrow J_{cv}(\theta^{(1)}) \\
d=2, \quad & h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 && \longrightarrow \theta^{(2)} && \longrightarrow J_{cv}(\theta^{(2)}) \\
d=3, \quad & h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 && \longrightarrow \theta^{(3)} && \longrightarrow J_{cv}(\theta^{(3)}) \\
& \vdots \\
d=k, \quad & h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_k x^k && \longrightarrow \theta^{(k)} && \longrightarrow J_{cv}(\theta^{(k)})
\end{aligned}
$$

In the previous problem of a given model structure, only the model parameter vector $\boldsymbol{\theta}$ is calculated to minimize the cost function $J_{training}(\boldsymbol{\theta})$. Now, since the best model is not known, it is also necessary to find the best d to minimize the cost function. Since now there are two parameters to fit, $\boldsymbol{\theta}$ and d, three different sets of data are necessary for training, tuning and evaluation. Similar to the case where training and evaluating the model with the same data (for a given model) may lead to a biased evaluation, the added necessity to also select (or tune) the model also requires an extra set of data. These three separate sets of data are called training, cross-validation and test sets.

So, the idea is to have different subsets of data to train the model (calculating $\boldsymbol{\theta}$), to select the model structure and to evaluate the selected model structure and its corresponding parameter set. In order to do that, three distinct data sets are needed, as shown in Fig. 4-10, to train parameters $\boldsymbol{\theta}$, to select the model structure d and to associate an error value with the chosen model. Then, the model structure d which yields the smallest $J_{cv}(\boldsymbol{\theta}^{(d)})$ once minimized is selected and its generalized error is calculated with the test set $J_{test}(\boldsymbol{\theta}^{(d)})$.

Fig. 4-10 summarizes the steps to be followed, which are explained in more detail as:

1. Train the model for each model structure (d = 1,2,3 .. k) by using the training set (60% of all data, randomly selected) which will output the parameters $\boldsymbol{\theta}^{(d)}$ for the corresponding model (d). This phase is explained in more detail in the section *Calculating parameters* $\boldsymbol{\theta}$, however briefly, the general idea is to calculate the $\boldsymbol{\theta}$ that will minimize the cost function $J_{training}$ given as:

$$J_{training}(\boldsymbol{\theta}) = \frac{1}{2m_{training}} \sum_{i=1}^{m_{training}} \left(h_{\boldsymbol{\theta}}(x_{training}^{(i)}) - (y_{training}^{(i)}) \right)^2 \tag{4.14}$$

2. Then, by using the cross validation data set (20% of all data, randomly selected) calculate cross validation error for each model and select the model with the smallest cross validation error:

$$J_{cv}(\boldsymbol{\theta}) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_{\boldsymbol{\theta}}(x_{cv}^{(i)}) - (y_{cv}^{(i)}) \right)^2 \tag{4.15}$$

3. Then, by using the test set (20% of all data, randomly selected) the generalized error is estimated for the selected model:

$$J_{test}(\boldsymbol{\theta}) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_{\boldsymbol{\theta}}(x_{test}^{(i)}) - (y_{test}^{(i)}) \right)^2 \tag{4.16}$$

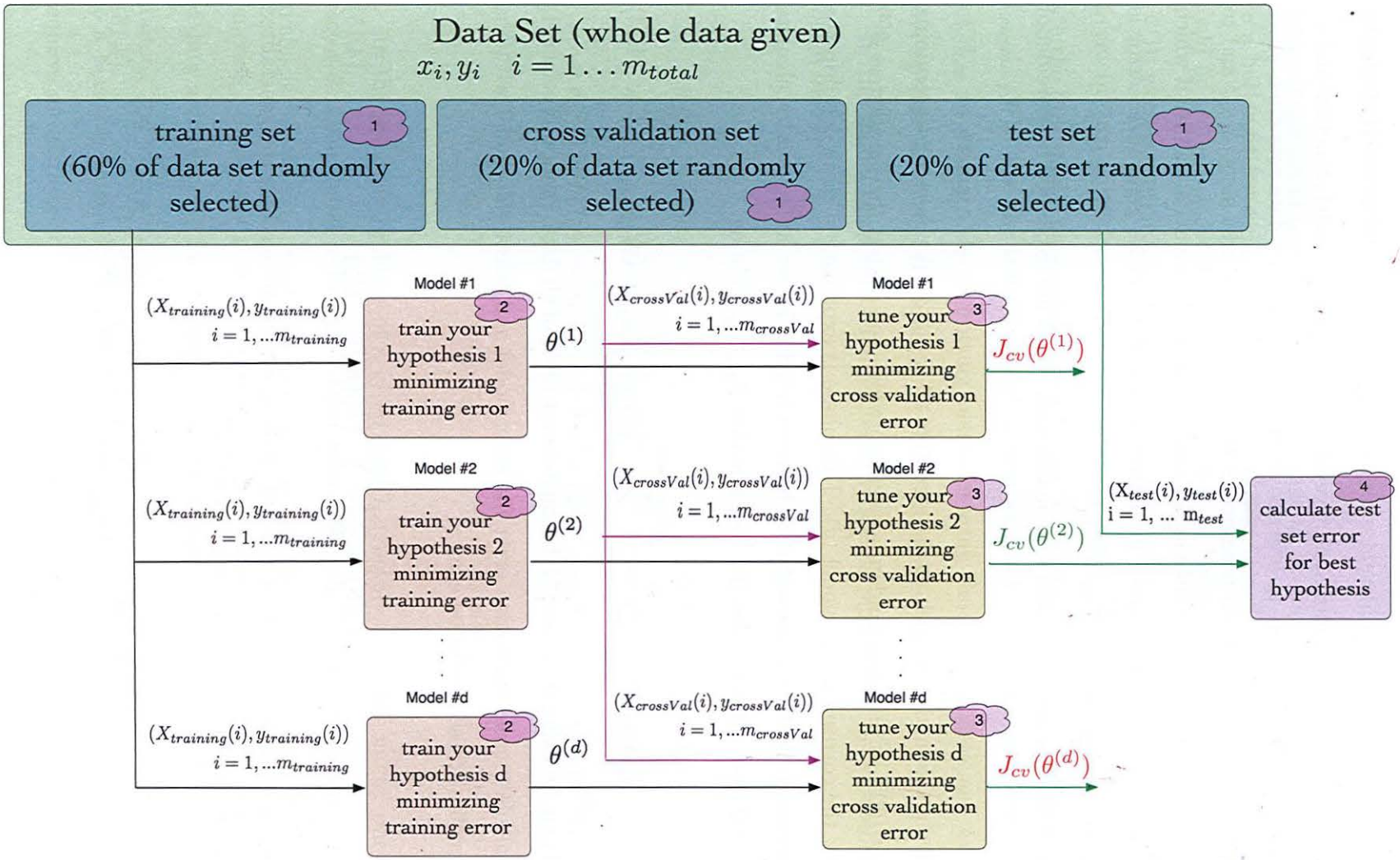

Figure 4-10: Model selection guide

Normalization (Scaling)

The next step is to normalize the data in order to constrain the values of features to change within the same order of magnitude. This helps to improve the convergence rate of the optimization algorithm during the calculation of the model parameters $\boldsymbol{\theta}$. Although there are different methods for normalization, a common method is to use this formula:

$$\bar{x}_j = \frac{x_j - \mu_j}{s_j} \tag{4.17}$$

where the mean μ_j and range s_j are given by:

$$\begin{aligned} \mu_j &= \frac{\sum_{i=1}^{m} x_j^i}{m} \\ s_j &= max(x_j) - min(x_j) \end{aligned} \tag{4.18}$$

An important point is to keep values μ_j and s_j (or standard deviation in the cases it is preferred instead of range s_j) which are calculated during training. These values are saved since they will also be used to scale the new inputs during the prediction phase.

Training and evaluation of the Classifier

Here, it is assumed that we have selected one model and we train the parameters of only this selected model. If multiple models are to be evaluated, this training phase will be applied to all different models. Then, the selection of the best model should be done, as already explained under *Selecting the model* section.

Before the training phase, the data should be split into two parts: training and test sets. Then, the parameters $\boldsymbol{\theta}$ are learnt using the training data. Finally, the trained classifier is evaluated by calculating the test set error using the test set. Those steps are given in more detail as follows:

1. Split the data set to two parts (30% for the test set and 70% for the training

set). Note that if the data set is randomly ordered, the first 70% can be used for training and the rest for testing, but if the data is not randomly ordered (data is time dependent or if there is any correlation), the percentage of the data set for traing should be randomly selected between 30% and 70%;

2. Allow the model to learn the parameters $\boldsymbol{\theta}$ from the new training set (70% of the whole data set). This step is further explained under *Calculating parameters* $\boldsymbol{\theta}$ section;

3. Calculate the test set error by using the test set (30% of all data) with the parameters learnt by using the training set (70% of all data).
   For linear regression, the test set error is the cost function evaluated by the test set $(x_{test}(i), y_{test}(i))$ pairs and the parameters trained by using the training set $(x_{train}(i), y_{train}(i))$ and can be given as:

$$J_{test}(\boldsymbol{\theta}) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_{\boldsymbol{\theta}}(x_{test}^{(i)}) - (y_{test}^{(i)}) \right)^2 \tag{4.19}$$

   For logistic regression, the test set error is given as:

$$J_{test}(\boldsymbol{\theta}) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y_{test}^{(i)} \, log(\, h_{\boldsymbol{\theta}}(\, x_{test}^{(i)})) + (1 - (y_{test}^{(i)})) \, log(\, h_{\boldsymbol{\theta}}(\, x_{test}^{(i)})) \right] \tag{4.20}$$

   Sometimes the misclassification error (between 0 and 1), given in Equ. 4.21, is used instead of Equ. 4.20:

$$test\ error = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left(err(\, h_{\boldsymbol{\theta}}(\, x_{test}^{(i)}), (y_{test}^{(i)}) \right) \tag{4.21}$$

where the error function is given as:

$$err(h_{theta}(x), y) = \begin{cases} 1 & if\ h_{\boldsymbol{\theta}} \geq 0.5,\ y = 0\ or\ if\ h_{\boldsymbol{\theta}} < 0.5,\ y = 1 \\ 0 & otherwise \end{cases} \tag{4.22}$$

Fig. 4-11 visualizes and summarizes the procedure explained above.

Calculating parameters $\boldsymbol{\theta}$

The cost function is minimized to calculate the best $\boldsymbol{\theta}$ with the use of optimization algorithms. To fit the parameters $\boldsymbol{\theta}$, we will try to find those that minimize the cost function $J(\boldsymbol{\theta})$:

$$\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{4.23}$$

For the case of a simplified example with one feature, the minimization problem can be written as:

$$\min_{\theta_0, \theta_1} J(\theta_0, \theta_1) \tag{4.24}$$

To achieve this minimization problem, there are a variety of algorithms available. A classical approach is the gradient descent method. Others include but are not limited to, conjugate gradient, BFGS (Broyden-Fletcher-Goldfarb-Shannon) algorithm, and L-BFGS (Limited-memory BFGS). They usually require the cost function, its gradient and the Hessian approximation in order to calculate the best $\boldsymbol{\theta}$ values.

Gradient Descent

Gradient descent will be presented for one feature problem due to its simplicity. Here, a linear regression problem is considered, where its hypothesis (model) and cost function is given in Equ. 4.25:

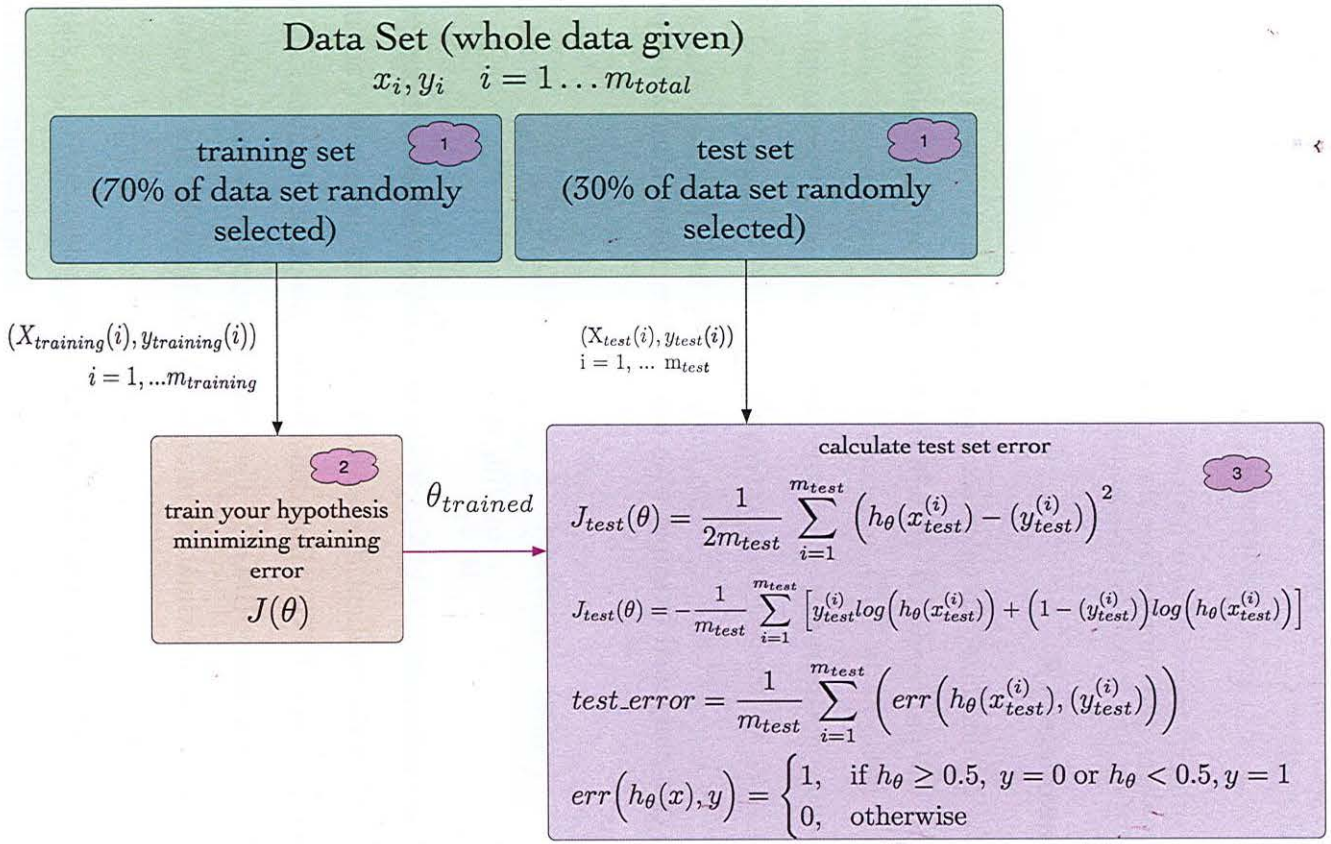

Figure 4-11: Training and evaluating the classifier

$$h_{\boldsymbol{\theta}}(x) = \theta_0 + \theta_1 x_1$$
$$J(\theta_1, \theta_2) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_{\boldsymbol{\theta}}(x^{(i)}) - (y^{(i)}) \right)^2 \tag{4.25}$$

The gradient descent algorithm for one feature is given as :

Algorithm 1 Gradient Descent for one feature only

```
function GradDesOneFeat(X, y, theta_init, alpha, num_iters)
        ▷ Inputs: X - training inputs, y - training outputs, theta - parameters,
alpha - learning rate, num_iters - number of iterations (termination condition)
    theta0 = theta(1))
    theta1 = theta(2))
    m = length(y)                                   ▷ Number of training examples
    for j = 1 to iter_num do                                  ▷ Do until satisfied
        for j = 1 to m do                          ▷ Do for all training examples
            initialize each grad to zero
            grad0 ← grad0 + (costFunc(x) − y)
            grad1 ← grad1 + (costFunc(x) − y) ∗ x
        end for
        theta0 ← theta0 − alpha ∗ grad0
        theta1 ← theta1 − alpha ∗ grad1
    end for
end function
```

The conventional approach is to assign an initial value to $\boldsymbol{\theta}$ is to set $\theta_{init} = 0$. It is important to know that for different initial values, $\boldsymbol{\theta}$ might converge to different values (local minima but not the global one), as in Fig. 4-12. However, for linear regression, it is shown that the cost function shown above is convex, meaning that it has only one global minima with no other local minima. An example of the cost function converging to different solutions depending on initialization is given in Fig.4-12. A slightly different choice of θ_{init} might lead to a convergence to the local minima, where $J(\boldsymbol{\theta})$ is smaller compared to local changes in $\boldsymbol{\theta}$, but $J(\boldsymbol{\theta})$ is still not at its global minimum.

Learning rate, α, gives the gradient descent its step size, so a larger value means a faster convergence. However, note that for a larger alpha, the solution might not converge or even diverge, while too small values might lead the optimization to con-

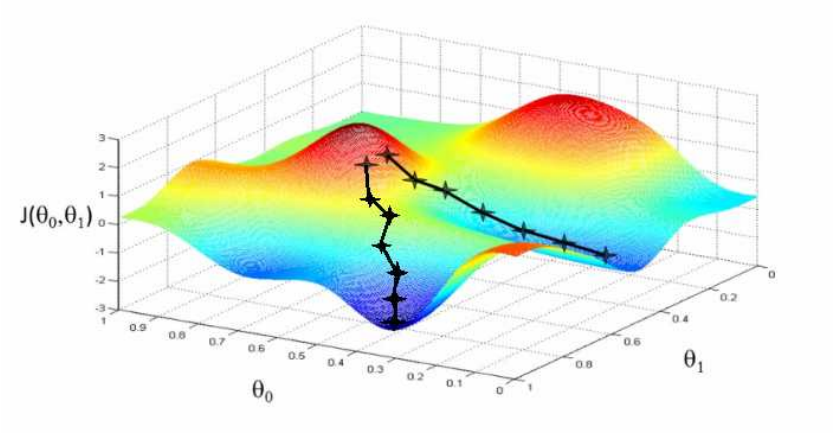

Figure 4-12: Gradient descent convergence dependance on θ_{init}. Two different but close choices of θ_{init} might converge to local or global minima [69].

verge very slowly. A set of α should be tried and evaluated by their performances. Armigo-Goldstein rule can be used to set α.

An important point in the algorithm is to update the $\boldsymbol{\theta}$ simultaneously. Thus, during one iteration, the same $\boldsymbol{\theta}$ values should be substituted into $h_\theta(x)$, in order to calculate the next values of θ_0, θ_1.

Overfitting

Overfitting is common for complicated models or decision boundaries with high order polynomial terms such as:

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 \cdots & x_1x_2^5 & x_2^6 \end{bmatrix}^T \tag{4.26}$$

If the model is suffering from overfitting, the trained model might end up fitting well to the training set, but may fail to generalize to the new data during the prediction phase. This means that the training set error is not a good predictor of how well the model would predict on new examples.

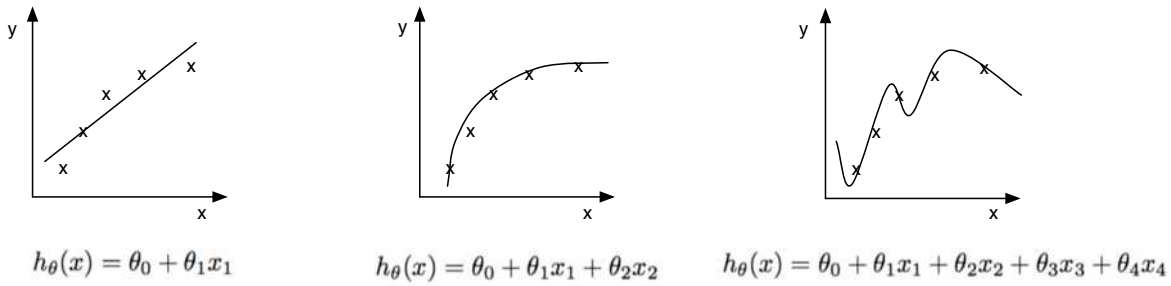

Figure 4-13: Underfitting, satisfactory or overfitting model examples (left to right)

Fig. 4-13 shows three model examples plotted with the data that they are trained with. From left to right, these three regression problem models are underfitted, satisfactory or overfitted. But for problems with more features it might not be that obvious. Sometimes it may not even be possible to plot the model to realize if it is overfit or not. In such cases, methods to detect overfitting should be used. To avoid overfitting, a regularization parameter is used, as explained in the next section.

Regularization

Regularization is practiced in order to avoid overfitting. In regularization, a term is added to the cost function in order to penalize parameters $\vec{\boldsymbol{\theta}}$ for being too large (i.e., to make $\vec{\boldsymbol{\theta}}$ as small as possible), except for θ_0 which is not penalized by convention. The cost function for linear regression with the added term for regularization is given as:

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - (y^{(i)})\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right] \tag{4.27}$$

For logistic regression the regularized cost function can be written as (for two-class e.g $y \in \{0, 1\}$):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\Big[-y^{(i)}log(h_\theta(x^{(i)})) - (1-(y^{(i)}))log(1-h_\theta(x^{(i)}))\Big] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{4.28}$$

Here, the challenge is to find the appropriate value for λ which serves as a weight between the original part of the cost function and the second part, which penalizes $\boldsymbol{\theta}$ for being too large. So, it tends to decrease the values of θ. But if this lambda is too large, then the model converges to:

$$h_\theta(x) = \theta_0 \tag{4.29}$$

since all terms are penalized except θ_0 (which is not penalized by convention), such that:

$$h_\theta(x) = \theta_0 + \cancel{\theta_1 x} + \cancel{\theta_2 x^2} + \cancel{\theta_2 x^3} + \cdots + \cancel{\theta_n x^n} \tag{4.30}$$

Thus, the model becomes underfitted for large values of λ, and it should be tuned to acquire the optimal value.

Visualizing error to diagnose over-fit/under-fit problems

To further debug the model, a guide is given in Fig. 4-14. Here, the errors for training and cross validation sets have to be calculated for a variety of polynomial degrees or regularization parameters. When training and cross-validation errors are plotted as a function of polynomial degree (or complexity of model), a larger training and cross-validation error shows that the model is likely to be an under-fit (high bias) model. A large cross-validation and low training error shows that the model is likely to be an over-fit model (high variance). Training and cross-validation errors are calculated as in Equations 4.14 and 4.15. Here, it is assumed that there is no regularization term in the cost function. The training and cross-validation errors are calculated in the same way as the cost function, since the cost function itself is a measure of error.

In the case that the cost function is regularized as given in Equ. 4.27, the training and cross-validation errors are calculated ignoring the regularization term, so they are defined exactly the same as given in Equations 4.14 and 4.15, respectively.

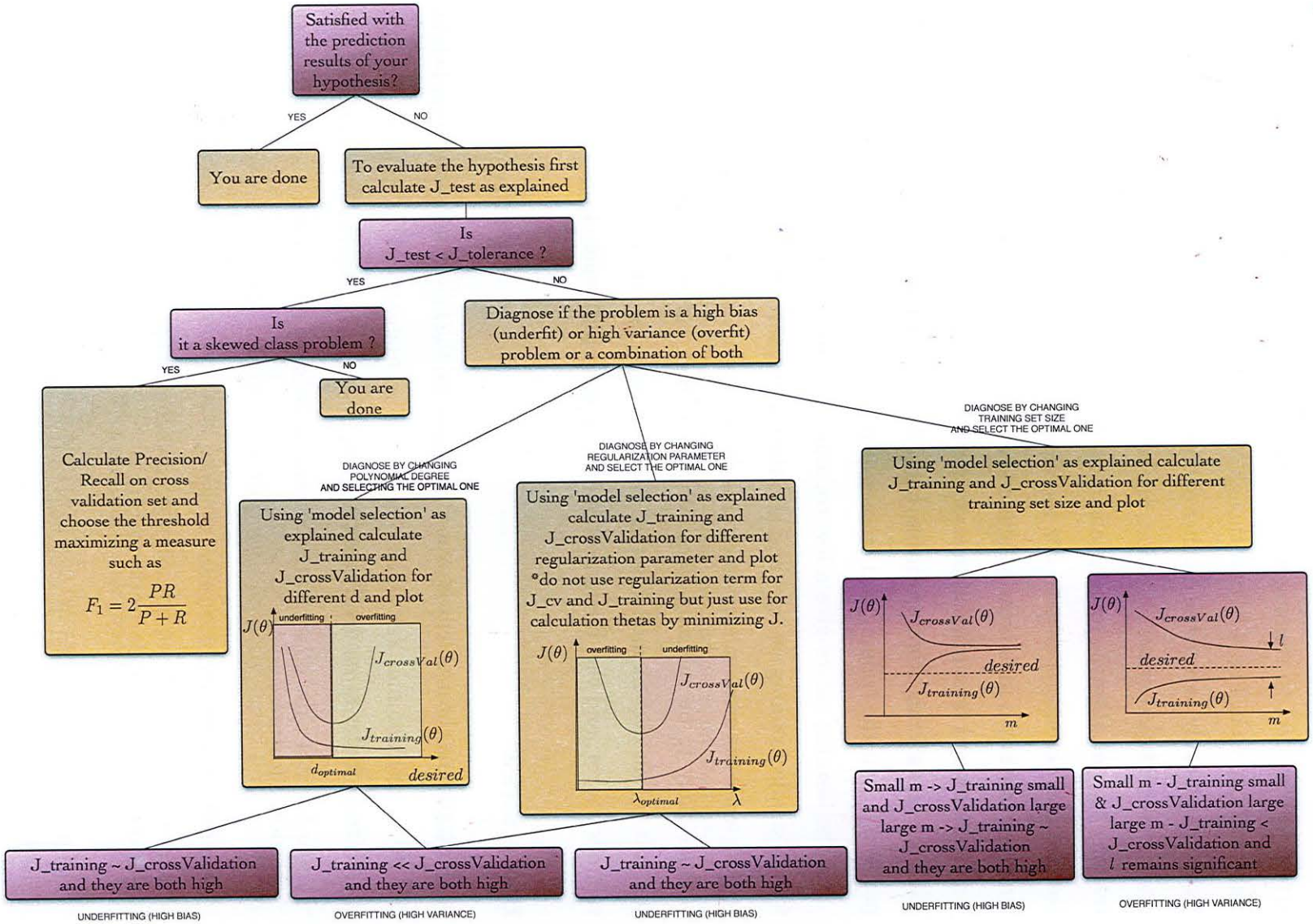

Figure 4-14: Diagnosing machine learning problem by plotting training and cross-validation errors

Another way to diagnose learning problems is to plot training and cross-validation errors with respect to training set size, also named as the learning curves (the two rightmost curves in Fig. 4-14). Although, the training set size is normally constant, here we artificially reduce the number of training examples and calculate the training and cross-validation errors. For that, the model parameters $\boldsymbol{\theta}$ are fit to this reduced training set, and then the training and cross-validation errors are also calculated over this reduced training data set. For the cases where the resulting curves give a low training error and a high cross-validation error for smaller training set sizes, and also high training and cross-validation errors for larger training sizes, the problem is most likely to be suffering from under-fit (high bias). Another signature of an under-fit problem is that the training and cross-validation errors are similar in magnitude.

If the learning curves give a low training error and a high cross-validation error for the smaller training set size, and also the training error is increasing with training set size while the cross-validation error is decreasing without leveling off, the problem is more likely to be suffering from overfitting (high variance). Another important signature of an over-fit model is that the training error will be less than the cross-validation error, and the difference between them would remain significant.

Best practices

After diagnosing the problem as over-fit or under-fit, the following practices are helpful to know.

To fix an under-fit model, it may help to tr:

- adding features;
- adding polynomial features;
- decreasing the regularization term.

To fix an over-fit model, it may help to try:

- adding more training examples;

- decreasing the number of features;
- increasing the regularization term.

Prediction

Finally, for a new set of input data, the output can be predicted using the trained model/classifier. One point to remember is that if scaling is applied to the training set, new data should be scaled as well before being fed to the model/classifier, using the same mean and standard deviation values previously calculated from the training set. After that, $h(\theta)$ could be evaluated using optimized θ and the scaled new data set x_new_scaled. So, $h_\theta(x\_new\_scaled)$ gives:

$$h_\theta(x\_new\_scaled) = P\Big(y = 1 \,|\, x;\, \theta\Big) \tag{4.31}$$

which is the probability that $y = 1$ given x_new_scaled parametrized by $\boldsymbol{\theta}$.

4.2 Support Vector Machines

4.2.1 Introduction

SVM is a relatively new approach for classification, offering promising generalization properties thanks to its foundations on the structural risk minimization principle, while other classifiers (such as logistic regression) usually minimize the empirical risk [44, 101]. This advances the capacity of generalization, even with a small number of instances, by reducing the risk of overfitting with properly tuned parameters. It can be applied to nonlinear systems and problems offering a vast number of features. Since the problem is represented as a convex optimization problem, SVM avoids local minima, to which Neural Networks(NN) are inherently prone.

The aim of SVM is to find an optimal hyperplane maximizing the margin, which is the distance in between the boundaries, by extending them until hitting the first data point, as in Fig. 4-15. These data points that are closest to the hyperplane

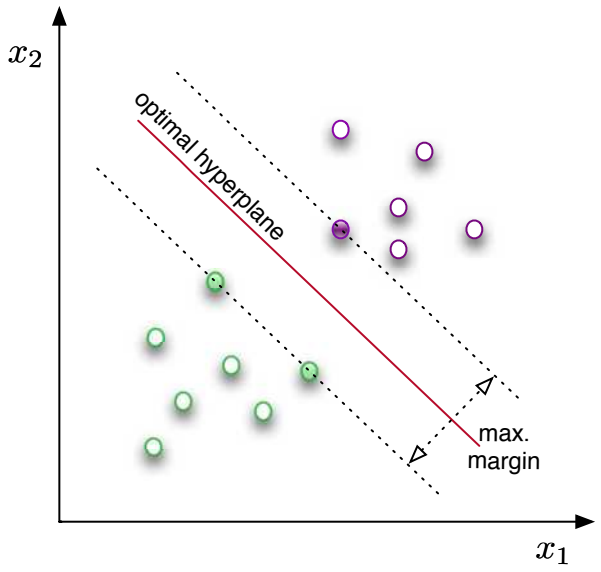

Figure 4-15: SVM working principle: The aim of SVM is to find an optimal hyperplane maximizing the margin, which is the distance in between the boundaries, by extending them until hitting the first data points (support vectors)

(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 abruptly decrease the data to handle, enhancing the ability to cope with the curse of dimensionality and reducing the computational complexity.

SVM implements the idea of having confidence in the prediction by using the concept of separating data with a large margin. Some other classification methods, such as logistic regression, output the probability of a new instance belonging to a particular class, thus inherently giving the confidence of the prediction as an output. On the other hand, SVM does only output if the new instance's probability of belonging to a particular class, so does not give its confidence on this decision explicitly. Rather than including this confidence information as an output in terms of probabilities, this information is introduced with the functional and geometric margins. Methods are available for calculating the posterior probabilities, which is the probability that the new measurements belongs to its predicted class [78].

The 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:

$$min_{\gamma,\omega,b} \quad \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i \tag{4.32}$$

$$s.t. \quad y^i(\omega^T x^{(i)} + b) \geq 1 - \xi_i, \ i = 1, \cdots, m \tag{4.33}$$

$$\xi_i \geq 0, \ i = 1, \cdots, m \tag{4.34}$$

Here, the aim is to calculate the parameters ω and b of the hypothesis by minimizing the objective function. In SVM, we drop the convention of denoting the hypothesis as $h(x) = \theta^T x$ and use $h_{\omega,b}(x) = g(\omega^T x + b)$ where $g(z) = 1$ if $z \geq 0$, and $g(z) = -1$ otherwise. In order to generalize to the non-linearly separable datasets, ξ_i is introduced in the equation's inequality constraint. Now, in the non-separable case, the functional margins $(y^i(\omega^T x^{(i)} + b))$ of the instances (examples) are permitted to have a value less than 1, different from when the case is separable. In order to ensure that most of the examples still do not violate the boundary, and thus have a functional margin of at least 1, the objective function is also increased by $C\xi_i$. C is called the box constraint, and usually needs to be tuned to achieve a satisfactory performance.

Introducing the dual form of the optimization problem, the use of kernels (to work efficiently in higher dimensional spaces) is eased. The dual form also allows the use of efficient optimization solvers such as Sequential Minimal Optimization (SMO) [77], which is the solver used in this work as well. Conventionally, the training phase of SVM requires the solution of a large quadratic programming (QP) problem. Especially for a large training set, the computational heaviness of this phase may

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.

SVM has other options to deal with not linearly separable problems, such as using kernels to map data into higher dimensional feature spaces where they can be separated with a linear hyperplane. Kernels are at the core of efficient SVM classifiers. A kernel, in general, is defined as:

$$K(x, z) = \phi(x)^T \phi(z) \tag{4.35}$$

where ϕ represents the feature mapping. 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.

4.2.2 Application

A binary classifier is used in this work to classify two classes, faulty and nominal. SVM, being a supervised classification algorithm, has two main phases as shown in Fig. 4-2: training and prediction.

In the training phase, the model learns to fit the labeled data that is fed to the SVM algorithm. The labeled data set is first divided into two portions with percentages of 80% and 20%, where the larger chunk is the training set and the remaining is the test set, respectively.

This phase is usually followed with a tuning phase where some of the parameters of SVM are changed. 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 (it may 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 new data input, the classifier is used to predict which class the new data belongs to.

Training of the classifier

The first step is to normalize the features of the data as shown in Eq. 4.17 in order to make the values of features change within the same order of magnitude. Normalization offers advantages for the optimization algorithm and its convergence rate, during the calculation of model parameters. An important point is to keep the values of the mean and range μ_j, s_j (calculated with Eq. 4.18) and standard deviation (if it is used instead of range s_j). During the prediction phase, the data should first be scaled with these values attained from training data.

Default settings for SVM binary classification, included in Matlab Statistics and its Machine Learning Toolbox, utilizes SMO for optimization if outlier fractions are not specified during the function call.

Default settings of Matlab's binary SVM classifier fits a linear model, which results in a linear decision boundary. In this study, a Gaussian Kernel, which corresponds to an infinite dimensional feature mapping, is used to map the attributes:

$$K(x,z) = exp\bigg(-\frac{\|x-z\|^2}{2\sigma^2}\bigg) \tag{4.36}$$

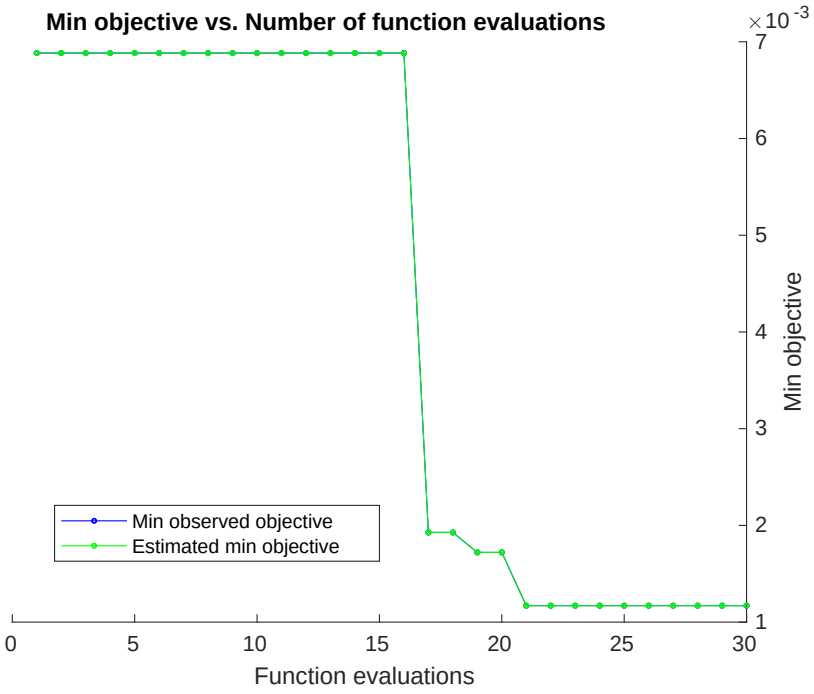

Figure 4-16: Convergence of the objective function

Tuning of the classifier

The training phase is usually followed by a tuning phase where some of the parameters of SVM are tuned and the 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.

To avoiding overfitting, which is the main problem of parametric discrimination approaches such as neural networks, C (box constraint) and σ (kernel scale), are tuned. The classifier is trained with the training set and tuned via the cross-validation set, and then the selected classifier is evaluated with the test set. The cross-validation set selection of Matlab uses 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 (σ) are tuned considering the presence of the outliers to generalize the distribution of the data rather than resulting in fine fits for each individual datum in the training set.

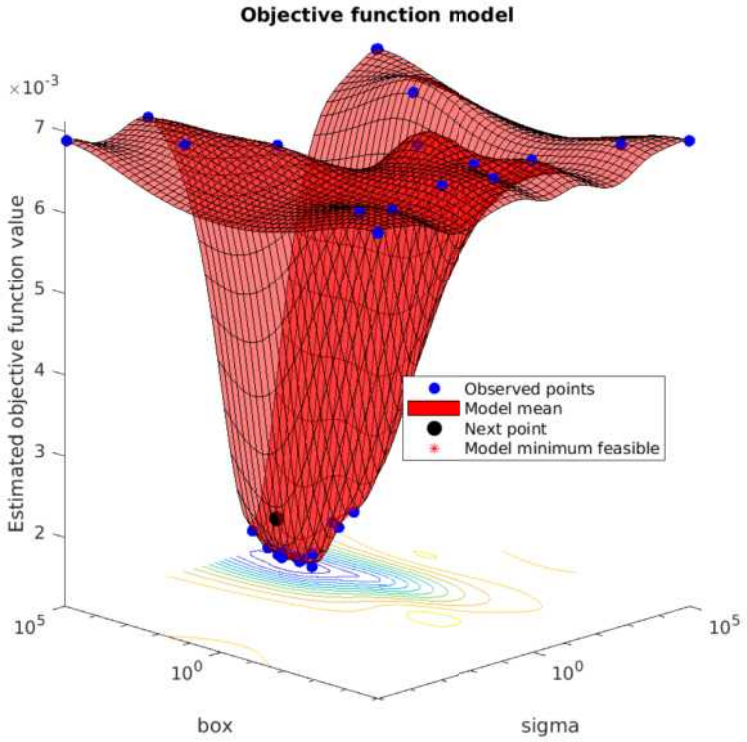

Figure 4-17: Objective function values for different box parameter and sigma values

Two different ways are used 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, box constraint parameters from a geometric sequence have been tried (11 values, from 1e-5 to 1e5 by a factor of 10). Increasing box constraint might decrease the number of support vectors, but may also increase the training time. Usually, increasing box constraint too much induces overfitting (high variance).

The Bayesian optimization tool from Matlab can be used in conjunction with the classification tool to optimize box constraint and the kernel scale. During optimization, the objective function is printed with respect to function evaluations by the optimization tool, as shown in Fig. 4-16. Here, the objective is to successfully converge to its minimum value after 21 function evaluations. Fig. 4-17 shows the objective function values for a variety of box constraint and kernel scale values. The

optimized values for box constraint and kernel scale in each iteration are also presented as a table (see Fig. 4-18). Also, a summary of Bayesian optimization showing the calculated and estimated best values for sigma and box constraint is given at the end of optimization (see Fig. 4-19).

Evaluating the classifier

The performance of the classifier is first quantified by the *loss of classification* (represented as *kFoldLoss* in results) in this work. The training data set is separated into 10 folds. For each fold, the *loss of classification* is computed for in-fold observations with a model trained on out-of-fold observations. Finally, taking an average of the *loss of classification* of 10 folds, the final classification error is calculated. 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 may not generalize well to new data.

Another means by which to evaluate the 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, the *classification score* is considered as the signed distance from each observation to the decision boundary.

While all of these variables would be satisfactory for the performance analysis of classifiers, due to the skewed-class nature of the problem, another means of evaluation 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 with the fact that predicting only the more frequent class may lead to a misunderstanding that the classifier gives superior performance, although it might not be even learning at all. For the problem of the fault detection of a control surface, it would serve as a good example to clarify more. Since nominal

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	sigma	box
1	Best	0.0067456	5.9508	0.0067456	0.0067456	0.0045697	0.00062436
2	Accept	0.0067456	6.0353	0.0067456	0.0067456	10225	0.002352
3	Accept	0.0067456	5.7157	0.0067456	0.0067456	82503	466.87
4	Accept	0.0067456	120.58	0.0067456	0.0067456	0.045189	162.15
5	Accept	0.0067456	6.766	0.0067456	0.0067456	0.80469	0.0003344
6	Best	0.0026156	9.6041	0.0026156	0.0026156	0.88414	24452
7	Best	0.002478	8.9766	0.002478	0.002478	821.47	99999
8	Accept	0.0067456	110.34	0.002478	0.002478	0.00030436	53383
9	Best	0.0015831	122.92	0.0015831	0.0015832	36.623	41412
10	Accept	0.0021338	11.758	0.0015831	0.0015832	126.08	23741
11	Best	0.0010325	447.56	0.0010325	0.0010325	15.435	92255
12	Accept	0.0013078	587.24	0.0010325	0.0010325	8.3594	96906
13	Accept	0.0015831	240.63	0.0010325	0.0010325	39.235	99945
14	Accept	0.001239	72.811	0.0010325	0.0010325	12.36	5040.3
15	Accept	0.0067456	7.3057	0.0010325	0.0010325	88.547	0.00022298
16	Accept	0.0010325	325.18	0.0010325	0.0010325	14.61	43633
17	Accept	0.0067456	7.0633	0.0010325	0.0010325	1.0006e-05	1.0165e-05
18	Accept	0.0011013	280.57	0.0010325	0.0010326	16.38	49813
19	Accept	0.0011013	658.47	0.0010325	0.0010326	12.919	90034
20	Accept	0.0010325	293.65	0.0010325	0.0010325	8.2745	15324
Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	sigma	box
21	Accept	0.0013767	119.47	0.0010325	0.0010325	18.204	13408
22	Accept	0.0010325	69.685	0.0010325	0.0010325	4.0255	3063.4
23	Accept	0.0010325	89.881	0.0010325	0.0010325	5.7358	3044.8
24	Accept	0.0018585	77.761	0.0010325	0.0010325	3.2846	65963
25	Accept	0.0011702	18.874	0.0010325	0.0010325	2.7212	243.33
26	Accept	0.0010325	26.302	0.0010325	0.0010325	3.9759	362.27
27	Accept	0.0067456	8.6061	0.0010325	0.0010325	382.52	750.49
28	Best	0.00096366	26.488	0.00096366	0.00096366	2.7047	911.28
29	Accept	0.0011702	30.843	0.00096366	0.00096372	3.5526	846.04
30	Accept	0.0021338	11.662	0.00096366	0.00096373	1.1441	1020.8

Figure 4-18: Box constraint C, kernel scale σ and objective function $J(\theta)$ values after each iteration during Bayesian Optimization. Minimum objective (0.00096366) can be seen at iteration number 28, and corresponding $C = 911.28$ (given as *box*) and $\sigma = 2.7047$ (given as *sigma*) are selected as the best values in this tuning.

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 3833.7318 seconds.
Total objective function evaluation time: 3808.709

Best observed feasible point:
    sigma      box
    ______    ______
    2.7047    911.28
Observed objective function value = 0.00096366
Estimated objective function value = 0.00096373
Function evaluation time = 26.4882

Best estimated feasible point (according to models):
    sigma      box
    ______    ______
    2.7047    911.28
Estimated objective function value = 0.00096373
Estimated function evaluation time = 25.986
```

Figure 4-19: Final results for the optimization, time of execution, optimal values of box constraint and sigma values

data is not difficult to generate in real flight, while it is difficult to fly faulty, the nominal data is much more 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. 4-20 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 that 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 were predicted as faulty. Precision can be related to number of false alarms which is cautiously avoided in flight control systems. Precision is defined as :

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{4.37}$$

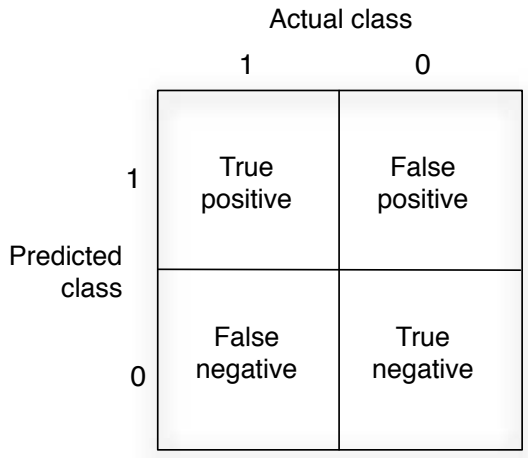

Figure 4-20: Confusion matrix

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

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{4.38}$$

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

$$f1Score = \frac{2 * precision * recall}{precision + recall} \tag{4.39}$$

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

4.3 Conclusion

In this chapter, an introduction to machine learning algorithms is presented. Two main branches of machine learning — supervised and unsupervised learning — are

introduced. Terminology and construction of input and output vectors are presented using easy examples of regression and classification. Regression and classification are two common problems discussed under supervised learning problem. The problems that are frequently encountered during machine learning applications are discussed and best practices to achieve more accurate results are presented.

After this generic introduction to machine learning implementations, *Support Vector Machines* (SVM) are discussed. A very brief section explains their mathematical background and then application procedures are presented.

The next chapter explains the simulation results by applying SVM to drone flight data. First, results from the application of SVM to simulated measurements are presented. Then, SVM is applied to flight data. To be precise on how the data is generated, the injection of faults during flight, and the modifications to the *Paparazzi* autopilot in order to realize the faults, are explained in detail.

Chapter 5

Simulation Results

This chapter focuses on the results of fault classification simulations under two main sections: classification of faults based on simulated flight measurements and classification of faults based on real flight data. In the first part, flight data simulation uses the mathematical equations explained under mathematical modeling chapter of this thesis. The second part commences with a thorough explanation of the path taken to generate faults in real flight for two reasons: first, the importance of having knowledge on data, how it is generated/labeled; and second, constructing a guide for researches so that they can realize their own faulty flight campaigns. Following this, the classification for the control surface stuck fault and the loss of effectiveness fault are investigated separately.

5.1 Fault detection from simulated flight data

In this section, the model of an aircraft is simulated in Matlab using the equations of motion given in Equ. 3.95. This drone model will not be used for the design of FDD algorithms, but rather to generate data that will be used by the FDD algorithms. After the equations of motion of the drone have been solved numerically, the accelerometer and gyro measurements are simulated based on the statistics of the onboard sensors. This part of the study uses the model of a MAKO drone to simulate the measurements, while the real flights (that will be explained in the forthcoming

section) use a ZAGI drone.

For the MAKO simulation, the stability and aerodynamic force coefficients are generated by AVL. The input vector can be written as $\boldsymbol{u}(t) \in \mathbb{R}^3$:

$$\boldsymbol{u}(t) = \begin{bmatrix} \delta_a & \delta_e & n \end{bmatrix}^{\mathrm{T}} \tag{5.1}$$

Here, δ_a aileron deflection angle is in degrees, δ_e elevator deflection angle is in degrees, and n engine speed is in rev/s.

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

$$\boldsymbol{u}(t) = \boldsymbol{E}\boldsymbol{u}_c + u_f \tag{5.2}$$

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

Most of the FDD algorithms are designed considering open-loop systems, ignoring the probable influence of the controller on the detection performance [75]. Throughout this first section, the system is also open-loop. In this chapter, a step by step approach is followed. In this first section (diagnosis with simulated data), the effect of the controller is ignored, while in the next section (diagnosis with real flight data), diagnosis is achieved alongside a functioning controller.

The code for simulations in this section can be reached in *Github*[1]. First, the measurements are simulated for faulty and nominal flight conditions. An example control input (75% loss of efficiency), could be an actual aileron command of $\delta_a^a = 1^\circ$ corresponding to a desired $\delta_a^d = 4^\circ$. In the simulation, this fault is introduced after 120 s. Actual aileron command and corresponding simulated accelerometer $x - axis$ readings can be seen in Fig. 5-1. The measurements are labeled, and an example plot showing simulated measurements in two-dimensional feature space, a_x - a_y, is given

[1]https://github.com/benelgiz/curedRone

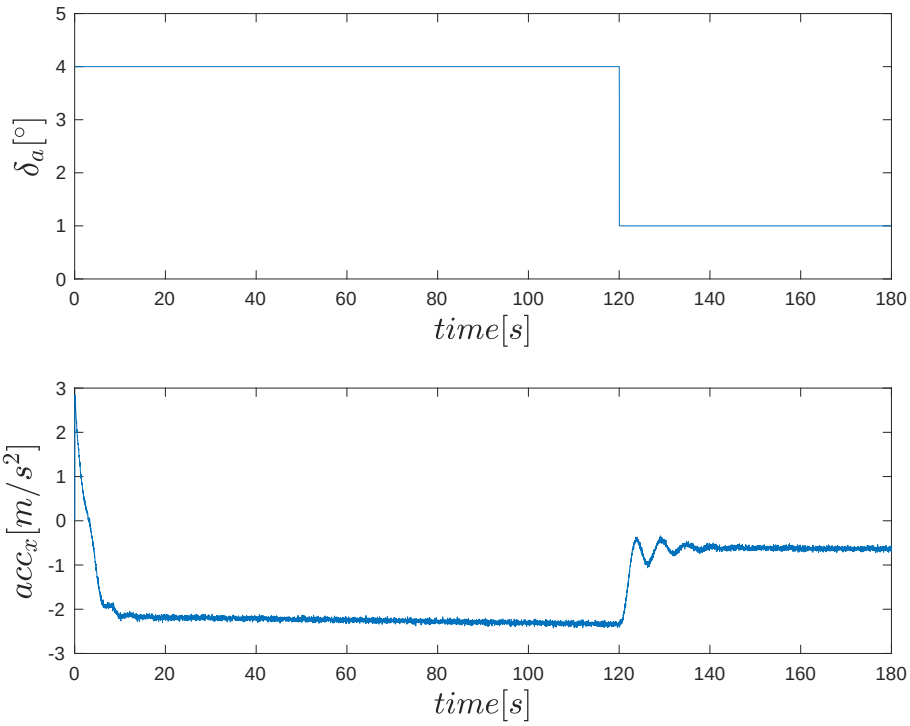

Figure 5-1: Loss of effectiveness fault simulation in aileron command and corresponding accelerometer x axis measurement

in Fig. 5-2.

It is always important to visualize the features to be able to comprehend data structure before applying machine learning algorithms. For that reason, the available observations form the six-dimensional pattern space, $\vec{\boldsymbol{x}} = \begin{bmatrix} a_x & a_y & a_z & \omega_x & \omega_y & \omega_z \end{bmatrix}$ can be visualized in pairs, such as in Fig. 5-2. There are further methods used to visualize multidimensional data such as *Tours*[6, 22, 23] and the *GGobi* data visualization system [24]. In this work, the dimensionality reduction technique, Principle Component Analysis (PCA), is utilized for visualization. If briefly explained, the feature vector $\boldsymbol{x} \in \mathbb{R}^n$ is mapped to a lower dimensional space where the new feature set will be represented by $\boldsymbol{z} \in \mathbb{R}^k$. The final two-dimensional feature set can be plotted to give an idea about the distribution of faulty and nominal measurements' in feature space, as shown in Fig. 5-3.

The aim of machine learning methods is to train a model with a given data set in order to predict the output values corresponding to a new input. A binary classifier

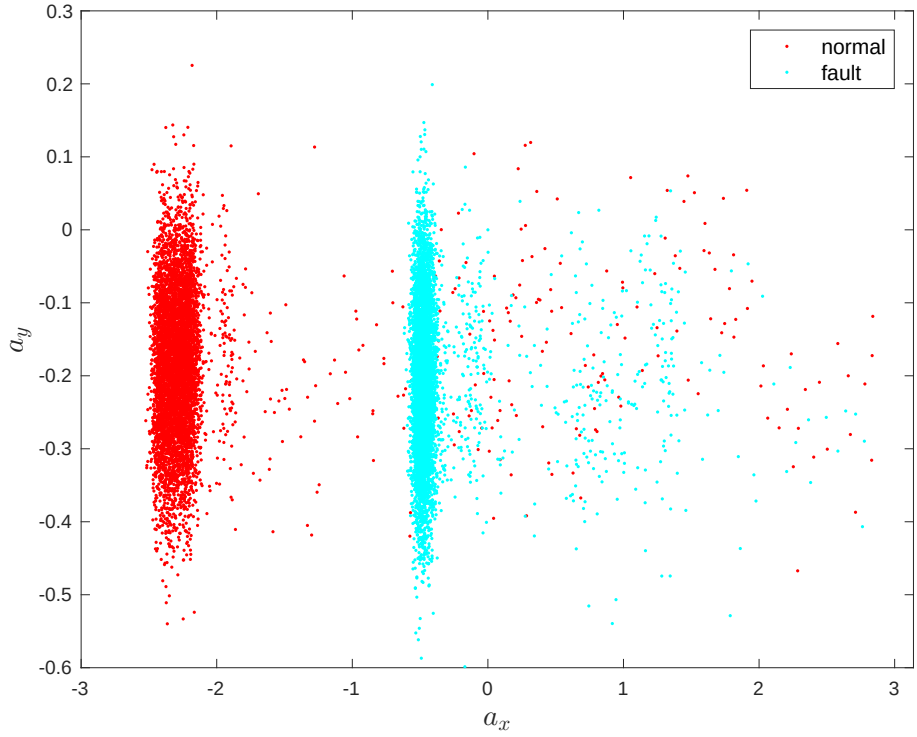

Figure 5-2: Accelerometer simulation a_x vs a_y

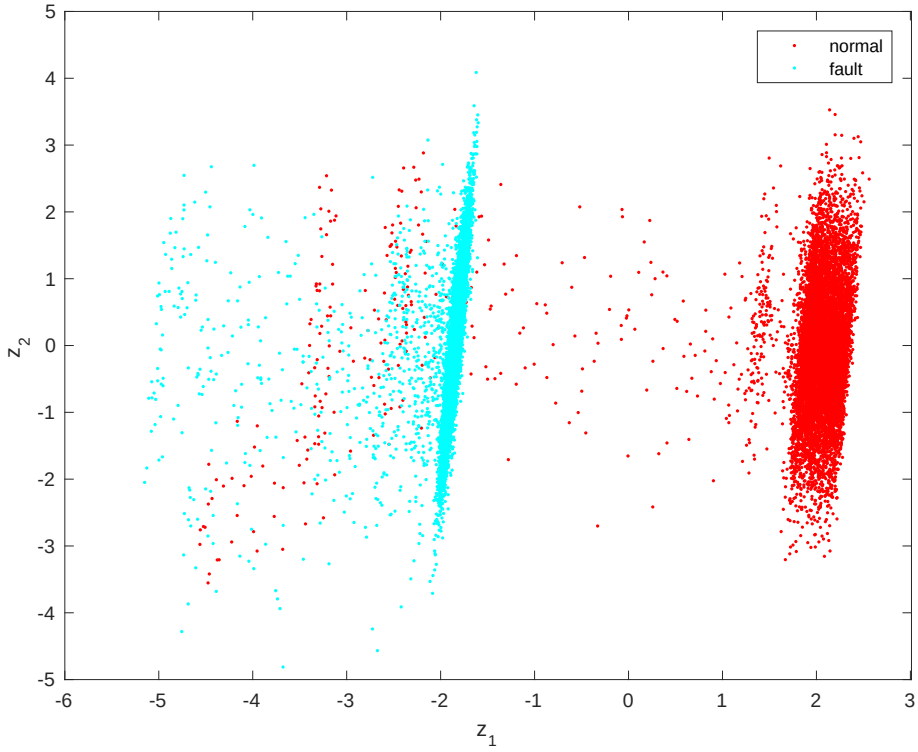

Figure 5-3: Reduced dimensional space features z_1 vs z_2

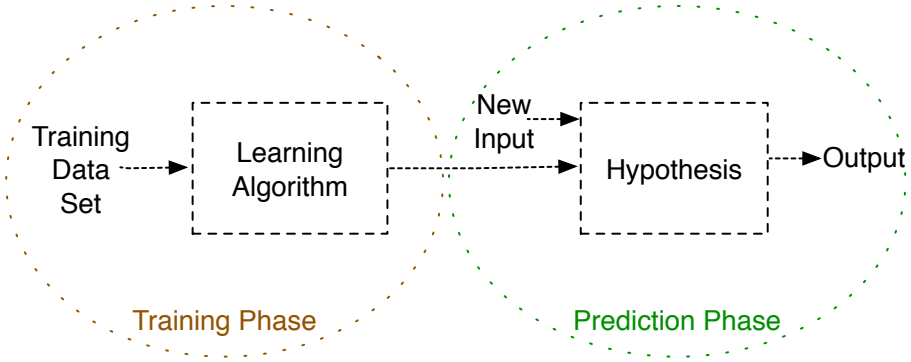

Figure 5-4: Supervised learning is achieved in two steps: *Training Phase* in which the model parameters are calculated given labeled data and *Prediction Phase* in which the label is predicted for a new input using the trained model.

is used in this work to classify two classes; faulty and nominal. The fault considered in this study is the loss of effectiveness of the control surfaces. SVM is a supervised classification algorithm and has two main phases as shown in Fig. 5-4. In the training phase, the model is learnt to fit with the labeled data that is fed to the SVM algorithm. This phase is usually followed by a tuning phase, where some of the parameters of SVM are changed and the results with the best fit are compared via cross validation in order to avoid overfitting. The last phase is the prediction, where for a new instance, the classifier predicts whether it corresponds to a faulty or nominal condition.

Training data is comprised of labeled data where the label can belong to one of two possible cases. This data set is saved in $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ where m, n correspond to number of instances and features, respectively (shown in Fig. 5-5).

$$X = \begin{bmatrix} acc_x^t & acc_y^t & acc_z^t & gyro_x^t & gyro_y^t & gyro_z^t \\ acc_x^{(t+1)} & acc_y^{(t+1)} & acc_z^{(t+1)} & gyro_x^{(t+1)} & gyro_y^{(t+1)} & gyro_z^{(t+1)} \\ \vdots & & & & & \\ acc_x^{(t+m-1)} & acc_y^{(t+m-1)} & acc_z^{(t+m-1)} & gyro_x^{(t+m-1)} & gyro_y^{(t+m-1)} & gyro_z^{(t+m-1)} \end{bmatrix}$$

Figure 5-5: Feature matrix $\boldsymbol{X} \in \mathbb{R}^{m \times 6}$ is comprised of accelerometer and gyro data of m instances

The label information corresponding to the measurement instances is also fed to

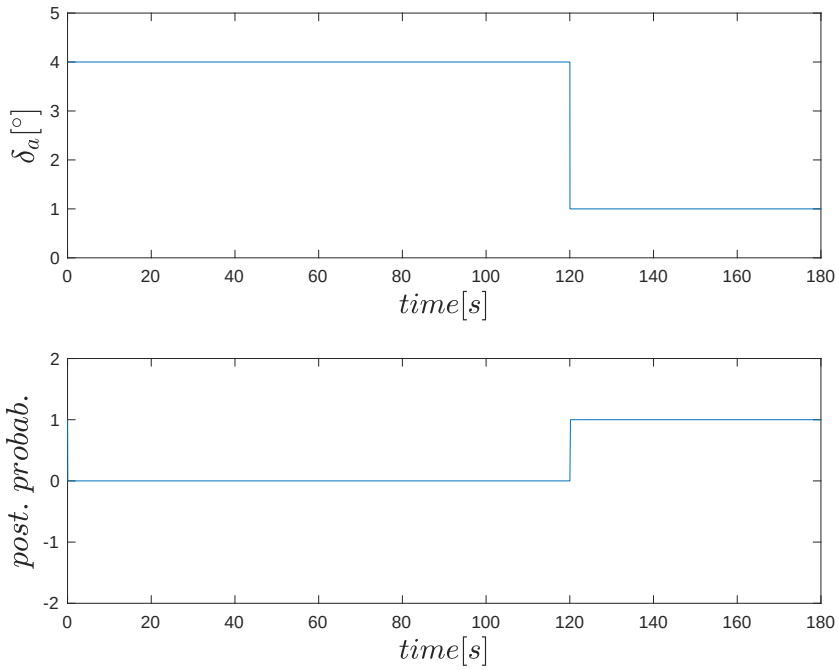

Figure 5-6: Posterior probability of loss in effectiveness fault for test set when a fault is injected at t = 120s.

the SVM algorithm during the training phase as an 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. For further details on SVM and machine learning in general, the reader could refer to the methodology chapter of this thesis.

To avoiding overfitting — which is the main problem of parametric discrimination approaches such as neural networks — parameter C (box constraint) is tuned to result in the optimal fit for the cross validation set. Box constraint C controls the values of the parameters learnt during the training phase, and is further explained in the methodology chapter. The data set available is first divided into two portions with a percentage of 20%; 80% where the bigger chunk is the training set and the remaining is the test set. Further, the training set is divided as cross-validation and training sets. The data is split in order to avoid overfitting. In overfitting, the models fit very accurately to the data that they are trained with, but they fail to generalize to new inputs, resulting in bad prediction performance for the new data. To improve the

Figure 5-7: A scene from one of many flight campaigns realized by ENAC UAV laboratory team. A safety pilot can be seen holding the remote control. Photo taken by Alexandre Bustico.

performance of the classifier trained with the training data, it is tuned with the cross-validation data. Finally, the ability of the classifier is tested on the test set. This parameter is also tuned for the outliers to generalize the distribution of the data, rather than having fine fits for all individual data in the training set. A satisfactory result of the training and tuning is followed by prediction, where the classifier predicts if the new measurement data belongs to the faulty or nominal class. The output of the SVM classification is not the probability that the new measurement belongs to one class (as in traditional classification problems), but the class information it belongs to. To investigate the performance of the classifier on the test set, a method [78] is used to calculate the posterior probabilities given the probability that the new measurements belong to a faulty mode. Results in Fig. 5-6 show that proper tuning achieves very accurate and instant detection of the drone fault.

5.2 Fault detection from real flight data

This second section of this chapter discusses the classification results using real flight data. Flight campaigns have been realized in the presence of a safety pilot to recover the drone in the case of loss of control when the faults are injected. A scene from one of the many flight campaigns realized by the ENAC UAV laboratory team is given in Fig. 5-7. The safety pilot can be seen in the photo holding the remote control.

In this section, the injection of faults to the flights are explained in detail. The Paparazzi GCS has been altered to inject real-time faults, and controllers in some of the flight modes have been changed to accommodate faults. Next, the selection of the faults and nominal phases, and also labeling data, are presented. Finally, the labeled data used to train classifiers for elevon stuck and loss of efficiency faults separately and the classifiers are evaluated. We implement a variety of techniques to improve performance, such as feature engineering and tuning the classifiers.

5.2.1 Injecting faults in flight from Paparazzi GCS

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

In order to inject faults real-time from GCS, a slider is added to the GCS to set the fault during flight and then set it back to normal flight conditions if necessary. Fig. 5-8 shows the GCS view with the fault settings open. This pane can be found under *Settings* > *FAULT* as highlighted in pink in Fig. 5-8. The four-row configuration represents, from top to bottom, the multiplicative error in the right elevon, the multiplicative error in the left elevon, the additive error in the right elevon, and finally, the additive error in the left elevon. The nominal condition where there is no fault, is given by $[right \quad left \quad right\_offset \quad left\_offset] = [1.0 \quad 1.0 \quad 0 \quad 0]$. This configuration allows the user to realize all types of actuator faults, such as control surface inefficiency or stuck. To generate the fault of the right elevon stuck at

Figure 5-8: View of fault injection tool in *Paparazzi* ground control station

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 the desired stuck position while keeping the first slider at zero. As soon as *Settings* is changed from the GCS (from nominal to faulty or from faulty to nominal), a message is saved to *Paparazzi Messages* (onboard SD card) with the values set in the GCS, as shown in Fig. 5-9 (SETTINGS Message). This message includes the time (the moment that the *Settings* are changed from the GCS), the aircraft number, and the values set from the GCS ($[right \quad left \quad right\_offset \quad left\_offset]$).

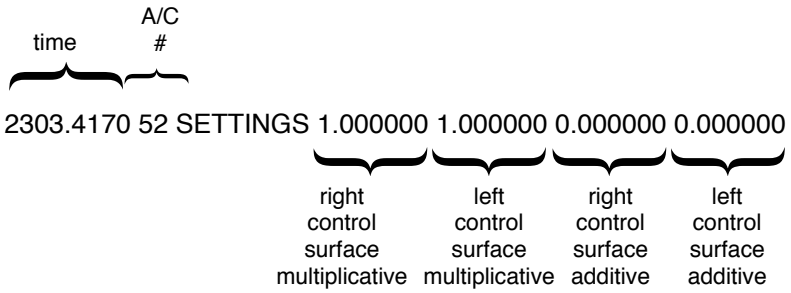

Figure 5-9: *SETTINGS Message* saves the multiplicative and additive fault values inserted from the GCS. [1.0 1.0 0.0 0.0] corresponds a command from the GCS to revert back to nominal phase

5.2.2 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, we examine the *Paparazzi* flight modes is necessary. For the majority of time, there are three modes for fixed-wings from the 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. The desired elevator and aileron commands are then calculated by the autopilot and passed to control allocation where the final desired servo commands are sent to servos (highlighted by *Auto 1* in Fig. 5-10. In the *Auto 2* mode, there is no need for a 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. 5-10. In *Manual*

mode, the pilot gives the desired elevator and aileron commands, and the desired servo commands are calculated in the autopilot's control allocation phase. So, there is still a very low sense of autonomy in the manual phase.

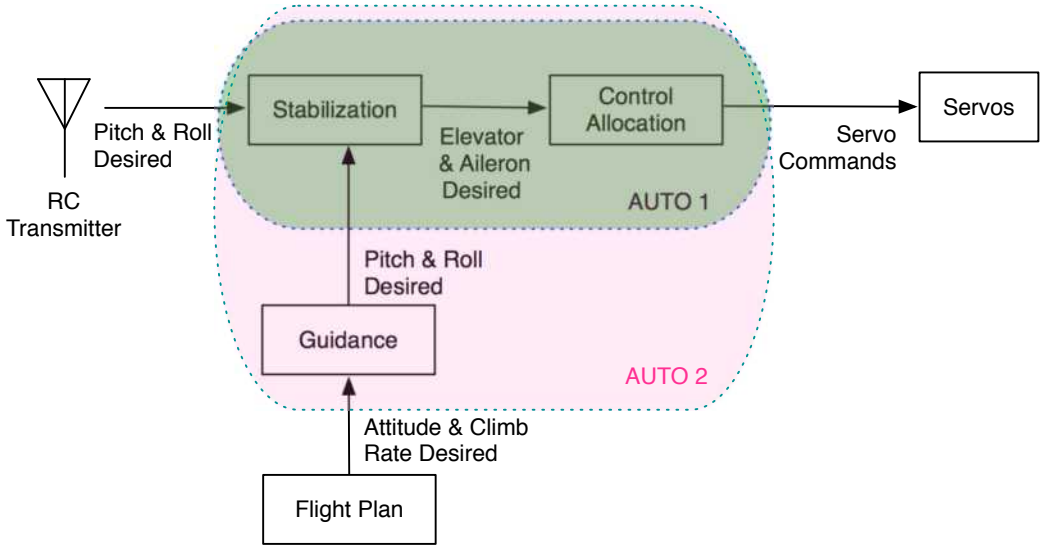

Figure 5-10: Paparazzi autonomy modes

Flying with faults is a challenge. The risk of crashing is increased on purpose, so a back up plan is necessary so as 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. Thus, when the pilot sees a safety problem during the faulty operation, s/he can switch to *Manual* mode from the remote controller and then has the control of the control surfaces free from faults. Depending on the nature of the fault injected, such as for some severe stuck control surface fault data gathering, this could be a game changer, since the drone would crash unless action is taken. This is shown in Fig. 5-11 with a switch initiated by the pilot's remote control.

This figure also 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 does not consider the injected faults.

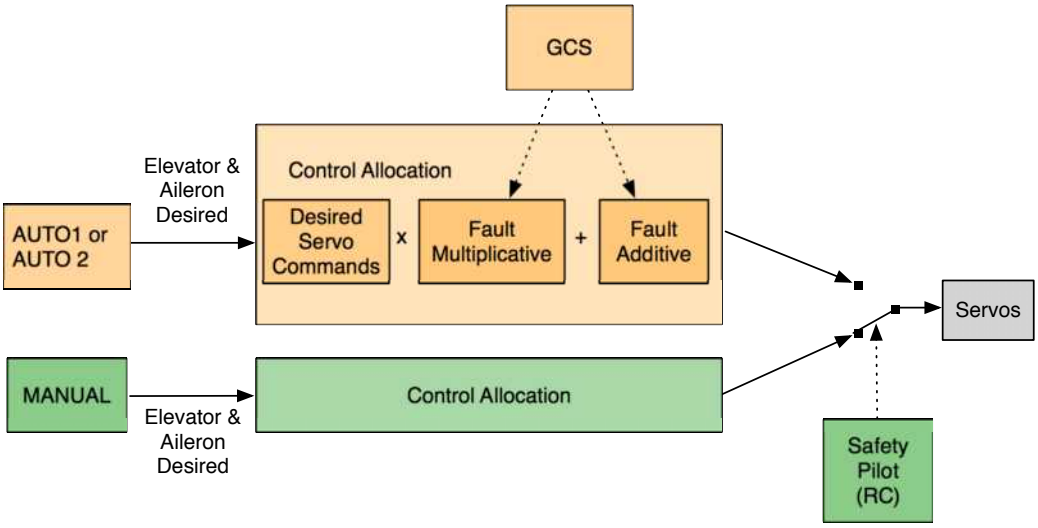

Figure 5-11: Modifications on the control modes of *Paparazzi* autopilot

The modifications in the control allocation code in order to consider additive and multiplicative faults can be seen in the *Paparazzi* code (in C) given below. In *Manual* mode, the servos are not changed by the injected faults. The desired elevator and aileron commands given by the pilot are changed to servo commands by control allocation as usual. In case of *Auto 1* and *Auto 2* modes, control allocation is modified by adding the additive faults ($vfault\_offset\_left$,$vfault\_offset\_right$) and multiplying by the multiplicative faults ($vfault\_left$, $vfault\_right$). This changes the calculation of servo commands ($AILEVON\_LEFT$ and $AILEVON\_RIGHT$) in the line of code ($set\ servo = AILEVON\_LEFT$ and $set\ servo = AILEVON\_RIGHT$). In the case of a nominal flight, or setting the drone back into the nominal flight, since the multiplicative faults ($vfault\_left$, $vfault\_right$) are set to 1.0 and additive faults ($vfault\_offset\_left$,$vfault\_offset\_right$) are set to 0.0, the allocation of the control commands into the servo commands are not modified. In the case of *Manual* mode, the autopilot ignores the fault injection with the same idea; that is, multiplicative faults ($vfault\_left$, $vfault\_right$) are set to 1.0 and additive faults ($vfault\_offset\_left$,
$vfault\_offset\_right$) are set to 0.0.

```
<command_laws>
  <let var="aileron" value="@ROLL * AILEVON_AILERON_RATE"/>
  <let var="elevator" value="@PITCH * AILEVON_ELEVATOR_RATE"/>
  <let var="manual" value="(fbw_mode==FBW_MODE_MANUAL)"/>
  <let var="vfault_left" value="100 * ($manual ? 1 : fault_left)"/>
  <let var="vfault_right" value="100 * ($manual ? 1 : fault_right)"/>
  <let var="vfault_offset_left" value="($manual ? 0 : fault_offset_left)"/>
  <let var="vfault_offset_right" value="($manual ? 0 :
     fault_offset_right)"/>
  <set servo="MOTOR" value="@THROTTLE"/>
  <set servo="AILEVON_LEFT" value="(($elevator - $aileron) * $vfault_left)
     / 100 + $vfault_offset_left"/>
  <set servo="AILEVON_RIGHT" value="(($elevator + $aileron) *
     $vfault_right) / 100 + $vfault_offset_right"/>
</command_laws>
```

5.2.3 Reading and labeling flight data

Flight data saved to the onboard SD card should be converted to *data* format by using the *sd2log* program of *Paparazzi*. To do this, from the terminal, browse into the folder including the log file in *LOG* format, which is the file format for the onboard data saved in Paparazzi.

Figure 5-12: Conversion of raw flight data saved to SD card onboard to .data file to be used in further calculations

Run the program under paparazzi/sw/logalize/sd2log and specify the file to be converted to *.data* format and where to export it, as shown in Fig. 5-12 ('.' means

extract to current folder).

An example of part of the flight data[2] is given in Fig. 5-14, and the whole file is available in *Github*[3]. The UAV used to realize the faulty flights in order to generate labeled data is given in Fig. 5-13.

Figure 5-13: The flying-wing: *ZAGI*

The duration of the flight was approximately one hour. The flight has also been practiced under a strong wind. Thirty-four different faults are injected. During the flight, the effect of the faults on the drone were sometimes visible to human eye, and sometimes not. For the control surface stuck faults, even if only one control surface was stuck, the drone was immediately out of control and the safety pilot then takes the initiative. Thanks to the piloting skills, no crashes occurred. For the ineffectiveness of the control surface faults — where the controller has still an effect but not as efficient as before — error in navigation was observed.

After converting the flight data file to *.data* format, the file can be read from Matlab. The next step is to detect the time stamps at which the faults are injected and then label all of the data corresponding to this fault interval, as designated with different colors in Fig. 5-14. As there is usually more than one fault type generated during flights, another step in data manipulation is to choose which faults to

[2] 17_07_06__10_21_07_SD.data

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

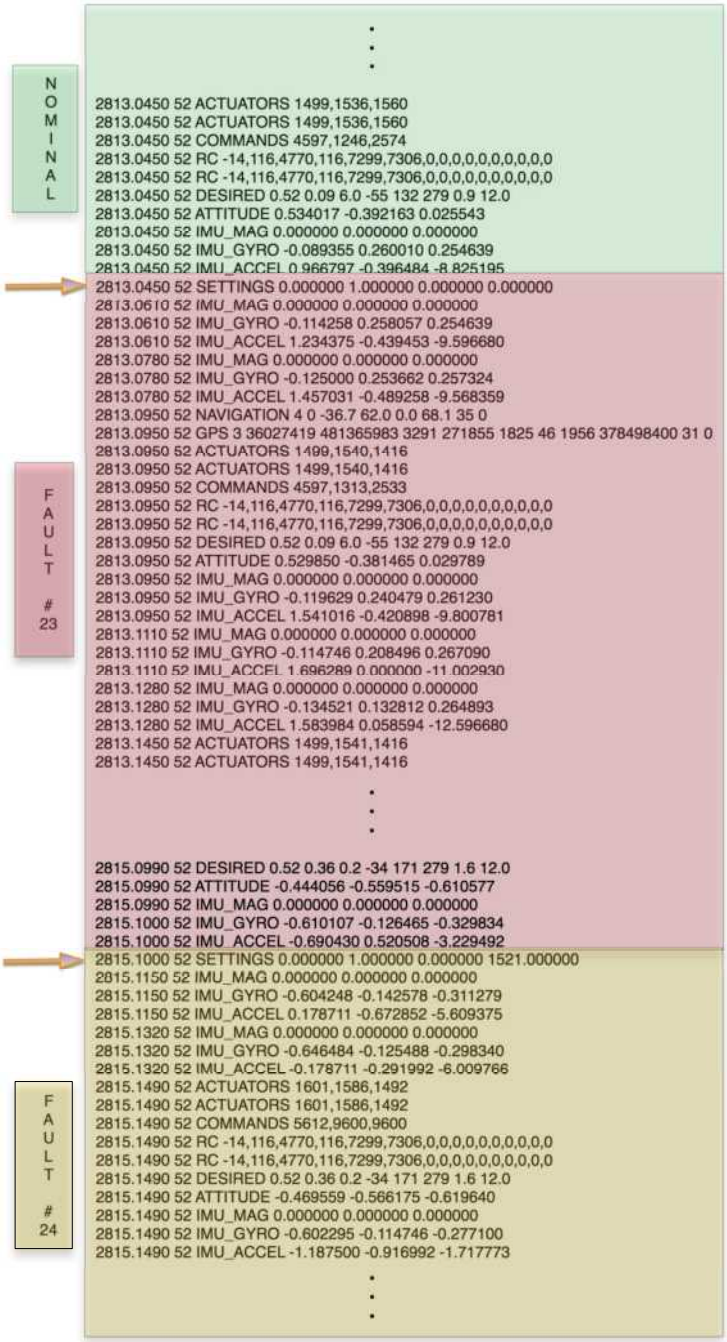

Figure 5-14: A piece of the flight data corresponding to nominal and two different fault phases of the flight. *SETTINGS* message exists only if there is a change in the multiplicative and additive fault parameters via GCS. The first data in each row corresponds to the time stamp. Second data in each row is the aircraft number which is 52 for this flight. Third data in each data line corresponds to the label of information in that line. As an example, on the last line, the label is IMU_ACCEL. This means that this line gives accelerometer readings. Last three data in this line are accelerometer x-axis (-1.187500), accelerometer y-axis (-0.916992) and accelerometer z-axis (-1.717773) readings.

investigate.

The fault injection or change from fault condition to nominal mode is done via the GCS, with a corresponding message saved to the SD card onboard in *Paparazzi 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 the nominal phase is [1.0 1.0 0.0 0.0], and a corresponding example line in the data (corresponding to a command to revert to nominal condition) can be seen in Fig. 5-9.

This means that to multiply the value given by the controller by 1 and add 0, it does not change the values given by the autopilot. As soon as any of the control surface effectiveness values are changed in the GCS, a new *Settings* value is saved to the file (shown with arrows in Fig. 5-14).

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 a return 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 the next *Settings* message is the last index of this nominal phase. The matrix holding the start and end index for the nominal phases are saved as *nominal_ start_ stop*, as shown in Table 5.1, where the first row corresponds the start index of each nominal set and the second row corresponds to the last index of the corresponding as nominal phase. As an example, we check the second nominal phase of the flight. The first nominal phase is the flight before any fault is injected, starting from take off to the first fault injection. After the fault injected for the first time, we wait a certain amount of time and then set the *Settings* back to normal (1.0 1.0 0 0) from the GCS. This corresponds to the 516,919th data line entry (which is *SETTINGS 1.0 1.0 0 0*) in the whole data set (see nominal_start_stop(1,1) in Table 5.1). Then, another fault is injected at the time corresponding to the 521,966th line in the data file, thus until this index (see nominal_start_stop(2,1) in Table 5.1) the data corresponds to the second nominal phase.

Table 5.1: Nominal phase start stop indexes of the flight

nominal phase number	2	3	4	5	$\cdots$	12
nominal start **nominal_start_stop(1,:)**	516919	551627	755227	824545	$\cdots$	1048055
nominal stop **nominal_start_stop(2,:)**	521965	622015	782580	924704	$\cdots$	1065548

For the fault indexes, a similar approach is followed except that the *Settings* messages selected are the ones that are different to [1.0 1.0 0.0 0.0]. An example is Fault #23 in Fig. 5-15 and its corresponding start and end indexes given in Table 5.2.

```
2813.0450 52 SETTINGS 0.000000 1.000000 0.000000 0.000000
```

Figure 5-15: SETTINGS message corresponding to stuck of right control surface

Table 5.2: Faulty phase start stop indexes of the flight

fault phase number	1	2	$\cdots$	23	$\cdots$	34
fault start **fault_start_stop(1,:)**	456888	473827	$\cdots$	924705	$\cdots$	1076847
fault stop **fault_start_stop(2,:)**	473826	485762	$\cdots$	925390	$\cdots$	1077117

The next step is to choose the phases of the flight to work with. As an example, a phase of the flight where there is enough nominal data followed with an extreme fault injection is selected. This corresponds to the fourth nominal phase of the flight followed by the 23rd fault selected. Fault #23 corresponds to the right control surface stuck and can be seen in Fig. 5-14. The next step is to select the measurement corresponding to these selected time intervals, and this is achieved by logically ANDing

the indexes of interest, such as the indexes of fault and gyro data measurements as shown in Fig. 5-16. The script for labeling the nominal and faulty measurements is the *selectDataToInvest.m* file given in Appendix B.2. The output of this file is the accelerometer and gyro measurements corresponding to the selected phases of the flight.

Setting index	Fault # 23 index	Gyro index	Faulty Gyro index	
0	0	0	0	2813.0450 52 ACTUATORS 1499,1536,1560
0	0	0	0	2813.0450 52 ACTUATORS 1499,1536,1560
0	0	0	0	2813.0450 52 COMMANDS 4597,1246,2574
0	0	0	0	2813.0450 52 RC -14,116,4770,116,7299,7306,0,0,0,0,0,0,0,0,0,0
0	0	0	0	2813.0450 52 RC -14,116,4770,116,7299,7306,0,0,0,0,0,0,0,0,0,0
0	0	0	0	2813.0450 52 DESIRED 0.52 0.09 6.0 -55 132 279 0.9 12.0
0	0	0	0	2813.0450 52 ATTITUDE 0.534017 -0.392163 0.025543
0	0	0	0	2813.0450 52 IMU_MAG 0.000000 0.000000 0.000000
0	0	1	0	2813.0450 52 IMU_GYRO -0.089355 0.260010 0.254639
0	0	0	0	2813.0450 52 IMU_ACCEL 0.966797 -0.396484 -8.825195
1	1	0	0	2813.0450 52 SETTINGS 0.000000 1.000000 0.000000 0.000000
0	1	0	0	2813.0610 52 IMU_MAG 0.000000 0.000000 0.000000
0	1	1	1	2813.0610 52 IMU_GYRO -0.114258 0.258057 0.254639
0	1	0	0	2813.0610 52 IMU_ACCEL 1.234375 -0.439453 -9.596680
0	1	0	0	2813.0780 52 IMU_MAG 0.000000 0.000000 0.000000
0	1	1	1	2813.0780 52 IMU_GYRO -0.125000 0.253662 0.257324
0	1	0	0	2813.0780 52 IMU_ACCEL 1.457031 -0.489258 -9.568359
0	1	0	0	2813.0950 52 NAVIGATION 4 0 -36.7 62.0 0.0 68.1 35 0
0	1	0	0	2813.0950 52 GPS 3 36027419 481365983 3291 271855 1825 46 1956 378498400 31 0
0	1	0	0	2813.0950 52 ACTUATORS 1499,1540,1416
0	1	0	0	2813.0950 52 ACTUATORS 1499,1540,1416
0	1	0	0	2813.0950 52 COMMANDS 4597,1313,2533
0	1	0	0	2813.0950 52 RC -14,116,4770,116,7299,7306,0,0,0,0,0,0,0,0,0,0
0	1	0	0	2813.0950 52 RC -14,116,4770,116,7299,7306,0,0,0,0,0,0,0,0,0,0
0	1	0	0	2813.0950 52 DESIRED 0.52 0.09 6.0 -55 132 279 0.9 12.0
0	1	0	0	2813.0950 52 ATTITUDE 0.529850 -0.381465 0.029789
0	1	0	0	2813.0950 52 IMU_MAG 0.000000 0.000000 0.000000
0	1	1	1	2813.0950 52 IMU_GYRO -0.119629 0.240479 0.261230
0	1	0	0	2813.0950 52 IMU_ACCEL 1.541016 -0.420898 -9.800781
0	1	0	0	2813.1110 52 IMU_MAG 0.000000 0.000000 0.000000
0	1	1	1	2813.1110 52 IMU_GYRO -0.114746 0.208496 0.267090
				.
				.
				.
1	0	0	0	2815.1000 52 SETTINGS 0.000000 1.000000 0.000000 1521.000000
0	0	0	0	2815.1150 52 IMU_MAG 0.000000 0.000000 0.000000
0	0	1	0	2815.1150 52 IMU_GYRO -0.604248 -0.142578 -0.311279
0	0	0	0	2815.1150 52 IMU_ACCEL 0.178711 -0.672852 -5.609375
0	0	0	0	2815.1320 52 IMU_MAG 0.000000 0.000000 0.000000
0	0	1	0	2815.1320 52 IMU_GYRO -0.646484 -0.125488 -0.298340
0	0	0	0	2815.1320 52 IMU_ACCEL -0.178711 -0.291992 -6.009766

Figure 5-16: Indexing SETTINGS, to find the fault and nominal flight intervals, indexing GYRO measurements to AND with FAULT indexes to find the indexes of faulty gyro measurements

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

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

This work investigates only binary classification, rather than multi-class classification considering multiple different faults. However, different faulty cases have been studied to further attack the problem of multi-class classification in future studies. The classifier's abilities in different fault conditions have been investigated. The main two types of faults of interest in this study are the control surface stuck fault and the LOE. The flight data[4] and code[5] that have been developed are publicly available under *Github*.

All three steps (training, tuning and evaluation) explained thoroughly in the SVM application section, have been applied and the results are given below.

5.2.4 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 a manual recovery by triggering the safety switch, shown in Fig. 5-11. So the time taken from starting the ignition of the control surface stuck fault until the manual pilot's intervention was very short (∼2 seconds), which can be seen from the accelerometer x-direction readings in Fig. 5-17.

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

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

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

special treatment, as explained above under Section 4.2.2.

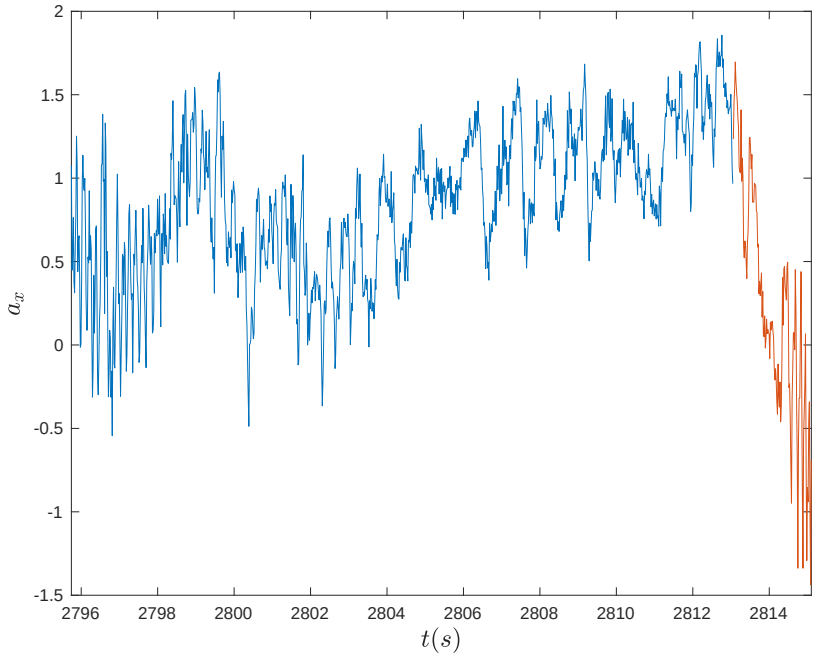

Figure 5-17: Accelerometer readings along x-direction during flight interval just before control surface stuck (nominal, represented with blue) and during control surface stuck at 0° until the safety pilot's intervention

The first problem considered for classification is the right elevon stuck at 0°. Gyro and accelerometer measurements are saved to onboard SD card at 60 Hz. Nominal class involves ∼5 minutes of accelerometer and gyro data while the faulty class comprises ∼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 dependent on the level of the engineer's experience, since some of the critical decisions are handled by her/him such as selection of features (though not for all machine learning methods). Fig. 5-17 shows accelerometer x-axis measurements for a duration of ∼20 seconds. Measurements plotted in blue indicate the part of the flight where the elevon works correctly, while the red part of the plot corresponds to the faulty phase. It can be interpreted from the Fig. 5-17 that the fault injected shows a distinct change in the x-axis ac-

celerometer measurements when the fault is injected.

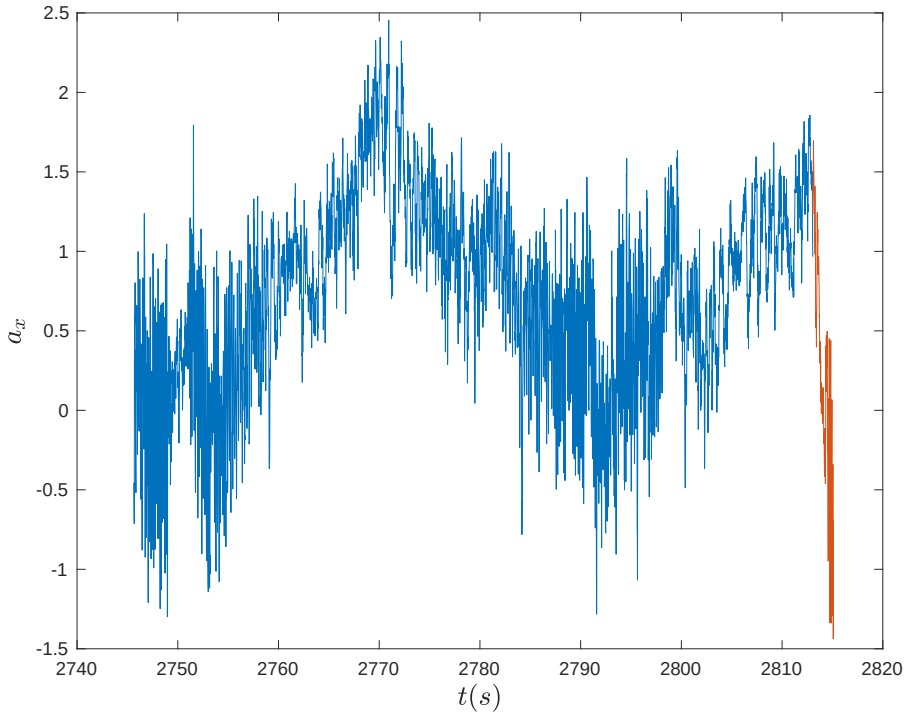

Figure 5-18: Accelerometer readings along x-direction during flight interval before control surface stuck (nominal, represented with blue) and during control surface stuck at 0° until the safety pilot's intervention

To investigate further, measurements have been plotted during a larger time scale involving ∼70 seconds of measurements, as shown in Fig. 5-18. Here, it is seen that the accelerometer measurements corresponding to the faulty phase could be observed during the nominal flight phase as well, thus it may be necessary to check the time change of translational acceleration to observe a difference in behavior.

Fig. 5-19 and Fig. 5-20 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 the 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. 5-21 and Fig. 5-22. It shows that a_x-a_z space gives a distribution that would

make it 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.

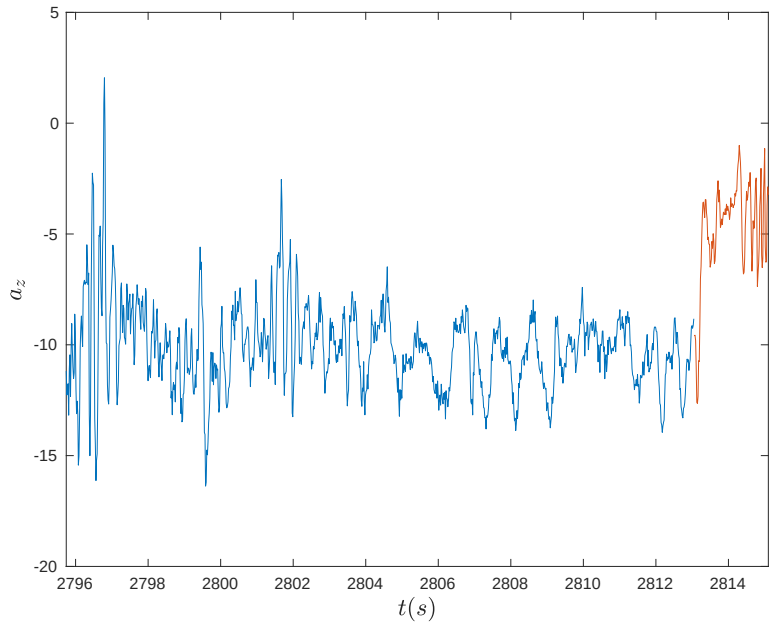

Figure 5-19: Accelerometer measurements along z direction a_z for faulty and nominal flight data

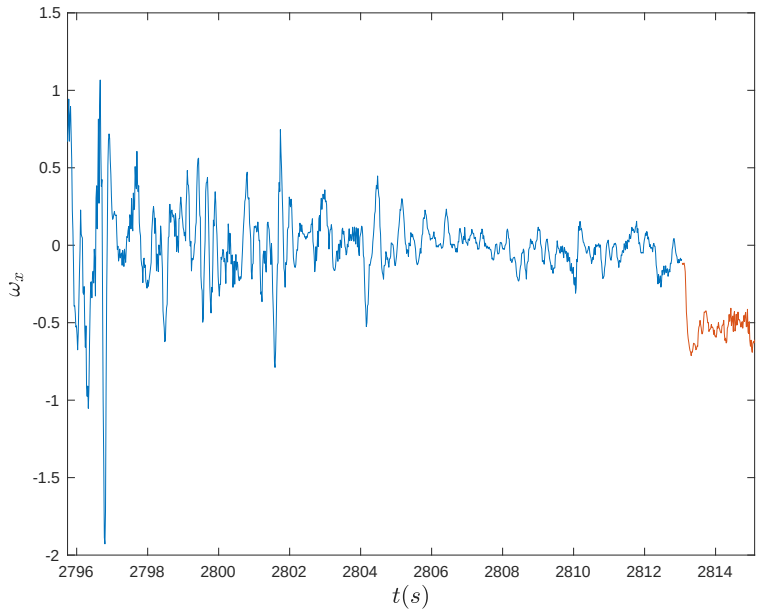

Figure 5-20: Angular velocity w_x for faulty nominal flight data

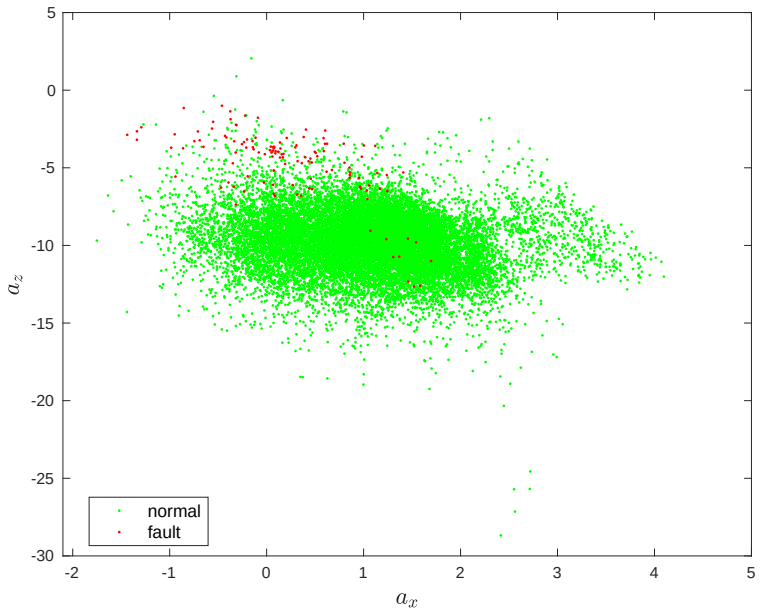

Figure 5-21: a_x vs a_z feature space for faulty nominal flight data

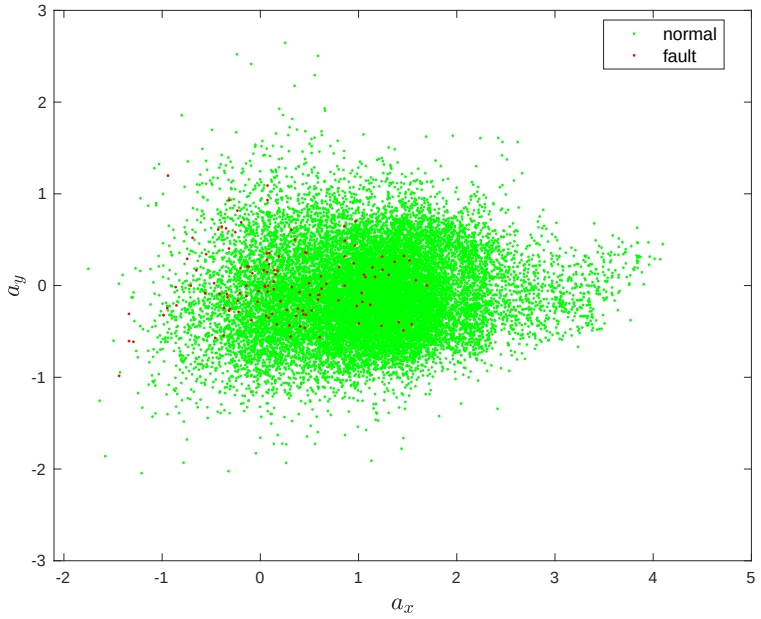

Figure 5-22: a_x vs a_y feature space for faulty nominal flight data

Table 5.3 shows the results of SVM classification for the untuned linear kernel, the untuned Gaussian kernel, and the tuned Gaussian kernel via two methods (heuristic and Bayesian), respectively, in its columns. Although a variety of variables used in

Table 5.3: 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 x 10^{-2}	1.2 x 10^{-3}	7.571 x 10^{-4}	1.2 x 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

the evaluation of classifiers have been presented to the reader for completeness, the 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).

To introduce the time change behavior of the system, the feature set has been widened with the additions of data from past measurements for each feature. We consider one of the features (shown in Fig. 5-23), accelerometer data in x-axis, corresponding to one of six columns (one of six features) in the feature matrix given in Fig. 5-5.

$$A_x = \begin{bmatrix} acc_x^t \\ acc_x^{(t+1)} \\ acc_x^{(t+2)} \\ \vdots \\ acc_x^{(t+m-1)} \end{bmatrix}$$

Figure 5-23: One of the features (accelerometer x-axis) of the original feature matrix (given in Fig. 5-5) before adding extra features

$$A_x^{'} = \begin{bmatrix} acc_x^{(t-N+1)} & \cdots & acc_x^{(t-1)} & acc_x^{t} \\ acc_x^{(t-N+2)} & \cdots & acc_x^{t} & acc_x^{(t+1)} \\ acc_x^{(t-N+3)} & \cdots & acc_x^{(t+1)} & acc_x^{(t+2)} \\ \vdots & \ddots & \vdots & \vdots \\ acc_x^{(t-N+m)} & \cdots & acc_x^{(t+m-2)} & acc_x^{(t+m-1)} \end{bmatrix}$$

Figure 5-24: Addition of features to involve $N-1$ previous measurements

Finally, this feature addition is applied to all features of the original feature matrix, resulting in $\boldsymbol{X'} \in \mathbb{R}^{\mathrm{m \times 6N}}$:

$$\boldsymbol{X'} = \left[\boldsymbol{A}_x^{'} \ \boldsymbol{A}_y^{'} \ \boldsymbol{A}_z^{'} \ \boldsymbol{G}_x^{'} \ \boldsymbol{G}_y^{'} \ \boldsymbol{G}_z^{'} \right]^{\mathrm{T}} \tag{5.3}$$

where $\boldsymbol{A}_y^{'}$, $\boldsymbol{A}_z^{'}$, $\boldsymbol{G}_x^{'}$, $\boldsymbol{G}_y^{'}$, $\boldsymbol{G}_z^{'}$ are constructed in the same way as $\boldsymbol{A}_x^{'}$ detailed in Fig. 5-24. For N = 4, the number of features have been increased to 24 in Table 5.4, which means only three 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, has deteriorated the performance of the classifier, especially for the classifier with an untuned Gaussian kernel (*f1Score*= 0.4167) ($\Delta f1Score = -0.5135$). For the classifiers with tuned Gaussian kernels, the *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 very obvious.

Table 5.4 and first three columns of Table 5.5 are the result of the same algorithm with same number of features and percentage of training/test set ratio. Due to the usage of random functions, the seed value has been set to a constant value for reproducibility. Stratified sampling has been implemented in order to ensure a reasonable amount of data from both classes in both sets (training and test). Finally, it has been discovered that even with the selection of the 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

Table 5.4: 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 x 10^{-3}	3.44 x 10^{-4}	6.19 x 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

seen by observing the tuned *f1Scores* (*f1Scores*=0.9444 in and *f1Score*= 0.9143) in Table 5.4 increasing to the *f1Score* (*f1Score*= 1) in Table 5.5.

Table 5.5 shows the evaluation results of the SVM classifiers trained with a different percentage of training and test sets. The first three columns present the results for a test to training set ratio of 1/4 while the last three columns present the ratio of 2/3. The *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).

5.2.5 Control surface loss of efficiency fault

The 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 the first minutes of the flight, the nominal mode was kept long so as to gather more nominal data from the flight before initiating the fault sequences. The results showed the insufficiency of the method to diagnose the fault (see Table 5.6 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 that since the control surface is not moving in a wide

Table 5.5: 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 x 10^{-3}	4.8 x 10^{-4}	6.8 x 10^{-4}	5.4 x 10^{-3}	5.5 x 10^{-4}	7.3421 x 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

Table 5.6: %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

range during the nominal course of action, a 10% degradation in the movement of the elevon does not noticeably change the dynamic behavior of the drone, and this is not easy to establish 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 re-computes the control signal to minimize this error. So, due either to due to a different wind condition error or a fault in the control surface, the controller's duty is to minimize this error, and a well-designed controller might handle that especially in the condition where the effect of disturbance is not catastrophic. This controller's compensation for the fault could be known by observing the output of the controller and feeding this information, as well the set of instances. However, during the course of this study, the abilities of a classifier without the information from the controller is investigated. The main reason behind this is to challenge the abilities of a small and inexpensive set of health monitoring system that does not rely on information from the autopilot.

The loss of efficiency of the left elevon is increased to 40% degradation in order to investigate if, in this case, the SVM classifier will be able to classify the fault. This fault is coined as LOE2 Table 5.6. The results show no recuperation in the classification results for either untuned or tuned classifiers, with $f1Score = 0.0032$ and $f1Score = 0.2$, respectively.

Table 5.7: %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

Since the results were very poor in terms of classifying faults, the faults have been increased to have a 40% degradation in both elevons. The idea is to see at which level the result of classification will improve, and also to explore any reasons behind this ineffectiveness, except for the controller's compensation. Another issue to investigate is the effect of the number of instances for the larger datasets (nominal class) on classification. For that purpose, three different sizes for nominal data set (∼15min, ∼6min, ∼3min) were investigated in terms of their effects on classification performance for a faulty measurement data set of ∼1min. Classification results for two of the situations (∼15min, ∼3min) are given in Table 5.7. The reduction of the nominal data set was not a random selection throughout the complete set, but was only for the last part of the nominal data set, which is closer to the faulty data in terms of flight time instance 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 5.7. Reducing the number of the larger class, gyro and accelerometer measurements during nominal phase, an untuned SVM classifier with Gaussian kernel still shows poor results in classification ($f1Score = 0.0073$), and tuning does not provide much help ($f1Score = 0.13$). 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

Table 5.8: %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

further reduction in the number 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 is 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 to a certain extent but does 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 are added to the input matrix to give 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 are shown in Table 5.8. We now discuss the effect of adding previous measurements as separate features to the feature set (previous, three measurements were added to the input data set), resulting in 4 x 6 = 24 features. Adding three previous instance in time for each sensor measurement decreased the performance of the untuned classifier with a Gaussian kernel $f1Score = 0.1375$, which is classification performance decrease of $\Delta f1Score = 0.1341$. By adding more features, the classifier was still not able to classify when untuned, but by tuning, the *f1Score* can be improved noticeably, producing a classifier with an *f1Score* of 0.9635 for 120 feature case and can be even more refined to an *f1Score* of 0.9896 for 300 features in total. So, adding features advances the classification but only when proper tuning is achieved. Otherwise, untuned, the

results are even worse than before adding the features.

5.2.6 Use of spinors as attributes

In this thesis, until now, the use of feature engineering to ameliorate the classification performance has been performed mainly by utilizing the addition of new features to include previous measurements in all instances. The purpose was to include the time evolution of the signal, as well as the instantaneous measurements in the instances. Here, another idea has been applied as feature engineering: to replace angular velocity measurements from gyros with spinors, such that a feature matrix is given, as in Fig. 5-25.

$$X = \begin{bmatrix} acc_x^t & acc_y^t & acc_z^t & spinor_x^t & spinor_y^t & spinor_z^t \\ acc_x^{(t+1)} & acc_y^{(t+1)} & acc_z^{(t+1)} & spinor_x^{(t+1)} & spinor_y^{(t+1)} & spinor_z^{(t+1)} \\ \vdots & & & & & \\ acc_x^{(t+m-1)} & acc_y^{(t+m-1)} & acc_z^{(t+m-1)} & spinor_x^{(t+m-1)} & spinor_y^{(t+m-1)} & spinor_z^{(t+m-1)} \end{bmatrix}$$

Figure 5-25: New feature matrix when spinors are used as features rather than gyro measurement

To calculate the spinors, the kinematic equations given in Eq. 5.4 are solved numerically to attain quaternions from angular velocities:

$$\begin{aligned} \dot{q}_0 &= -\frac{1}{2}\boldsymbol{q}_\nu^T\boldsymbol{\omega} \\ \dot{\boldsymbol{q}}_\nu &= \frac{1}{2}\Big(\boldsymbol{q}_\nu^\times + q_0\boldsymbol{I}_3\Big)\boldsymbol{\omega} \end{aligned} \tag{5.4}$$

Given the quaternion definition in Eq. 5.5:

$$\boldsymbol{q} = \begin{bmatrix} cos\Big(\frac{\theta}{2}\Big) & sin(\frac{\theta}{2})\boldsymbol{e} \end{bmatrix} \tag{5.5}$$

the spinor is calculated via taking the logarithm of quaternion described as in Eq.

5.6:

$$log(\boldsymbol{q}) = \begin{bmatrix} 0 & \frac{\theta}{2}\boldsymbol{e} \end{bmatrix} \tag{5.6}$$

The result given in Table 5.9 shows an improvement in classification with an *f1Score* reaching up to 0.9915.

Table 5.9: Results for untuned, tuned SVM classifiers with spinors as attributes

	Untuned Gaussian kernel	Untuned linear kernel	Tuned heuristic Gaussian kernel	Tuned Bayesian Gaussian Kernel
f1Score	0.9555	0.6878	0.9795	0.9915
kFoldLoss	0.0213	0.1047	0.0098	0037
precision	0.9618	0.9758	0.9704	0.9925
recall	0.9492	0.5311	0.9887	0.9906
boxConstraint	1	1	10^5	9.6 x 10^4
kernelScale	1	1	5.6611	3.8346
compTime	5.17s	5.47s	11470.03s	11373.91

Untuned classification superiority

The first promising result by using spinors for classification is its advantage in untuned classifiers. This can be seen by checking the *f1Score* of the SVM classifier without tuning (see Table 5.10). As mentioned before, the untuned classifier for added features gives poor results, although it increases the classification accuracy when tuned properly. An untuned SVM classifier with spinors as attributes results in an increased classification performance, with an *f1Score* of 0.6878. Using a Gaussian Kernel which is mathematically represented as in Eq. 5.7, the *f1Score* even increases up to 0.9555.

$$K(x,z) = exp\left(- \frac{\|x-z\|^2}{2\sigma^2} \right) \tag{5.7}$$

Tuned classification Bayesian optimization efficiency

Table 5.11 shows the results, except for the simulations using spinors. The heuristic optimization gives a better performance for tuning the classifier, while the Bayesian optimization is likely to converge to a local minima. However, when spinors used,

Table 5.10: Results for untuned SVM classifiers with added features, original features and spinors as attributes

	Untuned Gaussian kernel 300 feat.	Untuned Gaussian kernel 24 feat.	Untuned Gaussian kernel original feat.	Untuned Linear kernel spinors	Untuned Gaussian kernel spinors
f1Score	NaN	0.1375	0.2712	0.6878	0.9555

Bayesian optimization results in a better performance in tuning the classifier, as shown in Table 5.11 in the Tuned Gaussian Kernel spinors row.

Table 5.11: Results for tuned SVM classifiers with added features, original features and spinors as attributes

	Heuristic tuning	Bayesian tuning
Tuned Gaussian kernel 300 feat.	0.98	0.92
Tuned Gaussian kernel 24 feat.	0.64	0.35
Tuned Gaussian kernel original feat.	0.2756	NaN
Tuned Gaussian kernel spinors	0.97	0.99

5.3 Conclusion

In this chapter, we focus on the results of SVM classification application to the fault detection and diagnosis problem. Fault classification simulations are explained under two main sections: classification of faults based on simulated flight measurements; and the classification of faults based on real flight data.

The first part of this chapter gives the results of SVM classification on data generated from simulations. To simulate the data, equations of motions given in *Nonlinear Aircraft Model* have been numerically solved for the states. Then sensor measurements (accelerometer and gyro data) have been calculated using the states and the specifications of the sensors. Generated data is usually more structured compared to the real flight data. In this preliminary application of SVM to fault diagnosis,

we start with an easier problem, and use data generated from models. There is no controller involved in the model in this preliminary application of SVM to detection, in order to discard the controller's effect on the diagnosis.

The second part investigates fault detection with real flight data. SVM, being a supervised learning algorithm, requires labeled data for both nominal and faulty flight conditions. Thus, faulty flights were realized with a security pilot ready for recovery in the event of loss of control of the aircraft. For the faulty flight data gathering, some modifications to the *Paparazzi* autopilot was necessary in two main parts: first, injecting the faults real-time from GCS; and second, and editing the onboard controller so that the faults injected reconfigures the servos as manipulated from the GCS.

With the flight data, two main classes of faults — control surface stuck and loss of effectiveness — have been investigated separately. A variety of techniques have been implemented to improve the performance of the classification, such as feature engineering and tuning the classifiers.

Chapter 6

Conclusion

The integration of drones into the airspace demands the introduction of innovative designs to provide safe solutions for drones. One aspect of this issue is to ensure safe flight by designing fault detection and diagnosis systems with less expensive avionics, common in a vast number of drones. This work aims to design a classifier via SVM to solve FDD for drones with actuator faults. For that purpose, we introduce an end-to-end design to achieve data-driven fault diagnosis for the control surface faults in drones. All the data and the software code are available in the code sharing and versioning system *Github*.

In this thesis, fault classification simulations are investigated under two main sections: first, the classification of faults based on simulated flight measurements; and second, the classification of faults based on real flight data. We started with the easier problem: classification of faults based on simulated flight measurements.

For the classification problem using data generated by simulations, a model of a MAKO Unmanned Aerial Vehicle (UAV) is simulated. Sensor measurements (accelerometer and gyro data) are simulated using the information on the drone's motion and the specifications of the real sensors. Generated data is usually more structured compared to real flight data. There are no flight control loops involved in the model: discarding the controller's effect eases the diagnosis. The results show that the SVM classifier is accurate and fast in diagnosing the fault on the control surfaces, with a classification accuracy of 10^{-5}.

Next, fault detection with real flight data is investigated. Since SVM is a supervised classification method, labeled data is necessary in order to train the algorithm. For this reason, real flights are arranged to generate faulty flight data by manipulating the open source autopilot, *Paparazzi*. Training is held offline due to the need for labeled data and the computational burden of the tuning phase of the classifiers. Two types of faults are the focus of the investigation: a stuck elevon fault and the loss of effectiveness of the elevon. Results indicate that the control surface stuck fault can be detected relatively easily with three gyros and three accelerometer measurements, compared to the loss of effectiveness fault. The results show that over the flight data, tuned SVM yields an F1 score of 0.98 for the classification of control surface faults. The addition of features to accommodate the previous measurements improves the classification performance for tuned classifiers, while the untuned classification performance deteriorates. Classification performs poorly for the loss of efficiency faults, especially for small losses of effectiveness. For the loss of efficiency fault, some feature engineering — involving the addition of past measurements — is needed in order to attain similar classification performance.

A very promising result is discovered when *spinors* are used as features instead of angular velocities. Thus, the kinematic equations have been solved so as to calculate the *quaternions* using angular velocity measurements. Then *spinors* are calculated as a function of *quaternions*. Results show that by using spinors for classification, there is a vast improvement in classification accuracy, especially when the classifiers are untuned. Using spinors and a Gaussian Kernel, the untuned classifiers give an F1 score of 0.9555 which was 0.2712 when the gyro measurements were used as features.

This work thus shows that SVM yields a satisfactory performance levels for the classification of faults on the control surfaces of a drone using real flight data. The FDD algorithm designed here is a first step towards safer drone flights. The next section introduces possible future steps.

6.1 Future Work

For each failure type — jammed elevon or ineffective elevon — this thesis applies a two class classification problem: faulty data and nominal data. Future work should consider multi-class classification since it will lead to a more realistic application of fault diagnosis for drones; but the problem could be very complicated due to the vast number of possible faults. For such a problem, methods such as deep learning could be selected instead of SVM, since deep learning offers appealing performance on classification problems with a large number of classes. Another possible workaround could be to label the faults as severe, moderate and mild. Then, rather than determining the exact *nature* of the fault, at least an awareness of the *severity* of the fault could be gained. Fault Detection and Diagnosis should also be complemented by recovery. The abilities of a drone after a fault should be assessed and a control action should be taken to mitigate the faulty situation. If recovery is not a viable option, a ditching maneuver could be undertaken to reduce the harm on the ground. Knowledge of the fault's severity could be used to choose an adequate mitigation measure.

Appendix A

Codes for FD from simulated data

A.1 Read Me file for the aircraft simulation codes in Matlab

HOW TO:

1. run simDrone.m
2. If you want to change aircraft parameters, change them from configDrone.m.
3. If you want to change simulation time, change $sim\_duration\_min$ in simDrone.m

ASSUMPTIONS:

1. NED (North East Down) navigation frame is inertial where Newton's law apply.
2. Attitude sequence considered is Yaw-Pitch-Roll. And yes sequence matters!

INFO FOR BEGINNERS: Sequential Euler angle rotations relate the orientation of the aircrafts body-fixed frame to the navigation frame. For simulations quaternion preferred due to singularity issues of Euler angles. It is proven that attitude representations will either have a redundant component (4 components) as in quaternions or singularity (Euler angles have 3 components resulting in singularity for

pitch (theta) = 90 degrees - devision by 0). Quaternion representation used here has the scalar component as the first component: q = q0 + q1 * i + q2 * j + q3 k

w $\triangleq$ angular velocity vector with components p, q, r w = [p q r]' in detail, w describes the angular motion of the body frame b with respect to navigation frame n (NED), expressed in body frame.

Attitude transformation matrix (Direction Cosine Matrix) is used to change the frame of interest that the vector or points expressed in. Lets say that we have a vector A fixed in inertial frame. Its representation in two frames will differ even it is the same vector. The frame to express it could be changed utilizing a direction cosine matrix. In this work, to express vectors in different frames is necessary which makes the use of DCM essential. C_n^b transforms the vector A expressed in the navigation frame A^n into A^b, a vector expressed in the drone body-fixed frame. $A^b = C_n^b * A^n$ ——> in the code : $c\_n\_to\_b$ Likewise a direction cosine matrix C_b^n, changes the representation of vector A expressed in drone body-fixed frame A^b, to a representation of the same vector A in navigation frame(NED) A^n. $A^n = C_b^n * A^b$ and beware the relationship between these to transformation: $C_b^n = inverse(C_n^b) = transpose(C_n^b)$

TIPS AND TRICKS: Forces and Moments in the dynamical equations of motion for the attitude and translational motion, are calculated in modelDrone. During the numerical integration of dynamic and kinematic equations (more specifically during the function evaluations for calculating k1 k2 k3 k4), modelDrone is evaluated with different states (due to the nature of RK4). Since forces and moments are calculated in this model, beware that they are evaluated with different state values, since they are modeled as a function of states, ending up the change of forces and moments during one time step of numerical integration.

A.2 configDrone.m

```
% Copyright 2016 Elgiz Baskaya

% This file is part of curedRone.

% curedRone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.
clear all;
clc;

global g_e mass inert wing_tot_surf wing_span m_wing_chord prop_dia nc
    tho_n cl_alpha1 cl_ele1 cl_p
global cl_r cl_beta cm_1 cm_alpha1 cm_ele1 cm_q cm_alpha cn_rud_contr
    cn_r_tilda cn_beta cx1
global cx_alpha cx_alpha2 cx_beta2 cz_alpha cz1 cy1 cft1 cft2 cft3

% Earth gravitational constant
g_e = 9.81;

% UAV mass [kg]
mass = 28;
```

```
% UAV inertia matrix [kg*m^2]
inert = [2.56 0 0.5; 0 10.9 0; 0.5 0 11.3];

% wing total surface S [m^2]
wing_tot_surf = 1.8;

% wing span b [m]
wing_span = 3.1;

% mean aerodynamic wing chord c [m]
m_wing_chord = 0.58;

% diameter of the propeller prop_dia [m]
prop_dia = 0.79;

% engine speed reference signal nc
nc = 80;

% time constant of the engine tho_n [s]
tho_n = 0.4;

% roll derivatives
cl_alpha1 = - 3.395e-2;% cl_alpha2 = - clalpha1
cl_ele1 = - 0.485e-2;% cl_ele2 = - clele1
cl_p = - 1.92e-1;
cl_r = 3.61e-2;
cl_beta = - 1.3e-2;

% pitch derivatives
cm_1 = 2.08e-2;
cm_alpha1 = 0.389e-1;% cmalpha2 = cmalpha1
```

```
cm_ele1 = 2.725e-1;% cmele2 = cmele1
cm_q = -9.83;
cm_alpha = -9.03e-2;

% yaw derivatives
cn_rud_contr = 5.34e-2;
cn_r_tilda = -2.14e-1;
cn_beta = 8.67e-2;

% lift, drag, side force derivatives
cx1 = -2.12e-2;
cx_alpha = -2.66e-2;
cx_alpha2 = -1.55;
cx_beta2 = -4.01e-1;
cz_alpha = -3.25;
cz1 = 1.29e-2;
cy1 = -3.79e-1;

% thrust derivatives
cft1 = 8.42e-2;
cft2 = -1.36e-1;
cft3 = -9.28e-1;
```

A.3 modelDrone.m

```
% Copyright 2016 Elgiz Baskaya

% This file is part of curedRone.

% curedRone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

% inputs : stateInitial .:. States from the previous time t - 1.
%          VT .:. total airspeed of the aircraft
%          conAil1, conAil2, conEle1,conEle2, conRud .:. controlTorques

% Attitude kinematic and dynamic equations of motion
% Translational Motion

function state_dot = modelDrone(state_prev, contr_deflect, wind_ned)

global g_e inert mass wing_tot_surf wing_span m_wing_chord prop_dia
    cl_alpha1 cl_ele1 cl_p
global cl_r cl_beta cm_1 cm_alpha1 cm_ele1 cm_q cm_alpha cn_rud_contr
    cn_r_tilda cn_beta cx1
```

```
global cx_alpha cx_alpha2 cx_beta2 cz_alpha cz1 cy1 cft1 cft2 cft3 tho_n nc

quat_normalize_gain = 1;

% q .:. quaternion
% q = q0 + q1 * i + q2 * j + q3 * k;
q0 = state_prev(1);
q1 = state_prev(2);
q2 = state_prev(3);
q3 = state_prev(4);

% w .:. angular velocity vector with components p, q, r
% w = [p q r]'
% w describes the angular motion of the body frame b with respect to
% navigation frame ned, expressed in body frame.
p = state_prev(5);
q = state_prev(6);
r = state_prev(7);

% x .:. position of the drone in North East Down reference frame
% x = [x_n y_e z_d]';
% x_n = state_prev(8);
% y_e = state_prev(9);
% z_d = state_prev(10);

% v .:. translational velocity of the drone
% v = [u_b v_b w_b]
u_b = state_prev(11);
v_b = state_prev(12);
w_b = state_prev(13);

% Engine speed
```

```
eng_speed = state_prev(14);

% Flight altitude
altitude = state_prev(10);

% Control surface deflections
con_ail1 = contr_deflect(1);
con_ail2 = contr_deflect(2);
con_ele1 = contr_deflect(3);
con_ele2 = contr_deflect(4);
con_rud = contr_deflect(5);

% If A is any vector
% A^n = C^n_b * A^b = c_b_to_n * A^b = inv(c_n_to_b) * A^b = c_n_to_b' * A^b
% Direction cosine matrix C^b_n representing the transformation from
% the navigation frame to the body frame
c_n_to_b = [1 - 2 * (q2^2 + q3^2) 2 * (q1 * q2 + q0 * q3) 2 * (q1 * q3 - q0
    * q2); ...
2 * (q1 * q2 - q0 * q3) 1 - 2 * (q1^2 + q3^2) 2 * (q2 * q3 + q0 * q1); ...
2 * (q1 * q3 + q0 * q2) 2 * (q2 * q3 - q0 * q1) 1 - 2 * (q1^2 + q2^2)];

vel_t = [u_b; v_b; w_b] - c_n_to_b * wind_ned;

% Total airspeed of drone
vt = sqrt(vel_t(1)^2 + vel_t(2)^2 + vel_t(3)^2);

% alph .:. angle of attack
alph = atan2(vel_t(3),vel_t(1));

% bet .:. side slip angle
bet  = asin(vel_t(2)/vt);
```

```
% Low altitude atmosphere model (valid up to 11 km)
% t0 = 288.15; % Temperature [K]
% a_ = - 6.5e-3; % [K/m]
% r_ = 287.3; % [m^2/K/s^2]
% p0 = 1013e2; %[N/m^2]
% t_= t0 * (1 + a_ * altitude / t0);
% ro = p0 * (1 + a_ * altitute /t0)^5.2561 / r_ / t_;
t_ = 288.15 * (1 - 6.5e-3 * altitude / 288.15);
ro = 1013e2 * (1 - 6.5e-3 * altitude / 288.15)^5.2561 / 287.3 / t_;
dyn_pressure = ro * vt^2 / 2;

p_tilda = wing_span * p / 2 / vt;
r_tilda = wing_span * r / 2 / vt;
q_tilda = m_wing_chord * q / 2 / vt;

% calculation of aerodynamic derivatives
% (In the equations % CLalpha2 = - CLalpha1 and so on used not to inject
    new names to namespace)
cl = cl_alpha1 * con_ail1 - cl_alpha1 * con_ail2 + cl_ele1 * con_ele1 -
    cl_ele1 * con_ele2 ...
+ cl_p * p_tilda + cl_r * r_tilda + cl_beta * bet;
cm = cm_1 + cm_alpha1 * con_ail1 + cm_alpha1 * con_ail2 + cm_ele1 *
    con_ele1 + cm_ele1 * con_ele2 ...
+ cm_q * q_tilda + cm_alpha * alph;
cn = cn_rud_contr * con_rud + cn_r_tilda * r_tilda + cn_beta * bet;

l = dyn_pressure * wing_tot_surf * wing_span * cl;
m = dyn_pressure * wing_tot_surf * m_wing_chord * cm;
n = dyn_pressure * wing_tot_surf * wing_span * cn;

moment = [l m n]';
```

```
% tilda is to ignore output of the quat2angle function, since it is not
% used, a warning appears otherwise
[~, teta fi] = quat2angle([q0 q1 q2 q3]);

% ft .:. thrust force
ft = ro * eng_speed^2 * prop_dia^4 * (cft1 + cft2 * vt / prop_dia / pi /
    eng_speed + ...
                            cft3 * vt^2 / prop_dia^2 / pi^2 / eng_speed^2);
% Model of the aerodynamic forces in wind frame
% xf_w .:. drag force in wind frame
xf_w = dyn_pressure * wing_tot_surf * (cx1 + cx_alpha * alph + cx_alpha2 *
    alph^2 + ...
                             cx_beta2 * bet^2);
% yf_w .:. lateral force in wind frame
yf_w = dyn_pressure * wing_tot_surf * (cy1 * bet);
% zf_w .:. lift force in wind frame
zf_w = dyn_pressure * wing_tot_surf * (cz1 + cz_alpha * alph);

% describe forces in body frame utilizing rotation matrix c^w_b
% A^w = C^w_b * A^b   OR    A^b = C^b_w * A^w = (C^w_b)' * A^w here
% c_b_to_w = C^w_b
c_b_to_w = [cos(alph)* cos(bet) sin(bet) sin(alph) * cos(bet);...
-sin(bet) * cos(alph) cos(bet) -sin(alph) * sin(bet); -sin(alph) 0
    cos(alph)];

force = mass * [- g_e * sin(teta); g_e * sin(fi) * cos(teta); g_e * cos(fi)
    * cos(teta)] +...
    ([ft; 0; 0] + c_b_to_w' * [xf_w; yf_w; zf_w]);

% Kinematic and dynamic equations of motion of the drone

% Attitude dynamics of drone
```

```
% Skew symmetric matrix is used for cross product
pqr_dot = inert \ (moment - [ 0 -r q; r 0 -p; -q p 0] * (inert * [p q r]'));

% Attitude kinematics of drone
q_dot = 1 / 2 * [-q1 -q2 -q3; q0 -q3 q2; q3 q0 -q1; -q2 q1 q0] * [p q r]'
    ...
    + quat_normalize_gain * (1 - (q0^2 + q1^2 + q2^2 + q3^2)) * [q0 q1 q2
        q3]';

% x_dot is in NED frame. So a change in expression of v is needed.
x_dot = c_n_to_b' * [u_b v_b w_b]';

% dynamics for translational motion of the center of mass of the drone
v_dot = force / mass - [(q * w_b - r * v_b); (r * u_b - p * w_b); (p * v_b
    - q * u_b)];

% dynamics for engine speed
eng_speed_dot = - 1 / tho_n * eng_speed + 1 / tho_n * nc;

state_dot = [q_dot; pqr_dot; x_dot; v_dot; eng_speed_dot];
```

A.4 quat_to_euler.m

```
% Copyright 2016 Elgiz Baskaya

% This file is part of curedRone.

% curedRone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

function euler_ang = quat_to_euler(quater)

euler_ang = [atan2(2 .* (quater(3,:) .* quater(4,:) - quater(1,:) .*
    quater(2,:)), ...
    2 * ((quater(1,:)).^2 - 1 + (quater(4,:)).^2));...
    - atan((2 * (quater(2,:) .* quater(4,:) + quater(1,:) .* quater(3,:)))
        ./ sqrt(1 - (2 * (quater(2,:) .* quater(4,:) + quater(1,:) .*
        quater(3,:))).^2)); ...
    atan2(2 * (quater(2,:) .* quater(3,:) - quater(1,:) .* quater(4,:)), ...
    2 * ((quater(1,:)).^2 - 1 + 2 * (quater(2,:)).^2))];
```

A.5 rungeKutta4.m

```
% Copyright 2016 Elgiz Baskaya

% This file is part of curedRone.

% curedRone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

function xn = rungeKutta4(func, xo, cntrl, wind_ned, h)

k1 = feval(func, xo, cntrl, wind_ned);
k2 = feval(func, xo + 1/2 * h * k1, cntrl, wind_ned);
k3 = feval(func, xo + 1/2 * h * k2, cntrl, wind_ned);
k4 = feval(func, xo + h * k3, cntrl, wind_ned);

xn = xo + 1/6 * h * (k1 + 2 * k2 + 2 * k3 + k4);
end
```

A.6 simDrone.m

```
% Copyright 2016 Elgiz Baskaya

% This file is part of curedRone.

% curedRone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

% DRONE DYNAMICS SIMULATION

configDrone;

ti = 0.1;
sim_duration_min = 10;
tf = 60 * sim_duration_min;
t_s = 0 : ti : tf;

x_real = zeros(14,length(t_s));

% initial condition for the states
```

```
% xReal = [q0 q1 q2 q3 p q r x_n y_e z_d u_b v_b w_b eng_speed]';
x_real(:,1) = [1 0 0 0 0 0 0 0 0 0 1e-5 1e-5 1e-5 1e-2]';

control_deflections = [0 0 0 0 0]';
% controlTorque = [contAileron1 contAileron2 contElevator1 contElevator2
% contRudder]'

wind_ned = [0 0 0]';
% wind_ned .:. [wind_n wind_e wind_d]'

for i=1:length(t_s)-1
  % Nonlinear attitude propagation
  % Integration via Runge - Kutta integration Algorithm
  x_real(:,i+1) = rungeKutta4('modelDrone', x_real(:,i),
      control_deflections, wind_ned, ti);
end
```

Appendix B

Codes for FD from flight data

B.1 dataRead.m

```
% Written by Ewoud Smeur
% Modified by Elgiz Baskaya

%%%%%%% This part from Ewoud %%%%%%%%
% filename = '17_04_20__14_22_51_SD.data'; % Mulitplicative fault only
filename = '17_07_06__10_21_07_SD.data'; % Muret with Michel
% filename = '17_09_07__10_07_55_SD.data';

formatSpec = '%f%f%s%f%s%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f%f';

formatSpecHeader = '%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%s%[^\n\r]';
delimiter = ' ,';
startRow = 1;
fileID = fopen(filename,'r');
header = textscan(fileID, formatSpecHeader,1, 'Delimiter', delimiter,
    'EmptyValue' ,NaN);
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter,
    'EmptyValue' ,NaN,'HeaderLines' ,startRow, 'ReturnOnError', false);
```

```
fclose(fileID);

N = length(dataArray{1, 1})-1;

%%%%%% This part from Elgiz %%%%%%%%

% Selecting the drone with whose data you want to work with
index_drone_select = find(dataArray{1,2}==52);
drone_select_id = zeros(length(dataArray{1,1}),1);
% Set indexes to 1s if it is the drone of interest
drone_select_id(index_drone_select) = 1;

% Finding flight interval
index_altitude = find(dataArray{1,8}>190000);
altitude_limit_id = zeros(length(dataArray{1,8}),1);
% Sets indexes to 1s if the altitude is greater than the given limit
altitude_limit_id(index_altitude) = 1;

% dataArray{1,8} can be something else then GPS data so select the GPS
% indexed as well
gps_id = strcmp(dataArray{1,3},'GPS');

% find out the first time of pass of the altitude limit of drone of interest
% and last time of pass of the altitude limit of the drone of interest
% And indexes i. if it is GPS data
%           ii. if it is greater than the altitude of interest
%          iii. if it is drone of interest

index_drone_gps_alt = altitude_limit_id & gps_id & drone_select_id;
first_altPass = find(index_drone_gps_alt, 1, 'first');
last_altPass = find(index_drone_gps_alt, 1, 'last');
```

```
% All the times in between the first passing of altitude limit and last
% passing of the altitude limit are assumed to be the flight duration
flight_duration_id = zeros(length(dataArray{1,1}),1);
flight_duration_id(first_altPass:last_altPass) = 1;

%%%%%%%% Ewoud again ! %%%%%%%%%%%%
array_col_5 = zeros(length(dataArray{1, 5}),1);
for i = 1:length(dataArray{1, 5})
    try
        array_col_5(i) = str2num(dataArray{1,5}{i});
    end
end

%%%%%%%% Here we welcome Elgiz %%%%%%%
% The idea is to AND all the required indexes

gyro_id_only = strcmp(dataArray{1,3},'IMU_GYRO');
gyro_id = gyro_id_only & drone_select_id & flight_duration_id;
gyro(:,1) = dataArray{1, 4}(gyro_id);
gyro(:,2) = array_col_5(gyro_id);
gyro(:,3) = dataArray{1, 6}(gyro_id);
t_gyro = dataArray{1, 1}(gyro_id);

accel_id_only = strcmp(dataArray{1,3},'IMU_ACCEL');
accel_id = accel_id_only & drone_select_id & flight_duration_id;
accel(:,1) = dataArray{1, 4}(accel_id);
accel(:,2) = array_col_5(accel_id);
accel(:,3) = dataArray{1, 6}(accel_id);
t_accel = dataArray{1, 1}(accel_id);

commands_id_only = strcmp(dataArray{1,3},'COMMANDS');
commands_id = drone_select_id & commands_id_only & flight_duration_id;
```

```
commands_index = find(commands_id == 1);
commands(:,1) = dataArray{1, 7}(commands_id);
commands(:,2) = dataArray{1, 8}(commands_id);
t_commands = dataArray{1, 1}(commands_id);

gps_id = drone_select_id & gps_id & flight_duration_id;
altitude(:,1) = dataArray{1, 8}(gps_id)/1000;
t_altitude = dataArray{1, 1}(gps_id);

% Labeling outputs (Fault, Normal)

% FAULT (Here different faults are all labeled under one category which is
    fault)
% Finding the faulty command indexes
% SETTINGS give the multiplication factor that is used to inject the fault
% to control surfaces.

% Each time a fault is injected from the ground station, there
% appears a SETTINGS message, with the information on the fault signal.
% An example : 535.8420 18 SETTINGS 1.000000 0.500000
%             [time droneNum typeMass leftContSurfaceEfficiency
%             rightContSurfaceEfficiency]
% The values 1.000000 and 0.500000 are manual inputs from GCS by operator.

settings_id = strcmp(dataArray{1,3},'SETTINGS');
settings_index = find(settings_id == 1);

% Indexes of setting command where drone set to nominal control surface
% condition (1.00 1.00 for multiplicative fault)
set_nominal = settings_index((dataArray{1,4}(settings_index)==1)&...
(array_col_5(settings_index)==1)&(dataArray{1,6}(settings_index)==0)&...
(dataArray{1,7}(settings_index)==0));
```

```
% number of fault sets
num_fault_set = length(settings_index) - length(set_nominal);

% Initialization fault_start_stop and nominal_start_stop vectors
fault_start_stop = zeros(2, num_fault_set);
% adding the intervals before any fault injected (after an certain altitude
% to first fault anf after the last fault finishes to a certain altitude)
nominal_start_stop = zeros(2, length(set_nominal) + 1);
j = 1;
k = 2;
for i = 1 : (length(settings_index) - 1)
    if ~any(set_nominal==settings_index(i))
        % fault_start_stop rows : starting_index end_index
        %                 coloum : each coloumn is for a different fault
        %                 injected
        fault_start_stop(1:2,j) = [settings_index(i) (settings_index(i + 1)
            - 1)]';
        j = j + 1;

    else
        nominal_start_stop(1:2,k) = [settings_index(i) (settings_index(i +
            1) - 1)]';
        k = k + 1;
    end
end
% Adding the nominal intervals starting from passing an altitude to the
% first falut injected (at start of the flight) and starting after the last
% fault finishes until it descents until the same altitude limit (at the
% end of the flight)
nominal_start_stop(1:2,1) = [first_altPass (fault_start_stop(1,1) - 1)]';
```

```
nominal_start_stop(1:2,length(set_nominal) + 1) =
    [fault_start_stop(2,num_fault_set)+1 last_altPass]';

%%%%%%%%% Hello Ewoud %%%%%%%%%%
% act_id = strcmp(dataArray{1,3},'ROTORCRAFT_CMD');
% u_in(:,1) = dataArray{1, 4}(act_id);
% u_in(:,2) = array_col_5(act_id);
% u_in(:,3) = dataArray{1, 6}(act_id);
% u_in(:,4) = dataArray{1, 7}(act_id);
% t_act = dataArray{1, 1}(act_id);
%
% gps_id = strcmp(dataArray{1,3},'GPS_INT');
% ecefv(:,1) = dataArray{1, 11}(gps_id)/100;
% ecefv(:,2) = dataArray{1, 12}(gps_id)/100;
% ecefv(:,3) = dataArray{1, 13}(gps_id)/100;
% t_gps = dataArray{1, 1}(gps_id);
```

B.2 selectDataToInvest.m

```
% Copyright 2017 Elgiz Baskaya

% This file is part of cureDDrone.

% cureDDrone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

% FAULT DETECTION VIA SVM
% This code assumes that you already have a data set of normal and faulty
% situation sensor outputs.

%% FAULT SELECTION
% to check available fault indexes (the index when they are set by
% operator) setdiff(settings_index,set_nominal)
% and their corresponding start_index end_index, check fault_start_stop

fault_id = zeros(length(dataArray{1,1}),1);
% Select which fault interval you would like to investigate
% fault_id(fault_start_stop(1,FAULT_NUM_YOU_WANTTO_SIMULATE):...
% fault_start_stop(2,FAULT_NUM_YOU_WANTTO_SIMULATE)) = 1;

% % One surface stuck at zero fault
% fault_id(fault_start_stop(1,23):fault_start_stop(2,23)) = 1;

% One surface loss if efficiency fault
fault_id(fault_start_stop(1,3):fault_start_stop(2,3)) = 1;

% All faulty phase indexes
% for i = 1 : length(fault_start_stop)
%     fault_id(fault_start_stop(1,i):fault_start_stop(2,i)) = 1;
% end

gyro_fault_cond_id = fault_id & gyro_id_only;
```

```
gyro_fault_cond(:,1) = dataArray{1, 4}(gyro_fault_cond_id);
gyro_fault_cond(:,2) = array_col_5(gyro_fault_cond_id);
gyro_fault_cond(:,3) = dataArray{1, 6}(gyro_fault_cond_id);
t_gyro_fault_cond = dataArray{1, 1}(gyro_fault_cond_id);

accel_fault_cond_id = fault_id & accel_id_only;

accel_fault_cond(:,1) = dataArray{1, 4}(accel_fault_cond_id);
accel_fault_cond(:,2) = array_col_5(accel_fault_cond_id);
accel_fault_cond(:,3) = dataArray{1, 6}(accel_fault_cond_id);
t_accel_fault_cond = dataArray{1, 1}(accel_fault_cond_id);

% Selection of nominal condition
% to check available fault indexes (the index when they are set by
% operator) : see variable set_nominal
% and their corresponding start_index end_index, check nominal_start_stop

nominal_id = zeros(length(dataArray{1,1}),1);
% Select which nominal phase interval you would like to investigate
% nominal_id(nominal_start_stop(1,NOMINAL_COND_NUM_YOU_WANTTO_SIMULATE):...
% nominal_start_stop(2,NOMINAL_COND_NUM_YOU_WANTTO_SIMULATE)) = 1;

% % One surface stuck at zero fault
% nominal_id(nominal_start_stop(1,5):nominal_start_stop(2,5)) = 1;

% One surface loss if efficiency fault
nominal_id(nominal_start_stop(1,1):nominal_start_stop(2,1)) = 1;

% All nominal phase indexes
% for i = 1 : length(nominal_start_stop)
%     nominal_id(nominal_start_stop(1,i):nominal_start_stop(2,i)) = 1;
% end
```

```
gyro_nominal_cond_id = nominal_id & gyro_id_only;

gyro_nominal_cond(:,1) = dataArray{1, 4}(gyro_nominal_cond_id);
gyro_nominal_cond(:,2) = array_col_5(gyro_nominal_cond_id);
gyro_nominal_cond(:,3) = dataArray{1, 6}(gyro_nominal_cond_id);
t_gyro_nominal_cond = dataArray{1, 1}(gyro_nominal_cond_id);

accel_nominal_cond_id = nominal_id & accel_id_only;

accel_nominal_cond(:,1) = dataArray{1, 4}(accel_nominal_cond_id);
accel_nominal_cond(:,2) = array_col_5(accel_nominal_cond_id);
accel_nominal_cond(:,3) = dataArray{1, 6}(accel_nominal_cond_id);
t_accel_nominal_cond = dataArray{1, 1}(accel_nominal_cond_id);

% Forming the feature and output vectors to apply classification.
% Here we form the matrix as
% feature_vector = [acc_x_nominal acc_y_nom acc_z_nom gyro_x_nom gyro_y_nom
    gyro_z_nom
%                 acc_x_fault  acc_y_fault acc_z_fault gyro_x_faul
    gyro_y_fault gyro_z_fault]
feature_vector = [accel_nominal_cond gyro_nominal_cond; accel_fault_cond
    gyro_fault_cond];
% Assuming the time steps for the gyro and the accelerometers are sync.
t_features = [t_accel_nominal_cond; t_accel_fault_cond];
% Assuming same number of gyro and accelerometer data
% Labelling data

% nominal_label = cell(length(gyro_nominal_cond),1);
% nominal_label(:) = {'nominal'};
% fault_label = cell(length(gyro_fault_cond),1);
% fault_label(:) = {'fault'};
```

```
% label = [nominal_label; fault_label];
% output_vector = label;

nominal_label = zeros(length(gyro_nominal_cond),1);
fault_label = ones(length(gyro_fault_cond),1);
output_vector = [nominal_label; fault_label];

%% ADD FEATURES OF CONSEQUENT MEASUREMENTS

% % Number of next (and previous) measurements to add to the feature vector
    : N
% feature_vector_original = feature_vector;
% clear feature_vector;
% N = 3;
% [row,col] = size(feature_vector_original);
%
% addedFeat = zeros(row, N + 1);

% for i = 1 : col
%     % If features added before the current time measurement
%     addedFeat = addFeaturesBefore(feature_vector_original(:,i),N);
%     feature_vector(:,((i-1)*(N+1)+1):((i-1)*(N+1)+1+N)) = addedFeat;
%
%     % If features added both before and after the current time measurement
%     addedFeat = addFeaturesBeforeAfter(feature_vector_original(:,i),N);
%     feature_vector(:,((i-1)*(2*N+1)+1):((i-1)*(2*N+1)+1+2*N)) = addedFeat;
% end

% Figures to visualize data
% feature = [accel_nominal_cond;accel_fault_cond];
gscatter(feature_vector(:,1),feature_vector(:,3),output_vector,'gr')
legend('normal','fault')
```

```
set(legend,'FontSize',11);
xlabel({'$a_x$'},...
'FontUnits','points',...
'interpreter','latex',...
'FontSize',15,...
'FontName','Times')
ylabel({'$a_y$'},...
'FontUnits','points',...
'interpreter','latex',...
'FontSize',15,...
'FontName','Times')
print -depsc2 feat1vsfeat3.eps
```

B.3 arrangeDataSet.m

```
% Copyright 2017 Elgiz Baskaya

% This file is part of cureDDrone.

% cureDDrone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.
```

```
% FAULT DETECTION VIA SVM
% This code assumes that you already have a data set of normal and faulty
% situation sensor outputs.

%% Arrange training/test sets

feature_vec = feature_vector;
output_vec = output_vector;

% training set (around %80 percent of whole data set)
trainingDataExNum = ceil(80 / 100 * (length(feature_vec)));

% Select %80 of data for training and leave the rest for testing
randomSelectionColoumnNum = randperm(length(feature_vec),trainingDataExNum);

% Training set for feature and output
% feature_vec_training .:. feature matrix for training
% output_vec_training .:. output vector for training
feature_vec_training = feature_vec(randomSelectionColoumnNum, :);
output_vec_training = output_vec(randomSelectionColoumnNum, :);

% Test set for feature and output
feature_vec_test = feature_vec;
feature_vec_test(randomSelectionColoumnNum, :) = [];

output_vec_test = output_vec;
output_vec_test(randomSelectionColoumnNum, :) = [];

test_set_time = t_features;
test_set_time(randomSelectionColoumnNum) = [];
```

```
% To have same partions for cross-validations
rng(1);
cFold = cvpartition(length(feature_vec_training),'KFold',10);
```

B.4 svmFD.m

```
% Copyright 2017 Elgiz Baskaya

% This file is part of cureDDrone.

% cureDDrone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

% FAULT DETECTION VIA SVM
% This code assumes that you already have a data set of normal and faulty
% situation sensor outputs.

%% TRAINING PHASE
tic
% SVMModel is a trained ClassificationSVM classifier.
```

```
SVMModel = fitcsvm(feature_vec_training,output_vec_training,
    'KernelFunction','rbf','Standardize',true,'ClassNames',{'0','1'});
toc

% Support vectors
sv = SVMModel.SupportVectors;

%% CROSS VALIDATION
% 10-fold cross validation on the training data
% inputs : trained SVM classifier (which also stores the training data)
% outputs : cross-validated (partitioned) SVM classifier from a trained SVM
% classifier

% CVSVMModel is a ClassificationPartitionedModel cross-validated classifier.
% ClassificationPartitionedModel is a set of classification models trained
% on cross-validated folds.

CVSVMModel = crossval(SVMModel,'CVPartition',cFold);

% To assess predictive performance of SVMModel on cross-validated data
% "kfold" methods and properties of CVSVMModel, such as kfoldLoss is used

% Evaluate 10-fold cross-validation error.
% (Estimate the out-of-sample misclassification rate.)
crossValClassificErr = kfoldLoss(CVSVMModel);

% Predict response for observations not used for training
% Estimate cross-validation predicted labels and scores.
[elabelUntuned,escoreUntuned] = kfoldPredict(CVSVMModel);

max(escoreUntuned)
min(escoreUntuned)
```

```
% FIT POSTERIOR PROBABILITES fitPosterior(SVMModel) /
    fitSVMPosterior(CVSVMModel)
% [ScoreCVSVMModel,ScoreParameters] = fitSVMPosterior(CVSVMModel);
% Predict does not work here?


%% PREDICTION PHASE

[labelUntuned,scoreUntuned] = predict(SVMModel,feature_vec_test);

% %% FIT POSTERIOR PROBABILITES fitPosterior(SVMModel) /
    fitSVMPosterior(CVSVMModel)
% % "The transformation function computes the posterior probability
% % that an observation is classified into the positive class
    (SVMModel.Classnames(2)).
% % The software fits the appropriate score-to-posterior-probability
% % transformation function using the SVM classifier SVMModel, and
% % by conducting 10-fold cross validation using the stored predictor data
    (SVMModel.X)
% % and the class labels (SVMModel.Y) as outlined in REF : Platt, J.
% % "Probabilistic outputs for support vector machines and comparisons
% % to regularized likelihood methods". In: Advances in Large Margin
    Classifiers.
% % Cambridge, MA: The MIT Press, 2000, pp. 61-74"
% ScoreSVMModel = fitPosterior(SVMModel);
% [~,postProbability] = predict(ScoreSVMModel,feature_vec_test);

%% EVALUATING THE PERFORMANCE OF CLASSIFICATION WITH NEW DATA

% Evaluating the prediction performance of classification via
```

```
% CompactClassificationSVM class methods (e.g compareHoldout, edge, loss,
    margin,
% predict)

eUntuned = edge(SVMModel, feature_vec_test, output_vec_test);
mUntuned = margin(SVMModel, feature_vec_test, output_vec_test);

% Evaluating the prediction performance of classification via confusion
    matrix

[f1scoreUntuned, precisionUntuned, recallUntuned] =
    calcF1score(output_vec_test, str2double(labelUntuned));
%% Plot results
figure
gscatter(feature_vec_training(:,1),feature_vec_training(:,2),output_vec_training)
hold on
plot(sv(:,1),sv(:,2),'ko','MarkerSize',10)
legend('normal','fault','Support Vector')
legend('normal','fault')
hold off
set(legend,'FontSize',11);
xlabel({'$a_x$'},...
'FontUnits','points',...
'interpreter','latex',...
'FontSize',15,...
'FontName','Times')
ylabel({'$a_y$'},...
'FontUnits','points',...
'interpreter','latex',...
'FontSize',15,...
'FontName','Times')
print -depsc2 feat1vsfeat2.eps
```

B.5 svmFDtuningViaHeuristic.m

```
% Copyright 2017 Elgiz Baskaya

% This file is part of cureDDrone.

% cureDDrone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

% FAULT DETECTION VIA SVM
% This code assumes that you already have a data set of normal and faulty
% situation sensor outputs.

%% TUNING THE SVM CLASSIFIER using heuristic approach to select kernel scale

tic
% SVMModelTune is a trained ClassificationSVM classifier
% By passing 'KernelScale','auto' the software utilizes a heuristic
% approach to select kernel scale
```

```
SVMModelTune1 = fitcsvm(feature_vec_training,output_vec_training,
    'KernelFunction','rbf',
    'KernelScale','auto','Standardize',true,'ClassNames',{'0','1'});

%% CROSS VALIDATION
% 10-fold cross validation on the training data
% inputs : trained SVM classifier (which also stores the training data)
% outputs : cross-validated (partitioned) SVM classifier from a trained SVM
% classifier

% CVSVMModelTune is a ClassificationPartitionedModel cross-validated
    classifier.
% ClassificationPartitionedModel is a set of classification models trained
% on cross-validated folds.
CVSVMModelTune1 = crossval(SVMModelTune1,'CVPartition',cFold);

% To assess predictive performance of SVMModelTune on cross-validated data
% "kfold" methods and properties of CVSVMModelTune, such as kfoldLoss is
    used

% Evaluate 10-fold cross-validation error.
% (Estimate the out-of-sample misclassification rate.)
crossValClassificErrTuning1 = kfoldLoss(CVSVMModelTune1);

% Predict response for observations not used for training
% Estimate cross-validation predicted labels and scores.
[elabelTune1,escoreTune1] = kfoldPredict(CVSVMModelTune1);

max(escoreTune1)
min(escoreTune1)

%% RETRAIN SVM CLASSIFIER
```

```
% Retrain for different values of BoxConstraint and KernelScale
% This KernelScale is the KernelScale found by the heuristic approach
sampleSpace = 11;
kernelScaleFactor = zeros(1,sampleSpace + 1);
boxConstraint = zeros(1,sampleSpace + 1);
crossValClassificErrTuning2 = zeros(sampleSpace,sampleSpace);

ks = SVMModelTune1.KernelParameters.Scale;
boxConstraint(1) = 1e-5;
kernelScaleFactor(1) = 1e-5;
minCrossValClassificError = 100;

for i = 1 : sampleSpace
    for j = 1 : sampleSpace

        SVMModelTune2 = fitcsvm(feature_vec_training,output_vec_training,
            'KernelFunction','rbf', 'KernelScale',ks *
            kernelScaleFactor(j),'BoxConstraint',boxConstraint(i),...
        'Standardize',true,'ClassNames',{'0','1'});

        % CrossValidate
        CVSVMModelTune2 = crossval(SVMModelTune2,'CVPartition',cFold);
        crossValClassificErrTuning2(i,j) = kfoldLoss(CVSVMModelTune2);
        if crossValClassificErrTuning2(i,j) < minCrossValClassificError
            minCrossValClassificError = crossValClassificErrTuning2(i,j);
            kernelScaleOptim = SVMModelTune2.KernelParameters.Scale;
            boxConstraintOptim = SVMModelTune2.ModelParameters.BoxConstraint;
        end
        kernelScaleFactor(j + 1) = kernelScaleFactor(j) * 10;
    end
    boxConstraint(i + 1) = boxConstraint(i) * 10;
end
```

```
toc

%% TRAIN AGAIN WITH THE TUNED KernelScale and BoxConstraint
SVMModelTune = fitcsvm(feature_vec_training,output_vec_training,
    'KernelFunction','rbf',
    'KernelScale',kernelScaleOptim,'BoxConstraint',boxConstraintOptim,...
'Standardize',true,'ClassNames',{'0','1'});

%% PREDICTION PHASE

[labelTune,scoreTune] = predict(SVMModelTune,feature_vec_test);

%% EVALUATING THE PERFORMANCE OF CLASSIFICATION WITH NEW DATA

% Evaluating the prediction performance of classification via
% CompactClassificationSVM class methods (e.g compareHoldout, edge, loss,
    margin,
% predict)

eTune = edge(SVMModelTune, feature_vec_test, output_vec_test);
mTune = margin(SVMModelTune, feature_vec_test, output_vec_test);

% Evaluating the prediction performance of classification via confusion
    matrix

[f1scoreTune, precisionTune, recallTune] = calcF1score(output_vec_test,
    str2double(labelTune));
```

B.6 svmFDtuningViaOptim.m

```
% Copyright 2017 Elgiz Baskaya

% This file is part of cureDDrone.

% cureDDrone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

% FAULT DETECTION VIA SVM
% This code assumes that you already have a data set of normal and faulty
% situation sensor outputs.

%% TUNING THE SVM CLASSIFIER using Bayesian Optimization
sigma = optimizableVariable('sigma',[1e-5,1e5],'Transform','log');
box = optimizableVariable('box',[1e-5,1e5],'Transform','log');

minfn = @(z)kfoldLoss(fitcsvm(feature_vec_training,output_vec_training,...
    'CVPartition',cFold,'KernelFunction','rbf','BoxConstraint',z.box,...
    'KernelScale',z.sigma));

results = bayesopt(minfn,[sigma,box],'IsObjectiveDeterministic',true,...
    'AcquisitionFunctionName','expected-improvement-plus')
```

```
z(1) = results.XAtMinObjective.sigma;
z(2) = results.XAtMinObjective.box;
SVMModelTuned = fitcsvm(feature_vec_training,output_vec_training,...
'KernelFunction','rbf','KernelScale',z(1),'BoxConstraint',z(2));

%% CROSS VALIDATION
% 10-fold cross validation on the training data
% inputs : trained SVM classifier (which also stores the training data)
% outputs : cross-validated (partitioned) SVM classifier from a trained SVM
% classifier

% CVSVMModel is a ClassificationPartitionedModel cross-validated classifier.
% ClassificationPartitionedModel is a set of classification models trained
% on cross-validated folds.

CVSVMModelTuned = crossval(SVMModelTuned,'CVPartition',cFold);

% To assess predictive performance of SVMModel on cross-validated data
% "kfold" methods and properties of CVSVMModel, such as kfoldLoss is used

% Evaluate 10-fold cross-validation error.
% (Estimate the out-of-sample misclassification rate.)
crossValClassificErrTuned = kfoldLoss(CVSVMModelTuned);

%% PREDICTION PHASE
[labelTuned,scoreTuned] = predict(SVMModelTuned,feature_vec_test);

%% EVALUATING THE PERFORMANCE OF CLASSIFICATION WITH NEW DATA

% Evaluating the prediction performance of classification via
```

```
% CompactClassificationSVM class methods (e.g compareHoldout, edge, loss,
    margin,
% predict)

eTuned = edge(SVMModelTuned, feature_vec_test, output_vec_test);
mTuned = margin(SVMModelTuned, feature_vec_test, output_vec_test);

% Evaluating the prediction performance of classification via confusion
    matrix

[f1scoreTuned, precisionTuned, recallTuned] = calcF1score(output_vec_test,
    labelTuned);
```

B.7 calcF1score.m

```
function [f1Score,precision,recall] = calcF1score(labelActual,
    labelPredicted)

truePositive = sum(labelPredicted & labelActual);
falsePositive = sum(~((~labelPredicted)|labelActual));
falseNegative = sum((~labelPredicted) & labelActual);
% trueNegative = sum(~(labelPredicted|labelActual));

precision = truePositive / (truePositive + falsePositive);
recall = truePositive / (truePositive + falseNegative);
f1Score = 2 * precision * recall / (precision + recall);
end
```

B.8 addFeaturesBefore.m

```
% Copyright 2017 Elgiz Baskaya

% This file is part of cureDDrone.

% cureDDrone is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% curedRone is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with curedRone. If not, see <http://www.gnu.org/licenses/>.

% FAULT DETECTION VIA SVM
% This code assumes that you already have a data set of normal and faulty
% situation sensor outputs.

%% FEATURE ADDITION
% Add features (measurements) of up to N - 1 measurements before
% An example : Lets say N = 3
% Before adding the features the feature vector
% v = [v1
%      v2
%      v3
%      v4
%      v5
%      v6
```

```
%       .
%       .
%      v_m]    where m is the number of measurements

% For a given feature matrix above this file outputs
% REMINDER the number of measurements before and after you want to add to
% the feature vector is N - 1 both sides of the original feature vector.
% So this file will add N - 1 coloumns before and N - 1 coloumns after
% the original feature vector. And the indexes for the resulting matrix
    will be:
% m is the number of measurements and also the number of rows in the
% feature vector
% Coloumn marked with * is the original feature vector
%                                          *
% v_(2-N)    ...      v_(-1)    v_0      v_1
% v_(2-N+1)  ...        v_0      v_1      v_2
% v_(2-N+2)  ...        v_1      v_2      v_3
% v_(2-N+3)  ...        v_2      v_3      v_4
%     .      ...         .        .        .
%     .      ...         .        .        .
% v_(m-N)    ...     v_(m-2)  v_(m-1)   v_m

function [vNew] = addFeaturesBefore(v,N)

N = N + 1;
v_b = v;
vNew(:,N) = v;

for i = 1 : N - 1
    v_b(2:end,:) = v_b(1:end-1,:);
    vNew(:,N - i) = v_b;
end
```

Bibliography

[1] ADDSAFE FP7 Project. `http://addsafe.deimos-space.com/`, Accessed: 2016-07-09.

[2] National Aeronautics and Space Administration (NASA). *Safe2DitchTechnology*, 2018 (accessed August 22, 2018).

[3] Mark A Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[4] Amazon. *Revising the Airspace Model for the Safe Integration of Small Unmanned Aircraft Systems*, (accessed September 3, 2018).

[5] Plamen Angelov. *Sense and avoid in UAS: research and applications*. John Wiley & Sons, 2012.

[6] Daniel Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM journal on scientific and statistical computing*, 6(1):128–143, 1985.

[7] Thomas Bak. Spacecraft attitude determination. 1999.

[8] Elgiz Baskaya, Murat Bronz, and Daniel Delahaye. Flight simulation of a mako uav for use in data-driven fault diagnosis. In *IMAV 2017, 9th international microair vehicle conference*, 2017.

[9] Elgiz Baskaya, Guido Manfredi, Murat Bronz, and Daniel Delahaye. Flexible open architecture for uass integration into the airspace: Paparazzi autopilot system. In *IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. IEEE, 2016.

[10] Christine M Belcastro, John V Foster, Gautam H Shah, Irene M Gregory, David E Cox, Dennis A Crider, Loren Groff, Richard L Newman, and David H Klyde. Aircraft loss of control problem analysis and research toward a holistic solution. *Journal of Guidance, Control, and Dynamics*, 40(4):733–775, 2017.

[11] Christopher M Bishop. Pattern recognition and machine learning (information science and statistics) springer-verlag new york. *Inc. Secaucus, NJ, USA*, 2006.

[12] Mogens Blanke, Christian W Frei, Franta Kraus, Ron J Patton, and Marcel Staroswiecki. What is fault-tolerant control. In *Preprints of 4th IFAC Symposium on Fault Detection Supervision ans Safety for Technical Processes, SAFEPROCESS*, pages 40–51, 2000.

[13] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[14] Pascal Brisset, Antoine Drouin, Michel Gorraz, Pierre-Selim Huard, and Jeremy Tyler. The paparazzi solution. In *MAV 2006, 2nd US-European competition and workshop on micro air vehicles*, pages pp–xxxx, 2006.

[15] Murat Bronz, Hector Garcia de Marina, and Gautier Hattenberger. In-flight thrust measurement using on-board force sensor. In *AIAA Atmospheric Flight Mechanics Conference*, page 0698, 2017.

[16] Murat Bronz and Gautier Hattenberger. Aerodynamic characterization of an off-the-shelf aircraft via flight test and numerical simulation. In *AIAA Flight Testing Conference*, page 3979, 2016.

[17] Arthur E Bryson, Walter F Denham, and Stewart E Dreyfus. Optimal programming problems with inequality constraints. *AIAA journal*, 1(11):2544–2550, 1963.

[18] Sam Byford. *Deepmind Go Challenge*, (accessed September 11, 2018).

[19] E Chow and A Willsky. Analytical redundancy and the design of robust failure detection systems. *IEEE Transactions on Automatic control*, 29(7):603–614, 1984.

[20] European Commission. *Aviation: Commission is taking the European drone sector to new heights*, (accessed September 3, 2018).

[21] Jean-Philippe Condomines. *Développement d'un estimateur d'état non linéaire embarqué pour le pilotage-guidage robuste d'un micro-drone en milieu complexe*. PhD thesis, INSTITUT SUPERIEUR DE L'AERONAUTIQUE ET DE L'ESPACE (ISAE), 2015.

[22] Dianne Cook and Andreas Buja. Manual controls for high-dimensional data projections. *Journal of computational and Graphical Statistics*, 6(4):464–480, 1997.

[23] Dianne Cook, Andreas Buja, Javier Cabrera, and Catherine Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4(3):155–172, 1995.

[24] Dianne Cook and Deborah F Swayne. *Interactive and dynamic graphics for data analysis: with R and GGobi*. Springer Science & Business Media, 2007.

[25] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[26] DeepMind. *AlphaGo*, 2017 (accessed October 8, 2018).

[27] Deepmind. *Producing flexible behaviors in simulated environments*, 2017 (accessed October 8, 2018).

[28] M Delvaux. Report with recommendations to the commission on civil law rules on robotics (2015/2103 (inl)). Technical report, A8-0005/2017, 24 Jan 2017. European Parliament, Brussels Google Scholar, 2017.

[29] 12 drone disasters that show why the FAA hates drones. `http://www.techrepublic.com/article/12-drone-disasters-that-show-why-the-faa-hates-drones/`, March 20, 2015.

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

[31] European Aviation Safety Agency (EASA). *Policy Statement Airworthiness Certification of Unmanned Aircraft Systems (UAS), E.Y013-01*, 2009 (accessed August 30, 2018).

[32] European Aviation Safety Agency (EASA). *Technical Opinion, Introduction of a regulatory framework for the operation of unmanned aircraft*, 2015 (accessed August 29, 2018).

[33] European Aviation Safety Agency (EASA). *'Prototype commission' regulation on unmanned aircraft operations*, 2016 (accessed August 30, 2018).

[34] European Aviation Safety Agency (EASA). *Annual safety review, 2017*, 2017 (accessed August 30, 2018).

[35] European Aviation Safety Agency (EASA). *Loss of Control (LOC-I)*, 2018 (accessed August 20, 2018).

[36] Introduction of a regulatory framework for the operation of drones, 2015.

[37] Heinz Erzberger and Russel A Paielli. Concept for next generation air traffic control system. *Air Traffic Control Quarterly*, 10(4):355–378, 2002.

[38] Introduction of a regulatory framework for the operation of unmanned aircraft systems in the 'open' and 'specific' categories, 2018.

[39] Aislan Gomide Foina, Clemens Krainer, and Raja Sengupta. An unmanned aerial traffic management solution for cities using an air parcel model. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1295–1300. IEEE, 2015.

[40] Fpvhub. *What is First Person View (FPV) for Radio Controlled Flight?*, 2016 (accessed September 29, 2018).

[41] Ulrike Esther Franke. Civilian drones: Fixing an image problem? `http://isnblog.ethz.ch/security/civilian-drones-fixing-an-image-problem`, 2015.

[42] Philippe Goupil, Josep Boada-Bauxell, Andres Marcos, Paulo Rosa, Murray Kerr, and Laurent Dalbies. An overview of the fp7 reconfigure project: industrial, scientific and technological objectives. *IFAC-PapersOnLine*, 48(21):976–981, 2015.

[43] Wei-hua Gui and Xiao-ying Liu. Fault diagnosis technologies based on artificial intelligence for complex process. *Basic Automation*, 4:000, 2002.

[44] Steve R Gunn et al. Support vector machines for classification and regression. *ISIS technical report*, 14:85–86, 1998.

[45] Anna Hagenblad, Fredrik Gustafsson, and Inger Klein. A comparison of two methods for stochastic fault detection: the parity space approach and principal component analysis, 2004.

[46] Chingiz Hajiyev and Fikret Caliskan. 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.

[47] IBM. *Deep Blue*, (accessed September 11, 2018).

[48] International Civil Aviation Organization (ICAO). *Cir 328, Unmanned Aircraft Systems (UAS)*, 2011 (accessed August 29, 2018).

[49] International Civil Aviation Organization (ICAO). *Amendment No. 43 to the international standards, Rules of the air, Annex 2 to the convention on international civil aviation*, 2012 (accessed August 30, 2018).

[50] International Civil Aviation Organization (ICAO). *Amendment No. 6 to the international standards, Aircraft nationality and registration marks Annex 7 to the convention on international civil aviation*, 2012 (accessed August 30, 2018).

[51] International Civil Aviation Organization (ICAO). *Convention on International Civil Aviation*, 2015 (accessed August 29, 2018).

[52] International Civil Aviation Organization (ICAO). *Technical Opinion, Introduction of a regulatory framework for the operation of unmanned aircraft*, 2015 (accessed August 29, 2018).

[53] Tech Insider. *Google's DeepMind AI Just Taught Itself To Walk*, 2017 (accessed October 8, 2018).

[54] Rolf Isermann and Peter Ballé. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control engineering practice*, 5(5):709–719, 1997.

[55] JARUS guidelines on Specific Operations Risk Assessment(SORA), 2017.

[56] Parimal Kopardekar, Joseph Rios, Thomas Prevot, Marcus Johnson, Jaewoo Jung, and John E Robinson III. Unmanned aircraft system traffic management (utm) concept of operations. In *16th AIAA Aviation Technology, Integration, and Operations Conference*. AIAA Aviation, 2016.

[57] A Lambregts, Gregg Nesemeier, Richard Newman, and James Wilborn. Airplane upsets: Old problem, new issues. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 6867, 2008.

[58] Yann Le Cun. Learning process in an asymmetric threshold network. In *Disordered systems and biological organization*, pages 233–240. Springer, 1986.

[59] Yaniv Leviathan. *Google Duplex: An AI System for Accomplishing Real-World Tasks Over the Phone*, 2018 (accessed October 12, 2018).

[60] Minghu Li, Gang Li, and Maiying Zhong. A data driven fault detection and isolation scheme for uav flight control system. In *Control Conference (CCC), 2016 35th Chinese*, pages 6778–6783. TCCT, 2016.

[61] D Magill. Optimal adaptive estimation of sampled stochastic processes. *IEEE Transactions on Automatic Control*, 10(4):434–439, 1965.

[62] Sherri Magyarits. FAA,s Approach to Unmanned Aircraft Systems (UAS) Concept Maturation, Validation & Requirements Development, 2015.

[63] Guido Manfredi, Elgiz Baskaya, Jim Sharples, and Yannick Jestin. Unmanned aerial system operations for retail. In *ICAS 2018: 14th International Conference on Autonomic and Autonomous Systems, IASFR: Intelligent Autonomous Systems for the Future of Retail*, pages ISBN–978. IARIA, 2018.

[64] James W Melody, Thomas Hillbrand, Tamer Başar, and William R Perkins. H_∞ parameter identification for inflight detection of aircraft icing: The time-varying case. *Control Engineering Practice*, 9(12):1327–1335, 2001.

[65] Mikel M Miller. Modified multiple model adaptive estimation (m3ae) for simultaneous parameter and state estimation. Technical report, AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING, 1998.

[66] Marvin Minsky and Seymour Papert. Perceptron expanded edition, 1969.

[67] Markus Reinhard Möckli. *Guidance and control for aerobatic maneuvers of an unmanned airplaine*. PhD thesis, ETH Zurich, 2006.

[68] Andrew Ng. Lecture notes in machine learning, December 2017.

[69] Andrew Ng. Machine learning lectures. `https://www.coursera.org/learn/machine-learning`, 2017.

[70] Andrew Y Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

[71] Nokia. *Nokia showcases power of drones and LTE connectivity for public safety at D4G Award event in Dubai*, (accessed September 3, 2018).

[72] Stanford University School of Engineering. *Deep Reinforcement Learning*, 2017 (accessed October 8, 2018).

[73] Informal Meeting of EU Ministers Responsible for Territorial Cohesion and Urban Matters. *Declaration of Ministers towards the EU Urban Agenda*, 2015 (accessed August 29, 2018).

[74] Ministry of Infrastructure and Civil Aviation Authority Construction, European Aviation Safety Agency. *Drones as a leverage for jobs and new business opportunities*, 2016 (accessed September 29, 2018).

[75] Rohit Pandita, József Bokor, and Gary Balas. Closed-loop performance metrics for fault detection and isolation filter and controller interaction. *International Journal of Robust and Nonlinear Control*, 23(4):419–438, 2013.

[76] European Parliament and Council. *REGULATION (EC) No 216/2008 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*, 2008 (accessed August 29, 2018).

[77] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[78] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[79] Laurent Probst, Laurent Frideres, Bertrand Pedersen, and Christian Martinez-Diaz. Uav systems for civilian applications. `http://ec.europa.eu/DocsRoom/documents/13430`, 2015.

[80] RECONFIGURE FP7 Project. `reconfigure.deimos-space.com/`, Accessed: 2016-07-19.

[81] Unmanned Aerial Vehicle Reliability Study, 2003.

[82] Unmanned Systems Roadmap 2005 - 2030, 2005.

[83] Nathan D Richards, Neha Gandhi, Alec J Bateman, David H Klyde, and Amanda K Lampton. Vehicle upset detection and recovery for onboard guidance and control. *Journal of Guidance, Control, and Dynamics*, 40(4):920–933, 2016.

[84] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[85] Hector Rotstein, Ryan Ingvalson, Tamas Keviczky, and Gary J Balas. Fault-detection design for uninhabited aerial vehicles. *Journal of guidance, control, and dynamics*, 29(5):1051–1060, 2006.

[86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[87] P Russell and J Pardee. Joint safety analysis team-cast approved final report loss of control jsat results and analysis. *Commercial Aviation Safety Team, Washington, DC*, 2000.

[88] R Sharma and M Aldeen. Fault detection in nonlinear systems with unknown inputs using sliding mode observer. In *2007 American Control Conference*, pages 432–437. IEEE, 2007.

[89] Brian L Stevens, Frank L Lewis, and Eric N Johnson. *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons, 2015.

[90] John Stuelpnagel. On the parametrization of the three-dimensional rotation group. *SIAM review*, 6(4):422–430, 1964.

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

[92] Eric Thomas and Okko Bleeker. Options for insertion of rpas into the air traffic system. In *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*, pages 5B4–1. IEEE, 2015.

[93] SESAR Joint Undertaking. U-space blueprint. *web page, June*, 9, 2017.

[94] M. M. van Paassen, M. L. Bolton, and N. JimÃĺnez. Checking formal verification models for human-automation interaction. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3709–3714, Oct 2014.

[95] Vladimir Vapnik. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[96] Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

[97] Vladimir N Vapnik and Alexey J Chervonenkis. Theory of pattern recognition. 1974.

[98] Chris A Wargo, George Hunter, Kenneth Leiden, Jason Glaneuski, Brandon Van Acker, and Kevin Hatton. New entrants (rpa/space vehicles) operational impacts upon nas atm and atc. In *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*, pages 5B2–1. IEEE, 2015.

[99] James R. Wertz. *Spacecraft Attitude Determination and Control*. Kluwer Academic Publishers, 1978.

[100] Bong Wie. *Space vehicle dynamics and control*. American Institute of Aeronautics and Astronautics, 2008.

[101] Shen Yin, Xin Gao, Hamid Reza Karimi, and Xiangping Zhu. Study on support vector machine-based fault detection in tennessee eastman process. In *Abstract and Applied Analysis*, volume 2014. Hindawi Publishing Corporation, 2014.

[102] Youmin Zhang and Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual reviews in control*, 32(2):229–252, 2008.

[103] Lunlong Zhong. *Contribution to fault tolerant flight control under actuator failures*. PhD thesis, INSA de Toulouse, 2014.

Fault Detection & Diagnosis for Small UAVs via Machine Learning

Elgiz Baskaya
ENGIE Ineo - Groupe ADP - SAFRAN RPAS Chair
UAS Lab, ENAC
Toulouse, FRANCE
Email: elgiz.baskaya@enac.fr

Murat Bronz
UAS Lab, ENAC
Toulouse, FRANCE
Email: murat.bronz@enac.fr

Daniel Delahaye
Laboratoire MAIAA, ENAC
Toulouse, FRANCE
Email: deniel.delahaye@enac.fr

***Abstract*—The new era of small UAVs necessitates intelligent approaches towards the issue of fault diagnosis to ensure a safe flight. A recent attempt to accommodate quite a number of UAVs in the airspace requires to assure a safety level. The hardware limitations for these small vehicles point the utilization of analytical redundancy rather than the usual practice of hardware redundancy in the conventional flights. In the course of this study, fault detection and diagnosis for aircraft is reviewed. An approach of implementing machine learning practices to diagnose faults on a small fixed-wing is selected. The selection criteria behind is that, data- driven fault diagnosis enables avoiding the burden of accurate modeling needed in model-based fault diagnosis.**

In this study, first, a model of an aircraft is simulated. This model is not used for the design of Fault Detection and Diagnosis (FDD) algorithms, but instead utilized to generate data and test the designed algorithms. The measurements are simulated using the statistics of the hardware in the house. Simulated data is opted instead of flight data to isolate the probable effects of the controller on the diagnosis, which will complicate this preliminary study on FDD for drones.

A supervised classification method, SVM (Support Vector Machines) is used to classify the faulty and nominal flight conditions. The features selected are the gyro and accelerometer measurements. The fault considered is loss of effectiveness in the control surfaces of the drone. Principle component analysis is used to investigate the data by reducing the feature space dimension. The training is held offline due to the need of labeled data. The results show that for simulated measurements, SVM gives very accurate results on the classification of loss of effectiveness fault on the control surfaces.

I. Introduction

The cost effectiveness and reachability of COTS elements, shrinking size of electronics serve as a perfect environment for small flying vehicles to emerge. This accelerating trend towards small but capable flying vehicles is pushing the limits of both hardware and software potentials of industry and academia. Increasing usage of these vehicles for a variety of missions pushes a further liability to secure the flight.

To achieve a safe flight is not an easy task considering the unknowns of the systems hardware, environment and possible system faults and failures to emerge. Also, increasing demand on cost effective systems, resulting in the smaller sensors and actuators with less accuracy, impose the software to achieve even more. The expectation that UAVs should be less expensive than their manned counterparts might have a hit on reliability of the system. Cost saving measures other than the need to support a pilot/crew on board or decrement in size would probably lead to decrease in system reliability.

Systems are often susceptible to faults of different nature. Existing irregularities in sensors, actuators, or controller could be amplified due to the control system design and lead to failures. A fault could be hidden thanks to the control action (1).

The widely used method to increase reliability is to use more reliable components and/or hardware redundancy. Both requires an increase in the cost of the UAS conflicting one of the main reasons of UAS design itself band consumer expectations (2). To offer solutions for all different foreseen categories of airspace, a variety of approaches should be considered. While hardware redundancy could cope with the failure situations of UAVs in the certified airspace, it may not be suitable for UAVs in open or some subsets of specific categories due to budget constraints. Analytical redundancy is another solution, may be not as effective and simple as hardware redundancy, but relies on the design of intelligent methods to utilize every bit of information on board aircraft wisely to deal with the instances.

There are three approaches to achieve safe FTC in standard flight conventions. First one is the fail operational systems which are made insensitive to any single point component failure. The second approach is the fail safe systems where a controlled shut down to a safe state is practiced whenever a critical fault is pointed out by a sensor. The level of degradation assures to switch to robust (alternate) or direct (minimal level of stability augmentation independent of the nature of the fault) mode. Switching from nominal mode to the robust and direct modes leads to a decrease in the available GNC functions. This causes a degradation in ease of piloting. And also some optimality conditions could have

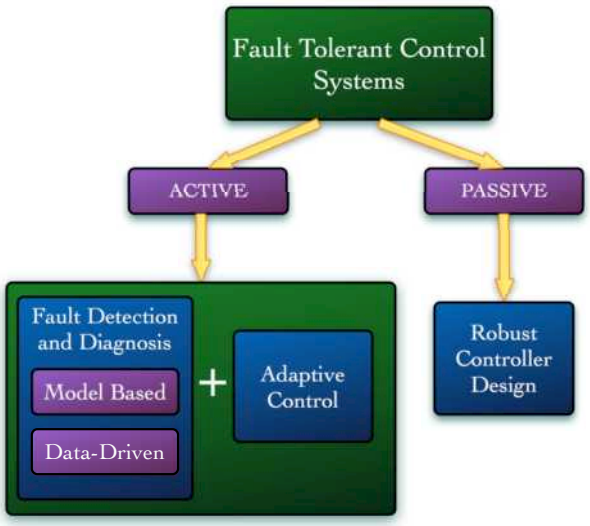

Fig. 1. Variations of fault tolerant control systems

been compromised. The third approach is fault tolerant control systems in which redundancy in the plant and the automation system is employed to design software that monitors the components and takes in action whenever needed. The strategy is most probably to try to keep plant availability and accept reduced performance (3).

II. Methods for FTCS

A common categorization of FTCS is passive and active FTCS. In passive FTCS, the flight controller is designed in such a way to accommodate not only the disturbances but also 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, this method most probably fails.

Even with a long list of available methods, aerospace industry has not implemented FTC widely, except some space systems, due to the evolving nature of the methods, the tricks coming with the nonlinear nature of the problem, design complexity and high possibility of wrong alarms in case of large disturbances and/or modeling uncertainties. So the already carried reliability measures concerning the hardware redundancy is now the preferred way because of its ease and maturity being implemented on various critical missions with considering human lives.

III. Fault Detection and Diagnosis

FDD is handled in two main steps; fault detection and fault diagnosis. Fault diagnosis encapsulates fault isolation and fault identification. The methods for detection and diagnosis are investigated for their frequency of utilization separately for sensor, actuator, process and controller faults in (4). FDD should not only be sensitive to the faults but also robust to the model uncertainties and external disturbances.

Two distinct options to proceed in analytical redundancy are the model based approaches and data-driven approaches. They form the two ends of a continuous solution set line, so utilizing them in a combination might end up with better solutions. 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.

A. Model Based

In model based approaches, relations between measurements and estimated states are exploited to detect possible dysfunction. The most common ways to implement a model based approach is to estimate the states, estimate the model parameters, or parity-space. The accuracy of the results depend on the type of faults (additive or multiplicative). Additive faults affects the variables of the process by a summation whereas the multiplicative faults by a multiplication. When only output signal can be measured, signal model based methods can be employed for fault detection such as Bandpass filters, Spectral analysis(FFT) and maximum entropy estimation. For the case, both the input and output signals are available, the utilized methods for fault detection are called the process based methods: State and output observers(estimators), Parity equations and Identification and parameter estimation. They generate residuals for state variables or output variables. When previous works investigated, it is concluded that the most widely used technique for sensor and actuator faults is the state and output observers (estimators) and for process faults, identification and parameter estimation (4).

The output of the model based fault detection methods is the stochastic behaviour with mean values and variances. With the use of change detection methods, deviations from the normal behavior can be detected. For that purpose, three available methods considered are, mean and variance estimation, likelihood-ratio-test and Bayes decision, run-sum test and two-probe t-test. Fault detection is only supported by simple threshold logic or hypothesis testing in most of the applications (4).

A bunch of studies discovers the band of different approaches for model-based fault detection. Detecting sensor and actuator faults via state estimation, utilizing an EKF is applied to a F-16 model in (5). Parameter identification via H_∞ filter is used to indicate icing in (6).

A drawback of model-based approaches is that they require accurate model of the aircraft for successful detection. In a small UAV system susceptible to various uncertainties/disturbances and most of the cases does not have an accurate model, leading a model-based approach might fail. And also, a mathematical model of a UAV is constructed within the flight envelope, and does not necessarily describe the possible dynamics invoked by a failure on board.

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 (7). 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 (8). (9) 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.

B. Machine Learning

Model-based approaches had various successful applications until now, most of them assuming accurate model is available on board. With the new era of UAVs, the airspace is expected to be populated by an abrupt increase in the number of UAVs. The variety of UAVs, expense of accurate modeling practices, the difficulty in modeling the behavior of UAV in case of failures, 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, 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 to label 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 (10).

Amidst data driven methods for FDD, such as Neural Networks (11) and Principal Component Qnalysis (PCA) (12), Support Vector Machines (SVM) appear more recently in the literature. (13) argues artificial intelligence methods for fault detection of complex systems. Comparison between PCA and model based stochastic parity space approaches is given in (14). In (15), 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 (16). 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 (17; 18). 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 (19). 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.

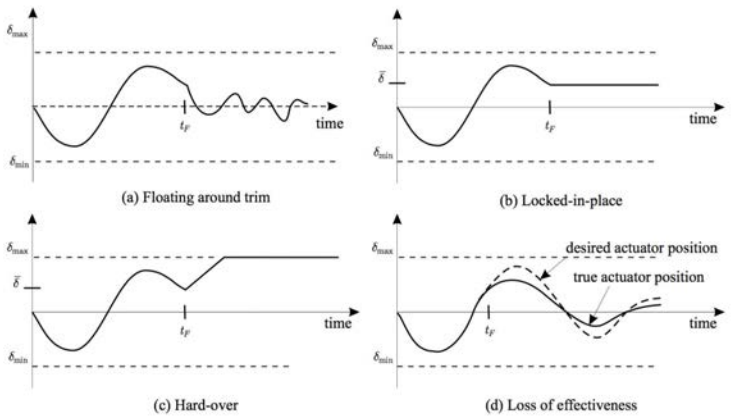

Fig. 2. Common actuator faults (1)

IV. System Modeling

In this study, first, a model of an aircraft is simulated. This model, will not be used for the design of FDI algorithms, but instead will be utilized to test them. Nonlinear aircraft flight dynamics for translational and attitude motion can be given as a system of first order partial differential equations

$$\dot{\boldsymbol{x}}_{NED} = \boldsymbol{C}_b^n \boldsymbol{v}^b \tag{1}$$

$$\dot{\boldsymbol{v}}^b = \frac{1}{m}\left[m\boldsymbol{g}^b + \boldsymbol{F}_t^b + \boldsymbol{F}_a^b\right] - \boldsymbol{\omega}_{b/i}^{b\,\times}\boldsymbol{v}^b \tag{2}$$

$$\dot{q}_0 = -\frac{1}{2}\boldsymbol{q}_\nu^T \boldsymbol{\omega}_{b/i}^b \tag{3}$$

$$\dot{\boldsymbol{q}}_\nu = \frac{1}{2}\Big(\boldsymbol{q}_\nu^\times + q_0 \boldsymbol{I}_3\Big)\boldsymbol{\omega}_{b/i}^b \tag{4}$$

$$\boldsymbol{J}\dot{\boldsymbol{\omega}}_{b/i}^b = \boldsymbol{M} - \boldsymbol{\omega}_{b/i}^{b\,\times}\boldsymbol{J}\boldsymbol{\omega}_{b/i}^b \tag{5}$$

where $\boldsymbol{x}_{NED} \in \mathbb{R}^3$ is the position of the center of mass of UAV with respect to inertial frame $\mathcal{I}$ expressed in the body frame $\mathcal{B}$, $\boldsymbol{v}^b$ is the velocity of the center of mass of UAV with to $\mathcal{I}$ expressed in $\mathcal{B}$, $\boldsymbol{q} = [q_0, \boldsymbol{q}_v^T]^T \in \mathbb{R}^3 \times \mathbb{R}$ is the unit quaternion representing the attitude of the body frame $\mathcal{B}$ with respect to inertial frame $\mathcal{I}$ expressed in the body frame $\mathcal{B}$, $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is the positive definite inertia matrix of the drone, $\boldsymbol{M} \in \mathbb{R}^3$ represents the moments acting on the drone. The notation $\boldsymbol{x}^\times$ for a vector $x = [x_1 \quad x_2 \quad x_3]^{\mathrm{T}}$ represents the skew-symmetric matrix

$$\boldsymbol{x}^\times = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{6}$$

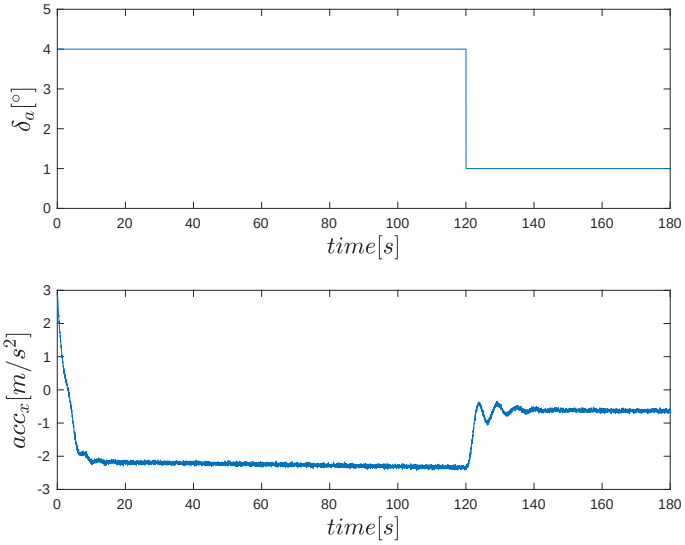

Fig. 3. Loss of effectiveness fault simulation in aileron command and corresponding accelerometer x axis measurement

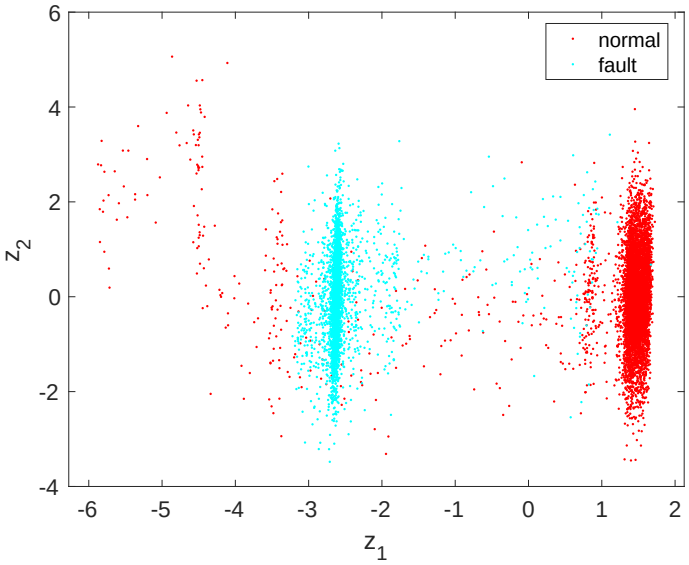

Fig. 4. Principal component analysis for visualization of faulty and normal data in reduced dimensional feature space

The stability and aerodynamic force coefficients are generated by AVL. The input vector can be written as $\boldsymbol{u}(t) \in \mathbb{R}^3$

$$\boldsymbol{u}(t) = \left[\delta_a \ \delta_e \ n\right]^{\mathrm{T}} \tag{7}$$

Here δ_a aileron deflection angle in degrees, δ_e elevator deflection angle in degrees, n engine speed in rev/s.

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

$$\boldsymbol{u}(t) = \boldsymbol{E}\boldsymbol{u}_c + u_f \tag{8}$$

where $\boldsymbol{u}_c$ is the desired control signal, $E = diag(e_1, e_2, e_3)$ is the effectiveness of the actuators where $0 \leq e_i \leq 1$ with $(i = 1, 2, 3)$ and u_f additive actuator fault. This model makes it possible to simulate all four types of actuator faults shown in Fig. 2. Most of the FDI algorithms are implemented to open-loop systems, ignoring the probable influences of the controller might cause on the detection performance (20). Here the system is open-loop as well. Further implementation of a controller is foreseen to understand the effect of the selected controllers. So we follow a step by step approach and hope to end with a more realistic case, in which real flight data is utilized and diagnosis is achieved online aside a functioning controller.

Since it is not possible to see all features, we take advantage of the dimensionality reduction technique called Principle Component Analysis (PCA) for visualization. Here what we do is to map the feature vector, $\boldsymbol{x} \in \mathbb{R}^n$ to a lower dimensional space where the new feature set will be represented by $\boldsymbol{z} \in \mathbb{R}^k$. Fig. 4 shows the resulted most significant elements for a mapped feature space from six dimensional feature vector to two.

V. Classification of Fault via SVM

SVM is a relatively new approach for classification offering better generalization property thanks to its foundations on the structural risk minimization principle (21; 22) while other classifiers usually only minimizes the empirical risk. This advances the capacity of generalization even with a small number of instances by reducing the risk of overfitting for a nicely tuned parameters setting. It can be applied to nonlinear systems and problems offering a vast number of features. Furthermore, taking advantage of convex optimization problems in the solution of SVM models, another attractive reason to use SVM rises as avoidance of global minimas, while Neural Networks is inherently prone to local minimas.

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. 5. 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.

SVM has other tricks to deal with not linearly seperable problems such as using kernels to map data into higher dimensional feature spaces where they can be separated with a linear hyperplane.

A binary classifier is used in this work to classify two classes, faulty and nominal. The fault considered in this study is the loss of effectiveness of the control surfaces. SVM being a supervised classification algorithm has two main phases as shown in Fig. 6. In the training phase, the model is learned as a fit to the labeled data that is fed to the SVM algorithm. This

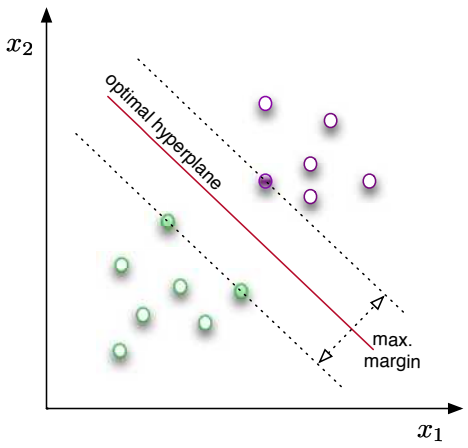

Fig. 5. SVM working principle

phase is usually followed with a tuning phase where some of the parameters of SVM is 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 is comprised of labeled data where the label can belong to one of two possible cases. This data set is saved in $\boldsymbol{X} \in \mathbb{R}^{\mathrm{m}\times\mathrm{n}}$ 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

$$min_{\gamma,\omega,b} \quad \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i \tag{9}$$

$$s.t. \quad y^i(\omega^T x(i) + b) \geq 1 - \xi_i, \ i = 1, \cdots, m \tag{10}$$

$$\xi_i \geq 0, \ i = 1, \cdots, m \tag{11}$$

$$\tag{12}$$

To avoiding overfitting, which is the main problem of parametric discrimination approaches such as neural networks, parameter C is tuned to result in the optimal fit for the cross validation set. The data set available is first divided to two portions with a percentage of %20, %80 where the bigger chunk 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. Overfitting means that the models trained being very accurate fit for the data they are trained to but fail to generalize with new inputs resulting in bad prediction performance for the new data. To assess the performance of the classifier trained with the training data is tuned to give a better performance with the cross validation data. And then the final ability of

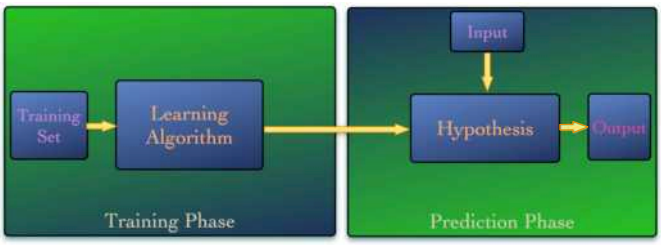

Fig. 6. Supervised learning basics

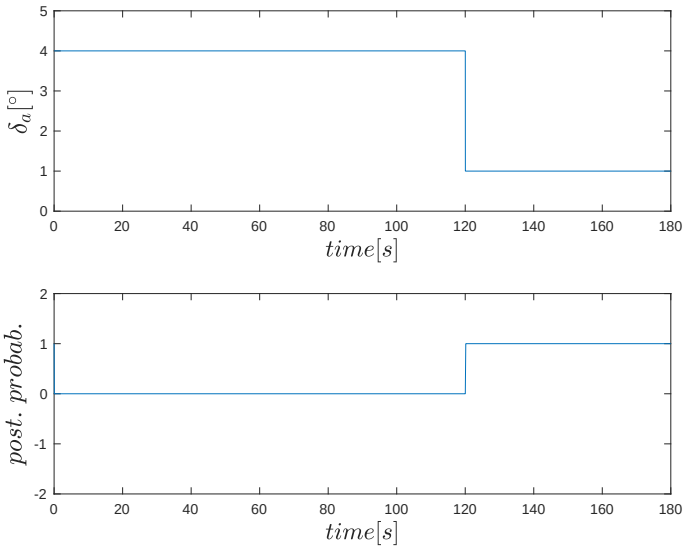

Fig. 7. Posterior probability of loss in effectiveness fault for test set when a fault is injected at t = 120s.

the classifier is tested on the test set. This parameter also tuned for the outliers to generalize the distribution of the data rather than resulting in fine fits for each individual data in the training set. 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. The output of the SVM classification is not the probability that the new measurement belongs to one class as is in the traditional classification problems, but directly the class information it belongs to. For investigating the performance of the classifier on the test set, a method (23) is used to calculate the posterior probabilities giving the probability that the new measurements belongs to faulty mode. Results shows as in Fig. 7 that prober tuning achieves very accurate and instant detection for the drone fault.

VI. Conclusion

Integration of drones into airspace needs the introduction of indigenous designs that will serve safe solutions for drones. One of the aspects of the problem is to assure a safe flight by designing fault detection and diagnosis with cheaper avionics common in a vast number of drones projected. This work aims to design a classifier via SVM to solve FDD of drones with

actuator faults. This problem possess various challenges. This work focuses on a loss of effectiveness fault which is more difficult than a stuck fault to diagnose, but easier to mitigate.

A model of a MAKO UAV is simulated to generate data and test the designed algorithms. The simulated data of gyro and accelerometer measurements are given to classifier to train for the two class labeled data set. A supervised classification method, SVM (Support Vector Machines) is used to classify the faulty and nominal flight conditions. Principle component analysis is used to investigate the data by reducing the feature space dimension. The training is held offline due to the need of labeled data but prediction is envisioned be held real time. The results show that for simulated measurements, SVM gives very accurate results on the classification of loss of effectiveness fault on the control surfaces.

Further study is envisaged to deal with the controller diagnosis interaction and classification of multiple faults. Also discussion of SVM for online training might be addressed since SVM is in need for labeled data which requires generating the labeled data during flight.

ACKNOWLEDGMENT

This work was supported by the ENGIE Ineo - Groupe ADP - SAFRAN RPAS Chair.

REFERENCES

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

[2] P. Angelov, *Sense and avoid in UAS: research and applications*. John Wiley & Sons, 2012.

[3] M. Blanke, C. W. Frei, F. Kraus, R. J. Patton, and M. Staroswiecki, "What is fault-tolerant control," in *Preprints of 4th IFAC Symposium on Fault Detection Supervision ans Safety for Technical Processes, SAFEPROCESS*, 2000, pp. 40–51.

[4] R. Isermann and P. Ballé, "Trends in the application of model-based fault detection and diagnosis of technical processes," *Control engineering practice*, vol. 5, no. 5, pp. 709–719, 1997.

[5] 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.

[6] 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.

[7] E. Chow and A. Willsky, "Analytical redundancy and the design of robust failure detection systems," *IEEE Transactions on Automatic control*, vol. 29, no. 7, pp. 603–614, 1984.

[8] H. Rotstein, R. Ingvalson, T. Keviczky, and G. J. Balas, "Fault-detection design for uninhabited aerial vehicles," *Journal of guidance, control, and dynamics*, vol. 29, no. 5, pp. 1051–1060, 2006.

[9] R. Sharma and M. Aldeen, "Fault detection in nonlinear systems with unknown inputs using sliding mode observer," in *2007 American Control Conference*. IEEE, 2007, pp. 432–437.

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

[11] M. Schlechtingen and I. F. Santos, "Comparative analysis of neural network and regression based condition monitoring approaches for wind turbine fault detection," *Mechanical systems and signal processing*, vol. 25, no. 5, pp. 1849–1875, 2011.

[12] X. Sun, H. J. Marquez, T. Chen, and M. Riaz, "An improved pca method with application to boiler leak detection," *ISA transactions*, vol. 44, no. 3, pp. 379–397, 2005.

[13] W.-h. Gui and X.-y. Liu, "Fault diagnosis technologies based on artificial intelligence for complex process," *Basic Automation*, vol. 4, p. 000, 2002.

[14] A. Hagenblad, F. Gustafsson, and I. Klein, "A comparison of two methods for stochastic fault detection: the parity space approach and principal component analysis," 2004.

[15] M. Li, G. Li, and M. Zhong, "A data driven fault detection and isolation scheme for uav flight control system," in *Control Conference (CCC), 2016 35th Chinese*. TCCT, 2016, pp. 6778–6783.

[16] V. Vapnik and A. Chervonenkis, "A note on one class of perceptrons," *Automation and remote control*, vol. 25, no. 1, p. 103, 1964.

[17] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[18] V. Vapnik, "The nature of statistical learning theory springer new york google scholar," 1995.

[19] N. Laouti, N. Sheibat-Othman, and S. Othman, "Support vector machines for fault detection in wind turbines," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 7067–7072, 2011.

[20] R. Pandita, J. Bokor, and G. Balas, "Closed-loop performance metrics for fault detection and isolation filter and controller interaction," *International Journal of Robust and Nonlinear Control*, vol. 23, no. 4, pp. 419–438, 2013.

[21] S. R. Gunn *et al.*, "Support vector machines for classification and regression," *ISIS technical report*, vol. 14, pp. 85–86, 1998.

[22] S. Yin, X. Gao, H. R. Karimi, and X. Zhu, "Study on support vector machine-based fault detection in tennessee eastman process," in *Abstract and Applied Analysis*, vol. 2014. Hindawi Publishing Corporation, 2014.

[23] J. Platt *et al.*, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

Flexible Open Architecture for UASs Integration into the Airspace: Paparazzi Autopilot System

Elgiz Baskaya*, Guido Manfredi*, Murat Bronz†‡ and Daniel Delahaye‡
*ENGIE Ineo - Safran RPAS Chair, ENAC, Toulouse, France
Email: elgiz.baskaya@enac.fr
†UAS Lab, ENAC, Toulouse, France
‡Laboratoire MAIAA, ENAC, Toulouse, France

***Abstract*—Air safety authorities are forced to develop regulations for UAS due to incidents disturbing public safety and demands from UAS operators. Despite numerous studies from the FAA and EASA, none of them decided on a regulation for UASs. The reliability of the flight is considered to be one of the main obstacles for UAVs integration. This is not an easy topic considering the unknowns of the systems, environment and possible failures. We believe the flexibility required for such solutions calls for open architectures. More specifically, this paper shows how the use of the *Paparazzi* open source auto-pilot system can ease the integration of low altitude UAS. To ensure safety, this integration needs to be achieved through airspace management and UAS reliability.**

Preliminary airspace designs, e.g. Amazon's, identify different zones depending on the UAS capabilities, population density and altitude. Plus, national rules evolution push to cope with a variety of requirements. Open source and modular architectures are key to adapt to these requirements. From a UTM point of view, *Paparazzi* provide features to ease congestion management, such as dynamic geofencing, trajectory communication and collision avoidance. Concerning reliability, current regulations focus on flight constraints but might be expected to involve regulations on software and hardware components as well. In such case, the increased cost will be inevitable for the demands of certification. In the *Paparazzi* software case, parts of the code have been formally proved and stable versions have thousands of flight hours. Such heritage might ease the certification process for smaller companies.

On top of its flexibility and reliability, *Paparazzi* offers a unique set of features, as an open source software, to achieve safe integration of low altitude UAS in the G airspace. To conclude this work, desirable new features and future work are discussed.

I. Introduction

The cost effectiveness and reachability of commercial off-the-shelf elements, and shrinking size of electronics serve as a perfect environment for small flying vehicles to emerge. Although inherited as military purposes in its infancy, nowadays Unmanned Aircraft System (UAS) are becoming efficient platforms for scientific/commercial domains. They offer benefits in terms of cost, flexibility, endurance as well as realizing missions that would be impossible with a human onboard. Increasing usage of these vehicles for a variety of missions, such as defense, civilian tasks including transportation, communication, agriculture, disaster mitigation applications pushes demand on the airspace. Furthermore, this congestion is predicted to accelerate with the growing diversity of these systems.

Commercial advantages, offered by these efficient systems, are already targeted by big companies worldwide, specifically in the US. The airspace regulatory authorities seem to be squeezed in between the companies, demanding a fast as possible access to airspace, and the concerns of the public about potential privacy breaches, safety and liability issues [1], [2]. Even with today's strictly regulated airspace, reported occurrences show that there are hurdles to solve before a further integration of UAS to airspace.

Advent of the new era of UAS seems to be hold by an unseen barrier of lack of regulatory framework for now. Different institutions all over the world, specifically National Aeronautics and Space Administration (NASA) and Federal Aviation Administration (FAA) in US, easa in Europe and international bases such as International Civil Aviation Organization (ICAO) are addressing safe integration of UAS in airspace. Although the approaches of regulatory bodies may vary, the aim remains the same: safe integration as soon as possible.

The tackles of safety during integration of UAS to airspace refer to different technical and organizational aspects including but not limited to control of traffic in segregated and non-segregated airspace, reliable communication, robust control of the Unmanned Air Vehicle (UAV), trajectory planning, detect&avoid.

In this study, we present the *Paparazzi* open source autopilot system, and its features, as a tool to ease the safe integration of low altitude UAS into National Air Space (NAS). In our argument, *Paparazzi* is favored thanks to its modular design, its support for congestion management and evolving reliability.

II. Open-source Autopilots for UAS

With the new era of First Person View (FPV) flights, especially for multi-rotor UAVs, there has been an exponential increment on the hardware and software of the open-source autopilots. A brief comparison of the popular current open-source and commercial autopilots is available in Table I. Usually the trend is to make the hardware as cheap as possible for recreational consumers. This reality is damaging the reputation of open-source autopilots as being not reliable

or robust, just because they are being used without attention. There exists a difference on the quality of sensors used on the open-source autopilot systems compared to commercial autopilots for sure, and this is extremely represented on the price of the units. This makes a difference on the flight quality of the vehicles, however with the new on-board processing power, more complex estimation algorithms and filters are being used in order to overcome this problem.

III. The PaparazziUAV project

The *Paparazzi* Autopilot System aims to provide a software and hardware solution for low-cost mini and micro unmanned air vehicles. It is started as a personal project in 2003 by Pascal Brisset and Antoine Drouin, and afterwards supported by ENAC in 2005. Being one of the first (if not the first) open-source autopilot system in the world, *Paparazzi* has attracted attention and led the others to start new branches and systems. The software is originally packaged for Debian/Ubuntu but can be manually installed on any GNU/Linux operating system even including MacOS-X. However, it is not compatible with Windows which automatically eliminates 90 % of the possible user community and therefore it is not as popular as other existing autopilot systems on the user market.

A. System Architecture

The system architecture of *Paparazzi* consists of three segments: Ground, Airborne, and the communication link between (Figure 1). The ground software of *Paparazzi* is mainly written in Ocaml, with some additional parts in Python and C. Thanks to its middle-ware communication bridge called Ivy-Bus, external softwares can be directly connected with publish and subscribe method to the ground segment without needing to modify and code at all. Airborne software is completely written in C, however there is an on-going work which implements the possibility of adding some C++ code parts.

B. Distinguishing Features

One of the distinguishing property of the *Paparazzi* is to support multi-UAV flight (Figure 2). Several projects have been using *Paparazzi* Autopilot System because of this additional feature. The communication of each vehicle is being transmitted to the ground control station directly on the current version, however air to air communication between flying vehicles and internal relay of information to ground control station is being implemented for a new coming version.

Modules are the easiest and most flexible way of adding new code into *Paparazzi*. There are over 130 modules written for *Paparazzi* by several developers and researchers including subjects on meteorology, imagery, surveillance, advance navigation, formation flight and collision avoidance, etc...

Paparazzi has its own complete flight plan language, where the user can define any possible trajectory by 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.

Fig. 1. Paparazzi autopilot system overview.

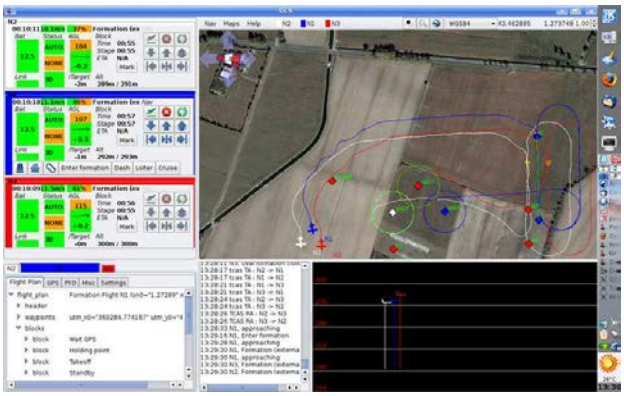

Fig. 2. Ground Control Station showing trajectories of 2 UAVs using the TCAS system

A real-time operating system based on ChibiOS is started to be used since 2015. The first implementation was mainly for adding a separate thread in order to make on-board logging at a high frequency. On going work is to divide all autopilot tasks into individual threads and manage everything according to priorities which will increase the safety and reliability.

C. Hardware

There exists over 20 different autopilot boards capable of running *Paparazzi*. Additionally, several mission based custom sensor boards have been designed under the *Paparazzi* project, such as *Meteo-Stick*[1].

The rest of this work highlights *Paparazzi*'s capacity to tackle the issues encountered during integration of UASs in the airspace. These issues have been grouped in three categories: modularity, congestion management and reliability. For each

[1]http://wiki.paparazziuav.org/wiki/MeteoStick_v1.10

TABLE I
COMPARISON OF POPULAR OPEN-SOURCE AND COMMERCIAL AUTOPILOTS.

Open-Source	Vehicle type	Hardware	Multi UAV	Flight Plan	Path Definition	Geofencing	Collision Avoidance	Latest stable release
Paparazzi	Fully configurable	varied	yes	scriptable	Formula Based	3-D	UAV TCAS	21-06-16
PixHawk	Fully configurable	specific	yes	scriptable	Circles Lines Patterns	3-D	none	06-08-16
Ardupilot	Copter Plane Rover	varied	alpha	scriptable	Circles Lines Patterns	2-D	none	22-06-16
OpenPilot	multicopter	specific	none	NA	NA	none	none	15-05-15
AeroQuad	multicopter	varied	none	NA	NA	none	none	31-01-13
Commercial								
Picollo	Copter Plane	specific	none	dynamic	Way Point	NC	NC	NC
MicroPilot	Copter Plane Rover Blimp	specific	yes	dynamic	Way Point	NC	NC	NC

category, *Paparazzi* offers a unique set of features to deal with the issues at hand.

D. Security

Latest micro processors used in the *Paparazzi* hardware has an additional feature that can be use for security augmentation. The cryptographic processor can be used to both encipher and decipher data using the DES, Triple-DES or AES (128, 192, or 256) algorithms. It is a fully compliant implementation of the following standards:

- The data encryption standard (DES) and Triple-DES (TDES) as defined by Federal Information Processing Standards Publication (FIPS PUB 46-3, 1999 October 25). It follows the American National Standards Institute (ANSI) $X9.52$ standard.
- The advanced encryption standard (AES) as defined by Federal Information Processing Standards Publication (FIPS PUB 197, 2001 November 26)

Since, with *Paparazzi*, the uav needs to be flashed after each modification, it is easy to rely on the random generator of the operating system (/dev/random under linux) to generate a key that will be flashed alongside with firmware, so this key will be :

- shared between UAV and groundstation
- regenerated for each flight, this make hostile decypher difficult
- in a multi UAV configuration, each UAV will have private cypher key

For the moment, the crypto driver for chibiOS is being written in order to be implemented into *Paparazzi* soon. This is generally a complicated task, however as the datalink bandwith is low, it becomes easier compared to a device driver.

E. Regular Use

Highly flexible architecture of the *Paparazzi* systems comes with the disadvantage of being too complex for a regular user such as a recreational (model aircraft) pilot. From the developer's point of view this is a great property, however beginners and regular users do not need to use all of the new features right away so they do not need to maintain the software at the latest version. Therefore the regular user will be using a recent stable release of the system, and usually will keep on using it without modification till he needs the new additional features coming from the developer community.

On the other hand, the developers will be able to use the software versioning system with GIT, maintained at Git-Hub, and can check out and contribute to the latest software version in the beta test status. The tools that the developers need in order to do that are not too different than the ones that are required for other software projects.

IV. MODULARITY

The current evolving nature of regulations and the variety of organizations in charge of the airspace rule making calls for flexible solutions to cope with these fruitful environments. *Paparazzi*, as an open source autopilot system, is largely modifiable through its modules to offer such flexible solutions.

A. Airspace Categorization

The UAS in the NAS project points to a performance-based routine access to all segments of the national airspace for all unmanned aircraft system classes, after the safety and technical issues are addressed thoroughly. As a start, NASA and faa seem to have a short term goal to integrate UAS in low-altitude airspace as early as possible. They further aim to accommodate increased demand safely, efficiently in the long term. NASA and faa seem to handle the airspace above 500 feet and the one below separately. European Aviation Safety Agency (EASA), tasked by the European Union, is planning a risk based approach, accommodating the UAS in the airspace under three different categories, low risk, specific and high risk [3]. Both regulators seem to categorize the airspace and scale regulatory needs according to some criteria. To answer

different needs of different categories, flexibility given by the high level of modularity of open source autopilot systems will be a handy tool. Customizability of the software and hardware depending on the airspace gives a chance for larger airspace community to utilize *Paparazzi* scaled for their specific needs. A larger community using the same system would lead to a natural evolution of the system toward a better design.

B. National Regulations

Circulation of UAS internationally is somewhat prevented by the Chicago convention unless an agreement holds between Contracting States [3]. ICAO is aiming to develop international standards and recommended practices to which the member states could refer to when developing their national civil aviation regulations. Even though a similar base is aimed, national aviation legislations will not be the same because of the different expectations of nations about UAS aviation. Thanks to the modular nature of the *Paparazzi* software and hardware suite, the functionality of the system could be enhanced according to the specific regulations held in the area of utilization.

C. Accommodating evolution of regulations

Prescriptive rules seem to cause some difficulties since the technical area on UAS systems develop rapidly [3]. Innovations both on the aircraft and the operation type of UAS will accelerate especially after the regulations are set. Thus, regulatory bodies call for refinable operational requirements and system architectures to evolve into a safer and efficient integration of UAS into airspace. The systems to cope with the regulations should also be modular and flexible in order not to be superseded by the innovations in the area. Thus, the aviation regulatory bodies aim to achieve designs with flexibility where possible, structure where needed. Having flexible hardware and software points to modularity, which is pretty much best supported via open source systems.

V. Congestion Management

According to *UAV Factory*, one of the large European UAS companies, "The future of the UAV industry is likely to be shaped by airspace congestion" [4]. Indeed, high level airspaces are getting crowded and large scale solutions, such as NextGen (US) or SESAR-JU (EU), are necessary to increase airspace capacity while maintaining the current safety levels. However, there is no such management solution existing for Very Low Level (VLL). Yet, large projects like Amazon's Prime Air and Google's Project Wing are already waiting to populate the VLL airspace.

Part of the congestion management problem is to avoid conflicts, and more importantly collisions, between UAS through strategic deconfliction and safety nets. Another mission of the congestion management system is to make sure that UAS do not go where they are not supposed to go, thus requiring geofencing. In order to implement the previously mentioned systems, the UAS autopilot needs to be able to perform complex operations, e.g. static waypoints following is likely to be insufficient. In the following, we divide these issues into four topics of interest: 4-D trajectory management, geofencing, safety nets, complex operations, and show how the *Paparazzi* system addresses them.

A. 4-D Trajectory Management

As noted in [5], 4-D trajectories will be central in future airspace management methods. The principle of 4-D trajectory management is to have every UAS broadcast its trajectory up to some time horizon and receive Unmanned Aircraft System Traffic Management (UTM)'s clearances under the form of trajectories. The trajectory information contains a path, the series of points through which the UAS will pass, and times associated to each of these points. Thanks to this information, the idea is to perform pro-active deconfliction, as explained by Thomas et al. in [6]. In clear, it implies that UTM detects future conflicts along the trajectories of all UAS and deconflicting them as safely and early as possible. This kind of approach requires the autopilot to represent UAS motions with trajectories and to be able to transmit them, which is the case in *Paparazzi*.

Paparazzi originally supports a basic description of trajectories based on circles and straight lines, but recent updates allow it to process advanced trajectories as well. Indeed, *Paparazzi* can represent trajectories as functions in 2-D (x, y), 3-D (x, y, z) or 4-D (x, y, z, t). The only requirement is for the function to be differentiable at least two times on every point. The function and its derivatives analytical formulas are computed offline and can then be quickly evaluated online. Moreover, gains can be tuned to adjust the convergence speed from the UAS starting point to the given trajectory. These gains influence the convergence path to a given trajectory and the resulting trajectory (convergence + commanded) can be computed offline given initial conditions and the commanded trajectory. This is of particular interest for UTM as it allows the UAS to provide not only its trajectory over time but also how it will reach this trajectory from its starting point.

It is important to note that UTM will have to solve conflicts between manned and unmanned aircraft. So an air traffic controller is needed in the loop with an appropriate display for 4-D trajectories and a communication link with both manned and unmanned aircraft.

B. Safety Nets

Trajectory deconfliction is the first step to manage congestion, however safety nets are also part of the congestion management. Indeed, safety nets such as self-separation and collision avoidance allow UAS to fly close to each other while preserving an acceptable safety level. Though *Paparazzi* does not include self-separation algorithms, it do contains a light version of the TCAS II collision avoidance system. It considers intruders one at a time and is capable of coordinated avoidance maneuvers. There is no strengthen resolution as the resolutions are not speed rates but an objective altitude to reach as fast as possible.

TABLE II
TYPICAL VALUES FOR MAIN VARIABLES OF THE *Paparazzi* TCAS MODULE.

TAU_TA	TAU_RA	DMOD	ALIM
10s	6s	15m	10m

Values such as distance thresholds have been adapted to suite the performances of small UAS. Table II shows example values used for fixed-wing UASs. Keep in mind that the *Paparazzi* philosophy is to be configurable, so these parameters can be easily changed from a configuration file.

On top of the TA and RA provided by the TCAS, a module has been recently developed to input data from traffic services and display them into *Paparazzi*'s ground control station, thus providing situational awareness to the remote pilot. Preliminary test have been done using opensky-network to display traffic around a given area, based on these information the remote pilot can effectively perform conflict resolution.

C. Geofencing

Keeping UAS away from each other is an important point. But keeping them out of forbidden areas is also crucial. Geofencing allows determining no-fly zones where the UAS should not enter. To accommodate land owners while managing traffic and limiting congestion, Foina et al. [7] proposed a participative dynamical airspace management method: the airparcel model. It allows land owners to authorize/forbid flights over their lands through a web interface. However, this type of approach asks from the UASs to be able to handle dynamical geofencing. Plus, though initially this model considers only cuboid parcels, the need for more precise airspace shapes may emerge making 3-D geofencing a need.

This type of model can be handled by *Paparazzi* by defining geofenced zones manually or programmatically. The zone is defined as a simple geometrical shape (e.g. a circle, a polygon, etc.) plus altitude limits thus effectively creating a 3-D geofencing. These zones can be static, activated dynamically from the ground station or from the flight plan with associated conditions.

D. Complex Operations

Having 4-D trajectory management, safety nets and geofencing is useless if the UASs cannot follow the instruction from UTM regarding these tools. Indeed, new UTM paradigms imply being able to change flight plans dynamically to answer to UTM demands. In [8] two types of complex operations examples are mentioned: space transition corridors and temporary flight restriction. Both these airspace management methods require from the UAS to be able to modify its flight plan according to new UTM instructions.

Modifying the flight plan dynamically is not possible, and not desirable, in *Paparazzi*. However, appropriate response to these complex operations can be provided through conditional flight plans. These enable the UAS to follow different courses of action depending on given parameters. These parameters are defined offline by the user and can then be modified online by values coming from sensors or from UTM. This allows intelligent reaction to exterior instruction, in particular it can be used by the user to give UTM control on portions of the flight.

VI. RELIABILITY

Improvement of the reliability of the flight is considered to be one of the main goals for integrating military UAVs into civil airspace according to Unmanned systems roadmap by US Office of the Secretary of Defense, DoD [9]. Compared to manned counterparts, UAVs experienced failures with a frequency of two orders of magnitude more in the military domain. Although this changed last years with the technological improvements, making the UAVs as reliable as early manned military aircraft, it seems not enough from the DoD perspective. This can be realized by checking the biggest chunk of control technologies budget for research and development, which is health management and adaptive control.

To achieve a safe flight is not an easy task considering the unknowns of the systems hardware, environment and possible system faults and failures to emerge. Also, increasing demand on cost effective systems, resulting in smaller sensors and actuators with less accuracy, impose the software to achieve even more. The expectation that UAVs should be less expensive than their manned counterparts might have a hit on reliability of the systems. Cost saving measures other than the need to support a pilot/crew onboard or decrement in size would probably lead to decrease in system reliability.

A. Small and Medium Enterprise (SME)s and Certification Costs

Utilizing drones for quicker and cheaper deliveries could be rewarding for SMEs since cost per mile of a drone is less then 1 / 30 of the average diesel truck. Being an early bird might put the SMEs in an advantageous position, considering the increase in the capabilities of the drones with inevitable acceleration thrusted by research activities and their widened application areas. Nevertheless, the fairly cheap access to drones and their relatively cheap utilization cost does not seem to be enough to put them to air right now due to the heavy cost of certification and regulatory hurdles [10]. In this concern, capable open source solutions could be a good way to loosen the tie. Otherwise, SMEs, an important factor in drone business might not survive. Even worse, they might operate them without relevant permission, scarifying a substantial fine as reported by Civil Aviation Authority (CAA). This will compromise security in the system contradicting the hopes for reliable integration of drones into airspace. As an open source autopilot system, *Paparazzi* is a known platform for computer scientist who want to test the viability of complex software systems. Thus parts of *Paparazzi* code (Ground Control Station (GCS)) have been formally proven [11] and stable version have thousands of flight heritage. Furthermore, although Paparazzi is not certified, to be able to have access to the certified airspace, the users can built up their configuration on readily

and freely available Paparazzi system, which might reduce the cost to have a certified system to reach specific airspaces.

B. Individuals and Education

Individuals, as well as SMEs, suffer from the same budget constraints. Personal UAV usage counts for a substantial amount of the drone ecosystem. Both US and European authorities mention the importance of individuals in utilizations of drones. There is a community with a passion of aviation and potential, but most probably not very experienced. *Paparazzi* offers a whole community to help/educate these beginners through its forums enriched by rising number of its users. A rich documentation is available through the Wikipedia page, encouraging users to self-teach.

C. Flight Heritage for Risk Assessment

Drone industry being extremely innovative, technical developments could supersede the prescriptive rules as regulations. Thus a solution might be to follow a risk based approach rather that to have strict rules to cope with. Predicted regulations in Europe seems to evolve under different categories dedicated to specific operation risks. Flight heritage and occurrence reporting is expected to be an inevitable part of safety risk assessment to achieve reliable flight. Wide utilization of *Paparazzi* and real time connection to ethernet could offer reliable and practical solutions to report occurrences.

D. Support for real time planning and onboard vehicle automation

To access low-altitude airspace with the use of small unmanned aircraft safely, an important ability could be to implement real-time planning and on-board vehicle automation. Amazon offers that this approach will allow some flexibility to adapt to variable situations such as weather changes, severe winds or any other emergency needs. *Paparazzi*, has a real-time planning ability already implemented, letting the user change the trajectory in real-time through its ground control station via different strategies. The user could switch between trajectories already available or even drag the waypoints to his/her preference. The automation of the vehicles is handled in different stages. For now, the most used modes of autonomy is no autonomy, assisted mode and the fully autonomous mode. No autonomy mode, or manual mode, gives whole responsibility of the aircraft dynamics to the pilot through the RC link. The assisted mode closes some attitude control loops, giving some stability to aircraft. This will ease the challenges of piloting which could be a great advantage especially for the unexperienced pilots. The fully autonomous mode handles both heading and navigation through selected trajectories. In case of an emergency, the pilot take the control of the aircraft anytime, through the RC link. Features already implemented on *Paparazzi*, such as geofencing, go home, and its ease in adding/modifying thresholds to various variables, support safety procedures.

VII. Future Evolutions

As many open source projects *Paparazzi* is in constant evolution through regular contributions. With its large community, it is hard to keep track of all the ongoing projects. In this section we present three features being currently explored and developed at ENAC's labs.

Though there is no life a stake, avoiding UAS collisions is desirable for safety and operational reasons. *Paparazzi*'s current TCAS-like collision avoidance allows operating numerous UAVs with little risk of collision between them. However, future missions will include more and more UAVs in bounded size airspaces. In this context, an efficient collision avoidance algorithm is desirable to allow closer operations with an acceptable amount of nuisance alerts. Because it relies on advanced logics and can be adapted to different types of performances, we have chosen to implement the ACAS Xu algorithm. Though this standard's definition just started, the baseline is already determined and may allow developing a simplified version for micro UAVs.

Regarding geofencing, most current methods use boundaries in the horizontal plane in which the UAV cannot enter. However, the utilization of the VLL airspace is likely to demand more sophisticated methods to handle 3-D geofencing where a 3-D no-fly shape can be described. Though *Paparazzi* can emulate 3-D geofencing, it cannot use full 3-D models yet. Work is underway to extend the current geofencing to allow loading terrain models and padding them with arbitrary shaped no-fly zones.

Finally, conventional procedures to handle failures onboard could be one of three approaches in order to achieve safe flight. First one is the fail operational systems which are made insensitive to any single point component failure. The second approach is the failsafe systems where a controlled shut down to a safe state is practiced whenever a critical fault is pointed out by a sensor. The third approach is fault tolerant control systems in which components of the system are monitored and action taken in whenever needed. The strategy is most probably to try to keep the system availability and accept reduced performance. For *Paparazzi* autopilot system, there is ongoing research to implement indigenous fault detection and diagnosis module to enable reconfigurable controller loops.

VIII. Conclusion

The introduction of UASs in the VLL airspace comes with numerous challenges. Though various ways to address them have been proposed, there is still no certainty about how it will be done in the end. One sure thing is that it will involve two main actors: UTM and UASs. This work focused on the later.

When it comes to autopilot systems, there exist a wide range of solutions. However, in the context of a quickly evolving regulation and technology, we showed that it may be beneficial to rely on an open source solution. Indeed, the community around it and the ability to adapt it to one's needs are precious assets. Among open source solutions, the *Paparazzi* autopilot system distinguishes itself with a unique set of features that, we have

showed, allow tackling many of the existing challenges in the UASs integration. The most interesting feature being its linux-like philosophy which confers it an unparalleled level of configurability. With a complete control, the user is not limited by the software and nothing stops him from doing the smartest designs.

Paparazzi is a living project and keeps evolving to satisfy it community, both from the research and industry. New features are added on a regular basis and the shape it will take in the future only depends on the new members that join the project.

ACKNOWLEDGMENT

This work was supported by the ENGIE Ineo - Sagem RPAS Chair. The authors would like to thank Gautier Hattenberger and Hector Garcia de Marina for sharing their expertise in *Paparazzi*.

REFERENCES

[1] C. Forrest. (2015) 12 drone disasters that show why the faa hates drones. [Online]. Available: http://www.techrepublic.com/article/12-drone-disasters-that-show-why-the-faa-hates-drones

[2] U. E. Franke. (2015) Civilian drones: Fixing an image problem? [Online]. Available: http://isnblog.ethz.ch/security/civilian-drones-fixing-an-image-problem

[3] "Introduction of a regulatory framework for the operation of drones," Advance Notice of Proposed Amendment 2015-10, Office of the Secretary, 2005.

[4] L. Probst, L. Frideres, B. Pedersen, and C. Martinez-Diaz. (2015) Uav systems for civilian applications. [Online]. Available: http://ec.europa.eu/DocsRoom/documents/13430

[5] H. Erzberger and R. A. Paielli, "Concept for next generation air traffic control system." *Air Traffic Control Quarterly*, vol. 10, no. 4, pp. 355–378, 2002.

[6] E. Thomas and O. Bleeker, "Options for insertion of rpas into the air traffic system," in *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*. IEEE, 2015, pp. 5B4–1.

[7] A. G. Foina, C. Krainer, and R. Sengupta, "An unmanned aerial traffic management solution for cities using an air parcel model," in *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*. IEEE, 2015, pp. 1295–1300.

[8] C. A. Wargo, G. Hunter, K. Leiden, J. Glaneuski, B. Van Acker, and K. Hatton, "New entrants (rpa/space vehicles) operational impacts upon nas atm and atc," in *Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th*. IEEE, 2015, pp. 5B2–1.

[9] "Unmanned systems roadmap 2005 - 2030," Report, Office of the Secretary of Defence, DoD, 2005.

[10] "Unmanned aerial vehicle reliability study," Report, Office of the Secretary of Defence, DoD, 2005.

[11] M. M. van Paassen, M. L. Bolton, and N. Jimnez, "Checking formal verification models for human-automation interaction," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Oct 2014, pp. 3709–3714.

Flight Simulation of a MAKO UAV for Use in Data-Driven Fault Diagnosis

Elgiz Baskaya*, Murat Bronz, and Daniel Delahaye
ENAC

ABSTRACT

Last decade witnessed the rapid increase in number of drones of various purposes. This pushes the regulators to rush for safe integration strategies in a way to properly share the utilization of airspace. Accommodating faults and failures is one of the key issues since they constitute the bigger chunk in the occurrence reports available. The hardware limitations for these small vehicles point the utilization of analytical redundancy rather than the usual practice of hardware redundancy in the conventional flights. In the course of this study, fault detection and diagnosis for aircraft is reviewed. Then a nonlinear model for MAKO aircraft is simulated to generate faulty and nominal flight data. This platform enables to generate data for various flight conditions and design machine learning implementations for fault detection and diagnosis.

1 INTRODUCTION

Unmanned Aircraft Systems (UAS) are becoming more efficient platforms everyday for scientific/commercial domains offering benefits in terms of cost, flexibility, endurance as well as realizing missions that would be impossible with a human onboard. Increasing usage of these vehicles for a variety of missions, such as defense, civilian tasks including transportation, communication, agriculture, disaster mitigation applications pushes demand on the airspace. Furthermore, this congestion is predicted to accelerate with the growing diversity of these vehicles[1].

Improvement of the reliability of the flight is considered to be one of the main goals for integrating UAVs into civil airspace according to Unmanned systems roadmap by US Office of the Secretary of Defense, DoD [2]. To achieve a safe flight is not an easy task considering the unknowns of the systems hardware, environment and possible system faults and failures to emerge. Also, increasing demand on cost effective systems, resulting in the smaller sensors and actuators with less accuracy, impose the software to achieve even more. The expectation that UAVs should be less expensive than their manned counterparts might have a hit on reliability of the system. Cost saving measures other than the need to support a pilot/crew onboard or decrement in size would probably lead to decrease in system reliability.

Systems are often susceptible to faults of different nature. Existing irregularities in sensors, actuators, or controller could be amplified due to the control system design and lead to failures. A fault could be hidden thanks to the control action [3].

Under the research and development programs and initiatives identified by DoD in order to develop technologies and capabilities for UAS, the biggest chunk in control technologies is the health management and adaptive control with a budget of 74.3 M dollars. Other safety features such as validation and verification of flight critical intelligent software is the second with 57.8 M dollars [2].

The widely used method to increase reliability is to use more reliable components and/or hardware redundancy. Both requires an increase in the cost of the UAS conflicting one of the main reasons of UAS design itself band consumer expectations [4]. To offer solutions for all different foreseen categories of airspace, a variety of approaches should be considered. While hardware redundancy could cope with the failure situations of UAVs in the certified airspace, it may not be suitable for UAVs in open or some subsets of specific categories due to budget constraints. Analytical redundancy is another solution, may be not as effective and simple as hardware redundancy, but relies on the design of intelligent methods to utilize every bit of information onboard aircraft wisely to deal with the instances.

There are three approaches to achieve safe FTC in standard flight conventions. First one is the fail operational systems which are made insensitive to any single point component failure. The second approach is the fail safe systems where a controlled shut down to a safe state is practiced whenever a critical fault is pointed out by a sensor. The level of degradation assures to switch to robust (alternate) or direct (minimal level of stability augmentation independent of the nature of the fault) mode. Switching from nominal mode to the robust and direct modes leads to a decrease in the available GNC functions. This causes a degradation in ease of piloting. And also some optimality conditions could have been compromised. The third approach is fault tolerant control systems in which redundancy in the plant and the automation system is employed to design software that monitors the components and takes in action whenever needed. The strategy is most probably to try to keep plant availability and accept reduced performance [5].

*Email addresses: elgiz.baskaya@enac.fr, murat.bronz@enac.fr, daniel.delahaye@enac.fr

RECONFIGURE project of FP7 [6] aims to attack at this problem of piloting degradation and optimality compromisation by attacking Flight Parameter Estimation (FPE) which is the online estimation of aircraft parameters, FDD and FTC in case of off-nominal events [7] They utilize a black box nonlinear model of aircraft and The project uses some outputs of a previous FP 7 project ADDSAFE leaded by Deimos Space [8].

2 Methods for FTCS

Since fault tolerant control is comprised of a set of different disciplines and a relatively new topic, the terminology is not solid. FDI could be a proper example to this ambiguity. In some works, it stands for Fault Detection and Isolation while in some other Fault Detection and Identification, which could also named after Fault Detection and Diagnosis, meaning that identification is added to Fault Detection and Isolation [9].

One of the first attempts to unify the terminology is carried out by IFAC SAFEPROCESS technical committee in 1996 and published by [10]. Fault, failure, and the methodology to handle those such as fault detection, fault isolation, fault identification, fault diagnosis and supervision terms explained separately to avoid the ongoing ambiguity in this field. Although fault detection methods are clearer in the work, difference between the methods for two steps of fault diagnosis, namely the fault isolation and fault identification is not very obvious.

Among different categorizations for the fault tolerance, there are options to handle faults on-line or off-line. Employing fault diagnosis schemes on-line is a way to achieve fault tolerance. In this case, as soon as a fault detected, a supervisory agent is informed via a discrete event signal. Then accommodation of the faults are handled either with the selection of a predetermined controller for the specific fault case, or by designing the action online with real-time analysis and optimization [5].

Another common categorization of FTCS is passive and active FTCS. In passive FTCS, the flight controller is designed in such a way to accommodate not only the disturbances but also the faults. Most of the times it a robust controller and does not require a diagnosis scheme. 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 [4]. 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, robust controller most probably fails.

Even with a long list of available methods, aerospace industry has not implemented FTC widely, except some space systems, due to the evolving nature of the methods, the tricks coming with the nonlinear nature of the problem, design complexity and high possibility of wrong alarms in case of large disturbances and/or modeling uncertainties. So the already

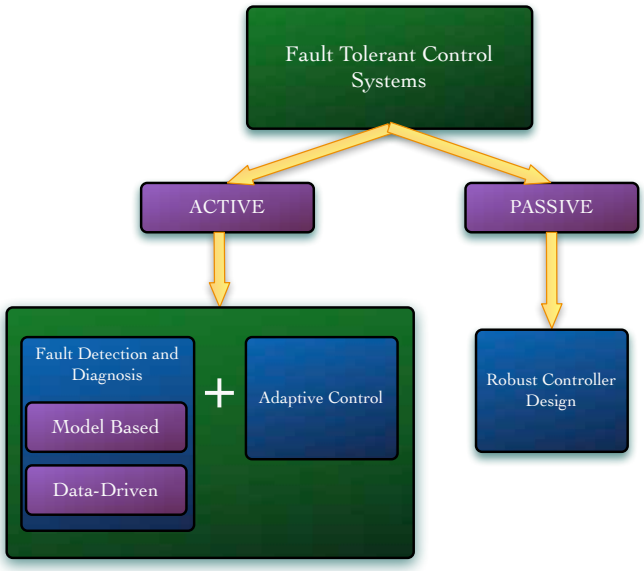

Figure 1: Variations of fault tolerant control systems

carried reliability measures concerning the hardware redundancy is now the preferred way because of its ease and maturity being implemented on various critical missions with considering human lives.

3 Fault Detection and Diagnosis

FDD is handled in two main steps; fault detection and fault diagnosis. Fault diagnosis encapsulates fault isolation and fault identification. The methods for detection and diagnosis are investigated for their frequency of utilization separately for sensor, actuator, process and controller faults in [10]. FDD should not only be sensitive to the faults but also robust to the model uncertainties and external disturbances.

Two distinct options to proceed in analytical redundancy are the model based approaches and data-driven approaches. They form the two ends of a continuous solution set line, so utilizing them in a combination might end up with better solutions. 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.

This work constitutes the basis for our research on fault detection. The idea to simulate the data using the MAKO model given here first, rather than utilizing flight data, is to start small in order to isolate some probable consequences such as the probable effect of the controller on the diagnosis. Detection of faults from real data is a challenging goal to start with as can be seen in Figure 2 real data accelerometer readings showing that it seems impossible to classify in case of a

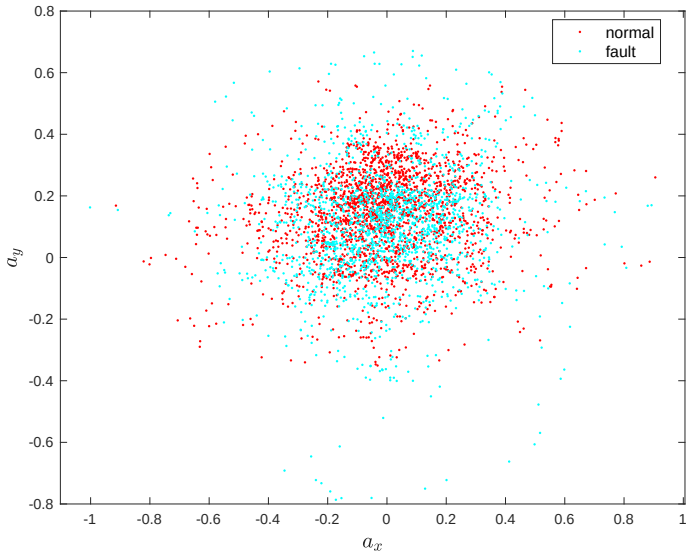

Figure 2: Accelerometer data readings a_x vs a_y

Figure 3: MAKO

fault with one of the control surfaces is 50% less efficient.

Most of the FDI algorithms are implemented to open-loop systems, ignoring the probable influences of the controller might cause on the detection performance [11]. Here the system is open-loop as well. So we follow a step by step approach and hope to end with a more realistic case in the future, in which real flight data is utilized and diagnosis is achieved on-line aside a functioning controller. Here we present a literature survey for FTC of drones followed by efforts to deliver a full drone simulation which will serve as an environment to simulate measurements. A MAKO simulation is given in Matlab script which is freely available though the GIT [1] platform, using specifications MAKO, stability derivatives, aerodynamic force derivatives generated by AVL.

4 Methodology and Simulations

In this study, first, a model of an aircraft is simulated. This model, will not be used for the design of FDI algorithms, but instead will be utilized to test them. Nonlinear aircraft flight dynamics for translational and attitude motion can be given as a system of first order differential equations

$$\dot{\boldsymbol{x}}_n = \boldsymbol{C}_b^n \boldsymbol{v}^b \tag{1}$$

$$\dot{\boldsymbol{v}}^b = \frac{1}{m}\left[m\boldsymbol{g}^b + \boldsymbol{F}_t^b + \boldsymbol{F}_a^b\right] - {\boldsymbol{\omega}_{b/n}^b}^{\times}\boldsymbol{v}^b \tag{2}$$

$$\dot{q}_0 = -\frac{1}{2}\boldsymbol{q}_\nu^T \boldsymbol{\omega}_{b/n}^b \tag{3}$$

$$\dot{\boldsymbol{q}}_\nu = \frac{1}{2}\Big(\boldsymbol{q}_\nu^{\times} + q_0 \boldsymbol{I}_3\Big)\boldsymbol{\omega}_{b/n}^b \tag{4}$$

$$\boldsymbol{J}\dot{\boldsymbol{\omega}}_{b/n}^b = \boldsymbol{M} - {\boldsymbol{\omega}_{b/n}^b}^{\times}\boldsymbol{J}\boldsymbol{\omega}_{b/n}^b \tag{5}$$

[1] https://github.com/benelgiz/curedRone/tree/MAKOmodel

where $\boldsymbol{x}_n \in \mathbb{R}^3$ is the position of the center of mass of UAV in navigation frame $\mathcal{N}$, $\boldsymbol{v}^b$ is the velocity of the center of mass of UAV in body frame $\mathcal{B}$, $\boldsymbol{q} = [q_0, \boldsymbol{q}_v^T]^T \in \mathbb{R}^3 \times \mathbb{R}$ is the unit quaternion representing the attitude of the body frame $\mathcal{B}$ with respect to navigation frame $\mathcal{N}$ expressed in the body frame $\mathcal{B}$, $\boldsymbol{\omega}_{b/n}^b$ is the angular velocity of the body frame $\mathcal{B}$ with respect to navigation frame $\mathcal{N}$ expressed in the body frame $\mathcal{B}$, $\boldsymbol{J} \in \mathbb{R}^{3\times3}$ is the positive definite inertia matrix of the drone, $\boldsymbol{M} \in \mathbb{R}^3$ represents the moments acting on the drone, $\boldsymbol{C}_b^n$ is the direction cosine matrix which transforms a vector expressed in the body frame to its equal expressed in the navigation frame, $\boldsymbol{I}_3 \in \mathbb{R}^{3\times3}$ is the identity matrix, $\boldsymbol{F}_t^b \in \mathbb{R}^3$ is the thrust force expressed in the body frame, $\boldsymbol{F}_a^b \in \mathbb{R}^3$ are the aerodynamic forces given in the body frame. The navigation frame is assumed to be a local inertial frame in which Newton's Laws apply. The notation $\boldsymbol{x}^{\times}$ for a vector $x = [x_1 \quad x_2 \quad x_3]^{\mathrm{T}}$ represents the skew-symmetric matrix

$$\boldsymbol{x}^{\times} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{6}$$

The stability derivatives and aerodynamic force coefficients are generated by AVL and given in Appendix A. AVL is an open source program developed at MIT and uses vortex-lattice method for the aerodynamic and stability calculations. The output of the program is linearized at a selected condition, therefore all the coefficients are calculated around the equilibrium point at $14m/s$ cruise flight condition. The center of gravity is located at $X_{CG} = 0.295\,m$, which corresponds to a $8\,\%$ of positive static margin that has been flight tested.

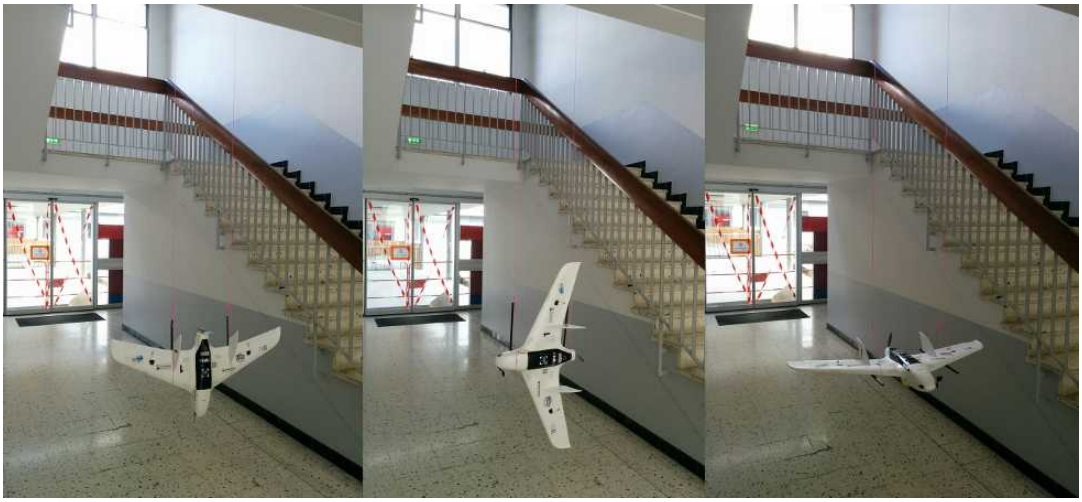

Figure 4: Moments of inertia measurements for each axis, I_{xx}, I_{yy}, I_{zz}.

As an addition to the aerodynamic coefficients and stability derivatives, it is useful to have the moments of inertia of the aircraft so that one can use the model in a simulator. For that purpose, the aircraft is hanged by two strings, at different orientations, as shown in Figure 4, and measurements performed by timing the oscillation period for each axis. The resultant moment of inertias are given Table 1.

Further, the equations for calculation of forces and moments are given in Appendix B to simulate translational and rotational motion of a MAKO UAV.

The input vector can be written as $\boldsymbol{u}(t) \in \mathbb{R}^3$

$$\boldsymbol{u}(t) = \left[\delta_a \ \delta_e \ n\right]^{\mathrm{T}} \tag{7}$$

Here δ_a aileron deflection angle in degrees, δ_e elevator deflection angle in degrees, n engine speed in rev/s.

To validate the written translational and attitude motion dynamics and kinematics, MATLAB Simulink *6DOF* block has been utilized. This block accepts inputs as the force and moment and outputs the states of aircraft motion Fig. 6. To compare the generated model and Simulink *6DOF* block, forces and moments have been calculated via equations and constants given in Appendix A and Appendix B. The simulated states from the model script have been saved in advance and called from Simulink by *From Workspace* blocks then compared with the *6DOF* outputs. The difference found to be negligible indicating the validity of the model.

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

$$\boldsymbol{u}(t) = \boldsymbol{E}\boldsymbol{u}_c + \boldsymbol{u}_f \tag{8}$$

where $\boldsymbol{u}_c$ is the desired control signal, $\boldsymbol{E} = diag(e_1, e_2, e_3)$ is the effectiveness of the actuators where $0 \leq e_i \leq 1$ with $(i = 1, 2, 3)$ and $\boldsymbol{u_f}$ additive actuator fault. This model makes it possible to simulate all four types of actuator faults shown in Fig. 5.

The measurements are simulated using the statistics of the hardware in the house. The sensor suit simulated is the InvenSense MPU-9250 Nine-axis (Gyro + Accelerometer + Compass) MEMS MotionTracking Device.

$$\boldsymbol{z}_{gyro} = \boldsymbol{k}_{gyro}\boldsymbol{\omega}^b_{b/i} + \boldsymbol{\beta}_{gyro} + \boldsymbol{\eta}_{gyro} \tag{9}$$

$$\boldsymbol{z}_{acc} = \boldsymbol{k}_{acc}\boldsymbol{\omega}^b_{b/i} + \boldsymbol{\beta}_{acc} + \boldsymbol{\eta}_{acc} \tag{10}$$

Here $\boldsymbol{\beta}$ is the bias, and $\boldsymbol{\eta}$ is the zero mean Gaussian process with $\boldsymbol{\sigma}^2$ variance and given in Table 5. For faulty and normal measurement values, drone model simulation which outputs the measurements as well should be run twice with different control surface input values. As an example, a faulty situation can be that even the controller gives an desired output of 4 degrees to the control surface, the control surface might have stuck at 1 degrees. Generated set of measurements can be visualized in feature space one by one. Such an example is the normalized accelerometer measurement components plotted Fig. 7.

It is always important to visualize the features to have a grasp of data structure. For that reason, available observations forms the 6-dimensional pattern space, $\boldsymbol{z} \in \mathbb{R}^6$ can be visualized in pairs to observe. There are further methods to visualize multidimensional data such as Tours methods [12, 13, 14], and GGobi data visualization system [15].

In this study, dimensionality reduction technique called Principle Component Analysis (PCA) is used for visualization. In PCA, the idea in general is to map the feature vector, $\boldsymbol{x} \in \mathbb{R}^n$ to a lower dimensional space where the new feature set will be represented by $\boldsymbol{z} \in \mathbb{R}^k$. Fig. 8 shows the resulted most significant elements for a mapped feature space from six dimensional feature vector to two. The structure of the data

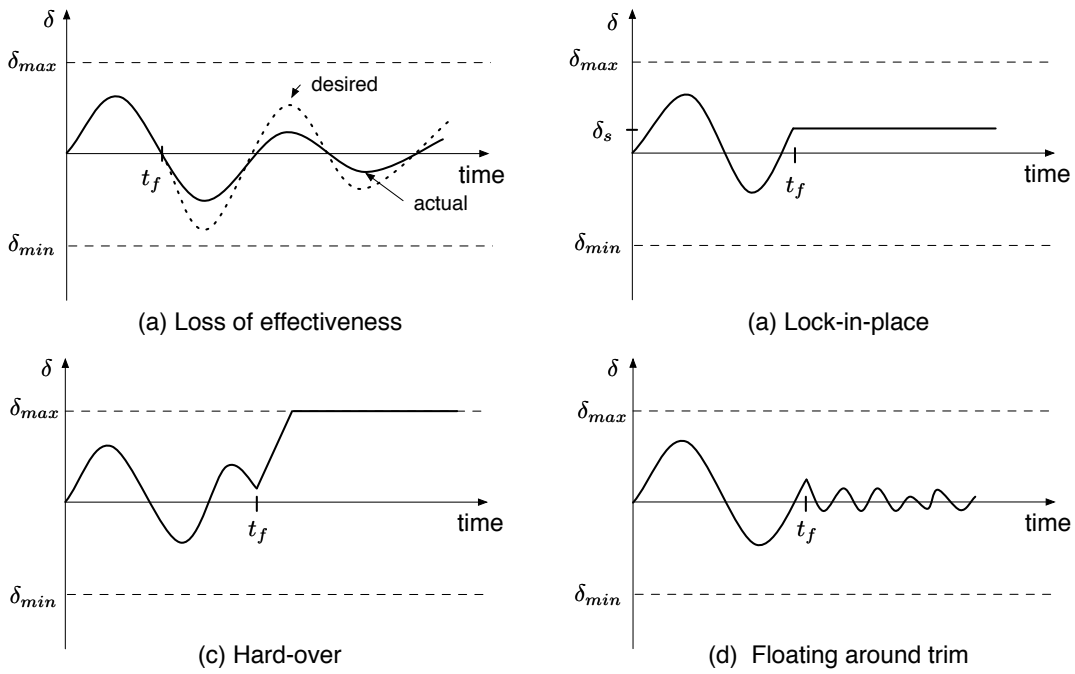

Figure 5: Common actuator faults [3]

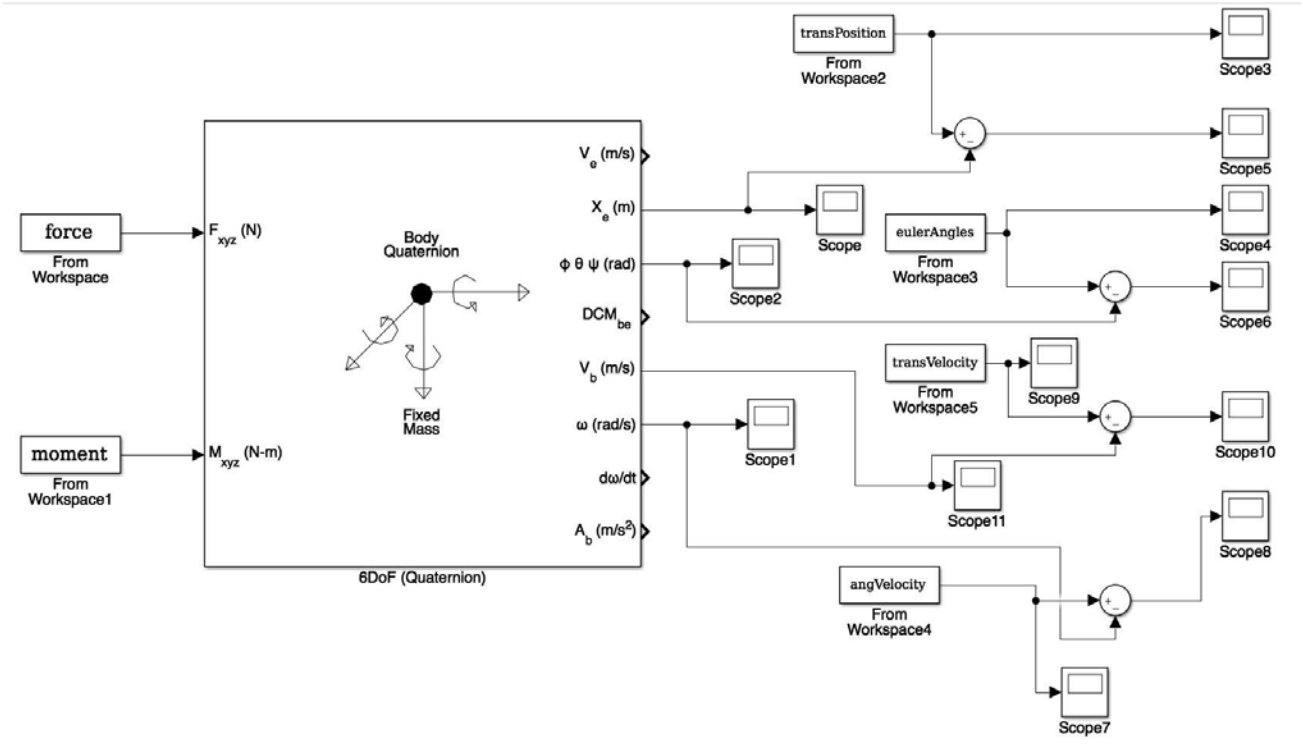

Figure 6: Validation with Simulink 6DOF aircraft model

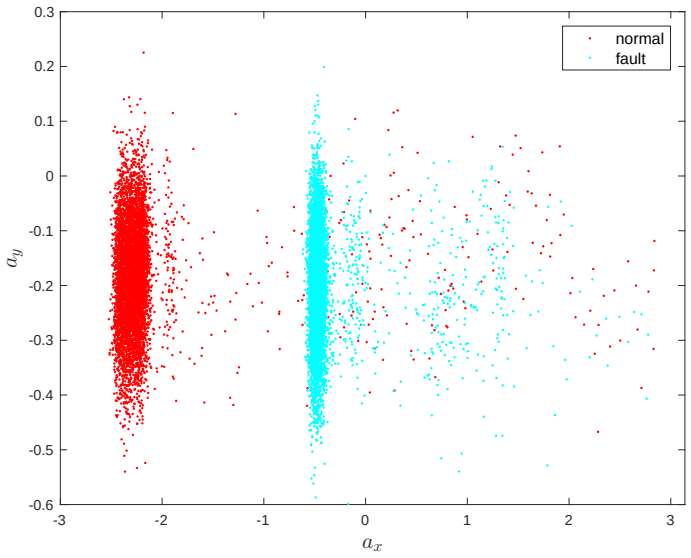

Figure 7: Accelerometer simulation a_x vs a_y

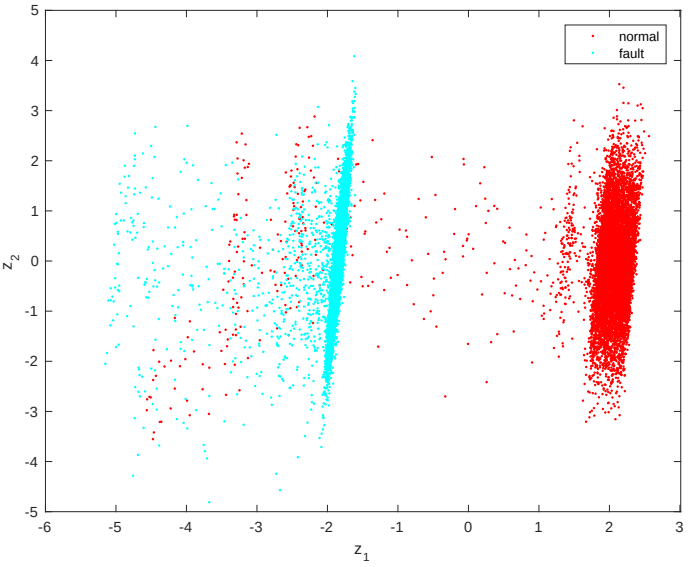

Figure 8: Reduced feature space z_1 vs z_2

gives insight for the selection of some parameters or kernels for the purpose of classification. Here, it seems that a linear kernel is satisfactory by discarding the outliers. Another point is that the classifier might need a nice tuning due the the presence of outliers. The learning phase utilizes data including outliers and preciseness to fit the model to each of the data might end up an overfitted model, resulting in worse performance to generalize to new data coming in the prediction phase.

5 Conclusion

In this work, first a review on fault tolerant control for UAVs is given by pointing out its importance on today's challenging task of safe integration of drones into airspace. Data-driven methods for fault diagnosis is aimed to avoid the burden of modeling each craft especially considering for small drones it is not very realistic for most of the applications to have an accurate model for a variety reasons such as cost. AVL program is used to generate the coefficients for MAKO and a full simulation is realized. Statistics of the sensor suite in house is used for simulation of accelerometer and gyro data. For a preliminary investigation on data, six dimensional feature space is mapped to two dimensions via PCA for visualization purposes. The data shows that a linear kernel might be satisfactory for the purpose of two class classification. Due to the presence of outliers, fine tuning or using optimization techniques could be needed to avoid overfitting or under fitting during the learning phase of the classification problem.

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, Xavier Paris, Michel Gorraz 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] Elgiz Baskaya, Guido Manfredi, Murat Bronz, and Daniel Delahaye. Flexible open architecture for uass integration into the airspace: Paparazzi autopilot system. In *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*, pages 1–7. IEEE, 2016.

[2] Unmanned Systems Roadmap 2005 - 2030, 2005.

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

[4] Plamen Angelov. *Sense and avoid in UAS: research and applications*. John Wiley & Sons, 2012.

[5] Mogens Blanke, Christian W Frei, Franta Kraus, Ron J Patton, and Marcel Staroswiecki. What is fault-tolerant control. In *Preprints of 4th IFAC Symposium on Fault Detection Supervision ans Safety for Technical Processes, SAFEPROCESS*, pages 40–51, 2000.

[6] Philippe Goupil, Josep Boada-Bauxell, Andres Marcos, Paulo Rosa, Murray Kerr, and Laurent Dalbies. An overview of the fp7 reconfigure project: industrial, scientific and technological objectives. *IFAC-PapersOnLine*, 48(21):976–981, 2015.

[7] RECONFIGURE FP7 Project. reconfigure.deimos-space.com/, Accessed: 2016-07-19.

[8] ADDSAFE FP7 Project. http://addsafe.deimos-space.com/, Accessed: 2016-07-09.

[9] Youmin Zhang and Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual reviews in control*, 32(2):229–252, 2008.

[10] Rolf Isermann and Peter Ballé. Trends in the application of model-based fault detection and diagnosis of technical processes. *Control engineering practice*, 5(5):709–719, 1997.

[11] Rohit Pandita, József Bokor, and Gary Balas. Closed-loop performance metrics for fault detection and isolation filter and controller interaction. *International Journal of Robust and Nonlinear Control*, 23(4):419–438, 2013.

[12] Daniel Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM journal on scientific and statistical computing*, 6(1):128–143, 1985.

[13] Dianne Cook and Andreas Buja. Manual controls for high-dimensional data projections. *Journal of computational and Graphical Statistics*, 6(4):464–480, 1997.

[14] Dianne Cook, Andreas Buja, Javier Cabrera, and Catherine Hurley. Grand tour and projection pursuit. *Journal of Computational and Graphical Statistics*, 4(3):155–172, 1995.

[15] Dianne Cook and Deborah F Swayne. *Interactive and dynamic graphics for data analysis: with R and GGobi*. Springer Science & Business Media, 2007.

[16] Murat Bronz and Gautier Hattenberger. Aerodynamic characterization of an off-the-shelf aircraft via flight test and numerical simulation. In *AIAA Flight Testing Conference*, page 3979, 2016.

[17] Murat Bronz, Hector Garcia de Marina, and Gautier Hattenberger. In-flight thrust measurement using on-board force sensor. In *AIAA Atmospheric Flight Mechanics Conference*, page 0698, 2017.

[18] Jean-Philippe Condomines. *Développement d?un estimateur d?état non linéaire embarqué pour le pilotage-guidage robuste d?un micro-drone en milieu complexe*. PhD thesis, INSTITUT SUPERIEUR DE L'AERONAUTIQUE ET DE L'ESPACE (ISAE), 2015.

APPENDIX A: MAKO COEFFICIENTS

Table 1: General specifications of MAKO [16]

Parameter	Value	Definition
Wing span	1.288	$[m]$
Wing surface area	0.27	$[m^2]$
Mean aero chord	0.21	$[m]$
Take-off mass	$0.7 - 2.0$	$[kg]$
Flight velocity	$10 - 25$	$[m/s]$
I_{xx}	0.02471284	$[kg \cdot m^2]$
I_{yy}	0.015835159	$[kg \cdot m^2]$
I_{zz}	0.037424499	$[kg \cdot m^2]$

Table 2: Stability derivatives for MAKO extracted from AVL program at $14m/s$ equilibrium cruise speed

Parameter	Value	Definition
C_{L_a}	-0.1956×10^{-2}	roll derivative
$C_{L_{\tilde{p}}}$	-4.095×10^{-1}	roll derivative
$C_{L_{\tilde{r}}}$	6.203×10^{-2}	roll derivative
C_{L_β}	3.319×10^{-2}	roll derivative
C_{M_0}	0	pitch derivative
C_{M_e}	-0.076×10^{-1}	pitch derivative
$C_{M_{\tilde{q}}}$	-1.6834	pitch derivative
C_{M_α}	-32.34×10^{-2}	pitch derivative
C_{N_a}	-0.0126×10^{-2}	yaw derivative
$C_{N_{\tilde{p}}}$	-4.139×10^{-2}	yaw derivative
$C_{N_{\tilde{r}}}$	-0.1002×10^{-1}	yaw derivative
C_{N_β}	2.28×10^{-2}	yaw derivative

Table 3: Aerodynamic force derivatives for MAKO extracted from AVL program at $14m/s$ equilibrium cruise speed

Parameter	Value	Definition
C_{Z_0}	-8.53×10^{-2}	lift derivative
C_{Z_α}	3.9444	lift derivative
C_{Z_q}	4.8198	lift derivative
C_{Z_e}	1.6558×10^{-2}	lift derivative
C_{X_0}	2.313×10^{-2}	drag derivative
C_{X_k}	1.897×10^{-1}	drag derivative
C_{Y_β}	-2.708×10^{-1}	side force derivative
$C_{Y_{\tilde{p}}}$	1.695×10^{-2}	side force derivative
$C_{Y_{\tilde{r}}}$	5.003×10^{-2}	side force derivative
C_{Y_a}	0.0254×10^{-2}	side force derivative

Table 4: Thrust force coefficients for propeller APC SF 9×6 from wind tunnel experiments [17]

Parameter	Value	Definition
$C_{F_{T1}}$	1.342×10^{-1}	thrust derivative
$C_{F_{T2}}$	-1.975×10^{-1}	thrust derivative
$C_{F_{T_{rpm}}}$	7.048×10^{-6}	thrust derivative
D	$0.228\,m$	propeller diameter

Table 5: Specifications of the sensor suit InvenSense MPU-9250 Nine-axis (Gyro + Accelerometer + Compass) MEMS MotionTracking Device[18]

Measurement	$\boldsymbol{\beta}$	$\boldsymbol{\sigma}$
$\boldsymbol{z}_{acc_x}$	0.142	0.0319
$\boldsymbol{z}_{acc_y}$	-0.3	0.0985
$\boldsymbol{z}_{acc_z}$	0.19	0.049
$\boldsymbol{z}_{gyro_x}$	-1.55	0.0825
$\boldsymbol{z}_{gyro_y}$	-1.13	0.1673
$\boldsymbol{z}_{gyro_z}$	-1.7	0.2214

APPENDIX B: FORCE, MOMENT CALCULATIONS

Roll torque

$$L_B = \bar{q}\, S\, b\, C_L \tag{11}$$

$$C_L = C_{L_a}\, \delta_a + C_{L_{\tilde{p}}}\, \tilde{p} + C_{L_{\tilde{r}}}\, \tilde{r} + C_{L_\beta}\, \beta \tag{12}$$

Pitch torque

$$M_B = \bar{q}\, S\, \bar{c}\, C_M \tag{13}$$

$$C_M = C_{M_e}\, \delta_e + C_{M_{\tilde{q}}}\, \tilde{q} + C_{M_\alpha}\, \alpha \tag{14}$$

Yaw torque

$$N_B = \bar{q}\, S\, b\, C_N \tag{15}$$

$$C_N = C_{N_a}\, \delta_a + C_{N_{\tilde{p}}}\, \tilde{p} + C_{N_{\tilde{r}}}\, \tilde{r} + C_{N_\beta}\, \beta \tag{16}$$

$$\bar{q} = \frac{\rho V_T^2}{2} \tag{17}$$

Lift force

$$Z^w = \bar{q}\, S\, C_Z(\alpha) \tag{18}$$

$$C_Z(\alpha) = C_{Z_0} + C_{Z_\alpha}\alpha \tag{19}$$

Drag force

$$X^w = \bar{q}\, S\, C_Z(\alpha, \beta) \tag{20}$$

$$C_X(\alpha) = C_{X_1} + C_{X_k}\, C_Z^2 = C_{X_1} + C_{X_k}\, (C_{Z_1} + C_{Z_\alpha}\alpha)^2 \tag{21}$$

Lateral force

$$Y^w = \bar{q}\, S\, C_Y(\beta) \tag{22}$$

$$C_Y(\beta,\, \tilde{p},\, \tilde{r},\, \delta_a) = C_{Y_\beta}\beta + C_{Y_{\tilde{p}}}\, \tilde{p} + C_{Y_{\tilde{r}}}\, \tilde{r} + C_{Y_a}\, \delta_a \tag{23}$$

Thrust force model

$$F_T = \rho n^2 D^4 C_{F_T} \tag{24}$$

$$C_{F_T} = C_{F_{T1}} + C_{F_{T2}} J + C_{F_{T3}} J^2 \tag{25}$$

$$J = \frac{V_T}{n\pi D} \tag{26}$$

Dr. Elgiz Baskaya

CONTACT INFORMATION

7, avenue Edouard Belin CS 54005
31055 Toulouse Cedex 4 France

Mobile: (+33)616460022
E-mail: benelgiz@gmail.com

RESEARCH INTERESTS

Fault Detection and Diagnosis, Machine Learning, Fault Tolerant Control, Estimation and Control

EDUCATION

PhD., Ecole Doctoral Aeronautique Astronautique 2016 - 2019 February (Expected)

University of Paul Sebatier, Toulouse, France

- Dissertation Topic: "Fault Detection & Diagnosis for Drones using Machine Learning"

PhD., Interdisciplinary Aeronautical and Astronautical Sciences 2010 - 2018
Faculty of Aeronautics and Astronautics

Istanbul Technical University, Istanbul, Turkey

- Dissertation Topic: "Heat Transfer Enhancement by Using Magnetic Nanofluids"

M.Sc., Interdisciplinary Aeronautical and Astronautical Sciences 2007 - 2010
Faculty of Aeronautics and Astronautics

Istanbul Technical University, Istanbul, Turkey

- Thesis: "Sensor Fusion Based Attitude Determination for Small Satellites"

B.S., Astronautics Engineering 2002 - 2007
Faculty of Aeronautics and Astronautics

Istanbul Technical University, Istanbul, Turkey

- Graduation Project: "Transfers Between Satellites and Planets Through Trajectories on Stability Boundary"

ACADEMIC EXPERIENCE

Researcher - The ENGIE Ineo - Grope ADP - SAFRAN RPAS Chair 2015 - Present
Ecole National l'Aviation Civile (Civil Aviation Institute of France), Toulouse, France
Designing fault diagnosis algorithms using machine learning techniques to enable a safe integration of drones into airspace. Algorithms designed will be a part of Paparazzi open-source autopilot system and be tested in real flights held under lab URI-DRONES of civil aviation institute of France, ENAC.

- Encourage innovation
- Strive to improve decisiveness

Research Assistant - Faculty of Aeronautics & Astronautics 2009 - 2015
Istanbul Technical University, Istanbul, Turkey

Researcher - TUBITAK (The Scientific and Technological Research Council of Turkey) project 2014 - 2015
Development of methods and algorithms for the integrated attitude determination systems with in-flight sensor calibration, long-term planning guidance and fault-tolerant attitude control for the small information satellites,

Worked on robust attitude determination algorithms, Earth's Magnetic Field modeling, orbit propagation. This was a joint project with Samara State Technical University and was funded by TUBITAK (The Scientific and Technological Research Council of Turkey) and Russian Science Foundation.

- Improve the effectiveness of communications and interactions with others

Researcher - Control and Avionics Laboratory **2010 - 2013**

High precision ADCS and indiginous bus architecture design and development project for Nano Satellites,

Developed the onboard orbit propagator and attitude determination software in MATLAB for ITUpSAT 2, which is the 2^{nd} small satellite project of ITU funded by TUBITAK. Additionally, worked on the design and development of the software/hardware-in-the loop system utilizing MATLAB/Simulink and STKConnect Module to verify the indigenous ADCS which enjoys a redundant reaction wheel set for reliability.

- Constantly sharpen and update skills

Resillience2050, EU 7th RTD Framework Programme

Worked on Decision Support Tools for Air Traffic Management to enable designs fostering safety, agility and resillience for Air Traffic Management.

Real time ATC Operator and Pilot Automation and Decision - Support Systems Design for New ATM Concept,

Worked on Decision Support Tools for Air Traffic Management. Specialized on development of algorithms to estimate trajectory of aircraft and detect collision in the presence of uncertainties. The project is funded by TUBITAK.

Researcher - Space Systems Laboratory **2008 - 2010**

Pico Satellite Design, Development of Engineering and Flight Models,

System Engineer during the design, assembly and test phases of ITUpSAT 1, which was launched on 23 September 2009 from Satish Dhawan Space Center India. ITUpSAT 1, funded by TUBITAK, is the first orbiting student satellite designed and manufactured in Turkey.

Researcher - Vestel Defense Industry **2008**

Studied on verification of the estimated model parameters with the use of measurements via least squares method.

Researcher - Vestel Defense Industry **2007**

Developed codes on estimation of aircraft model using Extended Kalman Filter.

Teaching Experience

Teaching Assistant - Faculty of Aeronautics & Astronautics **2009 - 2015**

Istanbul Technical University, Istanbul, Turkey

Probability & Statistics	2015, 2014
Orbital Mechanics	2014
Heat Transfer	2013
Thermodynamics	2013
Attitude Determination & Control	2014, 2013, 2012, 2011
Avionics	2012, 2011
Principles of Aircraft Design	2012, 2011
Flight Stability and Control,	2011
Automatic Control	2011, 2009
Introduction to Sci&Eng. Comp. - C Programming Language	2011
Dynamics	2009
Spacecraft System Design	2009

Presented "*Developments in Satellite Technology in Turkey*" during GAP Astronomy Journey to Southeastern Anatolia to educate and inform students on recent subjects. Funded by Republic of

Turkey, Regional Development Administration,
Southeastern Anatolia 2010

Course given to Okyanus College students on "*Space, Science and Technology Demonstrations*"
Okyanus College, Istanbul 2009, 2010

Internship

Baykar Technologies, Istanbul, Turkey
Intern **2006**
Summer Internship in one of the two unmanned air vehicle design companies in Turkey. Trained on Kalman Filtering.

TAI (Turkish Aerospace Industries, Inc.), Ankara, Turkey
Intern **2005**
Summer Internship in R&D Department. Trained on satellite thermal control system. Developed codes on thermal network method in Matlab to verify the in-house developed satellite thermal system simulator.

Publications

E. Baskaya, M. Bronz, D. Delahaye, "Fault diagnosis for UAVs using flight data via machine learning," (in preperation)

T. Cunis, E. Baskaya, "Controllability of Unmanned Aircrafts in the Event of Loss-of-control," 10^{th} *International Micro Air Vehicles (IMAV),* (accepted)

G. Manfredi, E. Baskaya, J. Sharpes, Y. Jestin, "Unmanned Aerial System Operations for Retail, " 14^{th} *International Conference on Autonomic and Autonomous Systems (ICAS),* May 20 - 24, 2018

E. Baskaya, M. Bronz, D. Delahaye, "Fault Detection & Diagnosis for Small UAVs via Machine Learning " 14^{th} *IEEE/AIAA Digital Avionics Systems Conference (DASC),* September 17 - 21, 2017

E. Baskaya, M. Bronz, D. Delahaye, "Flight Simulation of a MAKO UAV for Use in Data-Driven Fault Diagnosis " 9^{th} *International Micro Air Vehicles,* September 18 - 21, 2017

E. Baskaya, G. Manfredi, M. Bronz, D. Delahaye "Flexible open architecture for UASs integration into the airspace: Paparazzi autopilot system" 35^{th} *IEEE/AIAA Digital Avionics Systems Conference (DASC),* September 25 - 29, 2016

E. Baskaya, G. Komurgoz, I. Ozkol, "Investigation of Oriented Magnetic Field Effects on Entropy Generation in an Inclined Channel Filled with Ferrofluids, *Entropy,* 19(7), 377, 2017

E. Baskaya, G. Komurgoz, I. Ozkol, "Entropy Generation and Equipartition Phenomenon Investigation in a Variable Viscosity Channel Flow under Constant Magnetic Field via Generalized Differential Quadrature Method (GDQM) , " *Heat Transfer Research* (accepted)

E. Baskaya, G. Komurgoz, I. Ozkol, "Analysis of Variable Viscosity Channel Flow under Constant Magnetic Field via Generalized Differential Quadrature Method, " *Advanced Materials Research,* vol. 1016, pp.564-568, 2014

E. Baskaya, U. Daybelge, A. Sofyali, E. Topal, C. Yarim, "Developments in Astrodynamics in the Light of Chaos " *Journal of Istanbul Kultur University,* vol.4, pp.191-212, 2006

E. Baskaya, G. Komurgoz, I. Ozkol "Analysis of a Variable Viscosity Channel Flow under Constant Magnetic Field via Generalized Differential Quadrature Method" *ICMAE* 5^{th} *International Conference on Mechanical and Aerospace Engineering*, July 18 - 19, 2014

E. Baskaya, M. Fidanoglu, et at. "Investigation of MHD Natural Convection Flow Exposed to a Variable Magnetic Field via Differential Quadrature Method" *ASME* 12^{th} *Biennial Conference on Engineering Systems Design and Analysis*, June 24 - 27, 2014

M. Fidanoglu, E. Baskaya, et at. "Application of Differential Quadrature Method and Evolutionary Algorithm to MHD Fully Developed Flow of a Couple-Stress Fluid in a Vertical Channel With Viscous Dissipation and Oscillating Wall Temperature" *ASME* 12^{th} *Biennial Conference on Engineering Systems Design and Analysis*, June 24 - 27, 2014

E. Baskaya, G. Inalhan, et al. "Design and Development of a Reliable ADCS and Indigenous Bus Architecture for Nanosatellites : ITUpSAT II" 63^{rd} *International Astronautical Congress*, October 1- 5, 2012

E. Baskaya, U. Eren, G. Inalhan "Development of High - Precision Attitude Determination and Control System : ITUpSAT II project " *National Aeronautical and Astronautical Conference*, September 12 - 14, 2012

E. Baskaya, E. Koyuncu, G. Inalhan "Design of a Multi - Purpose Nanosatellite Bus : ITUpSAT II project " *National Aeronautical and Astronautical Conference*, September 12 - 14, 2012

E. Baskaya, U. Eren, et al. "A Precise ADCS Design for ITUpSAT II" *International Conference on Student Small Satellites*, April 25 - 27, 2012

U. Eren, E. Baskaya, et al. "Design of a Flexible Bus System for ITUpSAT II" *International Conference on Student Small Satellites*, April 25 - 27, 2012

E. Koyuncu, E. Baskaya, et al. "ITUpSAT II : High - Precision Nanosatellite ADCS Development Project" 5^{th} *International Conference on Recent Advances in Space Technologies*, June 9 - 11, 2011

G. Inalhan, E. Koyuncu, E. Baskaya, et al. "Design and Development of ITUpSAT II : On Orbit Demonstration of a High - Precision ADCS for Nanosatellites " 8^{th} *International ESA Conference on Guidance & Navigation Control Systems*, June 5 - 10, 2011

E. Baskaya, C. Hajiyev, G. Inalhan "Estimation of Small Satellite Attitude Dynamics via EKF using Magnetometers - ITUpSAT II Project" *National Aeronautical and Astronautical Conference*, September 16 - 18, 2010

Workshops

E. Baskaya, "ITUpSAT II ADCS : Getting Ready for Launch" 8^{th} *Annual CubeSat Developers' Workshop*, April 20 - 22, 2011

E. Baskaya, G. Inalhan "ITUpSAT II - Nanosatellite Platform for In-Space R&D" 7^{th} *Annual CubeSat Developers' Workshop*, April 21 - 23, 2010

Poster Presentations

U. Eren, Elgiz Baskaya, et al. "Design of a Flexible Nanosatellite Bus for Science Missions " *AIAA SPACE 2012 Conference & Exposition*, September 11- 13, 2012

E. Baskaya, U.Eren, et al. "ITUpSAT II - Design and Development " *Innovation Week Turkey*, December 6- 8, 2012

Workshops, Meetings & Lectures Attended

Tutorial on Approaches to Software Design Assurance for Avionics and Flight Controls: DO-178C and Beyond (Part 1 & Part 2), by Tom Ferrell, 2018

Tutorial on Reliable Navigation for Unmanned Aircraft Systems (UAS), by Maarten Uijt de Haag,

2018

Object Oriented Programming with C++, 160 hours course on Objected Oriented Programming with C++ in C System Programmers Association, 2014 - 2015

C Programming Language Training and Certificate, 160 hours course on C Programming language in C System Programmers Association, 2013 - 2014

Practical Adaptive Control, International Graduate School on Control, by Prof Anuradha Annaswamy, MIT Active Adaptive Control Laboratory, May 15 - 19, 2017

Tutorial on Multi-Sensor Navigation Focus on UAV Navigation, by Assist. Prof. Demoz Gebre-Egziabher, University of Minnesota, November 14, 2016

Resillience 2050 Progress Meeting, November 15 - 16, 2012

Research in Decision Support Tools for Future Air Traffic Management, *HALA! SEZAR Research Network Summer School*, July 9 - 12 2012

Lecture on Space System Engineering & CUBESATs by Prof. Dr. Eberhard GILL, May 24, 2012

Small Satellite Formations for Distributed Survelliance: System Design and Optimal Control Considerations, *NATO-RTO Lecture Series SCI-231*, April 14 - 15 2011

Multifunctional Structures and System Technologies for Small Spacecraft, *NATO RTO-AVT171*, April 12 - 15, 2010

Astronet Workshop, *The Astrodynamics Network*, March 30 - April 1, 2009

Skills

- Languages: Turkish (native), English (fluent)(Toefl IBT : 97), French(beginner)
- Programming Languages : MATLAB, C, C++, Fortran
- Simulation/ CAD Tools: MATLAB/Simulink, STK, STKConnect, Mathematica, R, CATIA
- OS: MacOS, Linux, Windows

Professional Activities

- President of Aerospace Engineering Research Assistants (2010 - 2011)
- Member of UUMK (Aeronautics and Astronautics Engineering Club) (2003 - 2007)
- Member of TEGV (Turkish Education Volunteers) (2002 - 2004)

Recreational / Personal Activities

- Pianist and Tango dancer in Galata Festival in 2007, still practicing both as a hobby
- Pianist in concert as PERA Fine Arts High-school students in 2000 in AKM (Ataturk Cultural Center) opera house
- Practicing Shotokai Karate once per week (orange belt) since 2016
- Dedicated Baroque music fan, especially by Bach and Glenn Gould
- Licensed volleyball player in 1999 and 2000 and won two cups as the best team in the town
- Member of the best team in group discussion challenge throughout town in 2002