Fault Tolerant Flight Control Applied to RPAS / UAS

by

Elgiz Baskaya

Submitted to the Applied Mathematics
in partial fulfillment of the requirements for the degree of

Doctor of Philisophy in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1990

Author .....
Applied Mathematics
May 18, 1990

Certified by .....
Daniel Delahaye
Professor
Thesis Supervisor

Certified by .....
Murat Bronz
Assistant Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Fault Tolerant Flight Control Applied to RPAS / UAS

by

Elgiz Baskaya

Submitted to the Applied Mathematics
on May 18, 1990, in partial fulfillment of the
requirements for the degree of
Doctor of Philisophy in Aeronautics and Astronautics

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.

Thesis Supervisor: Daniel Delahaye
Title: Professor

Thesis Supervisor: Murat Bronz
Title: Assistant Professor

Acknowledgments

This work is supported by ENGIE Ineo - Groupe ADP - Safran RPAS Chair.

Contents

List of Figures

List of Tables

Chapter 1

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. With the advent of the new era of UAS, different institutions all over the world, specifically National Aeronautics and Space Administration (NASA) [27] and Federal Aviation Administration (FAA) [29] in US, European Aviation Safety Agency (EASA) [16] in Europe and international bases such as International Civil Aviation Organization (ICAO) [25] are addressing safe integration of UAS in airspace [5].

1.1 Motivation

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 [37]. 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.

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 [37].

With the last turn in popularity and practicality contest among research topics of the last decade, machine learning and drones seem to be neck to neck. The sides of the race are mostly supported by high tech rather than the public who has concerns in both opponents. Is there a chance that these two can run side by side with cheers? Machine learning guides various aspects of our lives even without noticing it due to its abrupt introduction via the bigger tech companies. Its abilities rise, defeating 9-dan Go professional, their accuracy increase, enabling smooth voice recognition, adding intelligence to our daily lives. Another machine, who wants to enjoy this enabling technology is a drone. Drones, although still very strictly regulated in most countries have been spreading with great passion along their enthusiasts. Machine learning has already started to take part in aviation.

Systems are often susceptible to faults of different nature. Existing irregularities in sensors, actuators, or controller Fig. 1-1 could be amplified due to the control system design and lead to failures. A fault could be hidden thanks to the control action [15].

1.2 Terminology

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

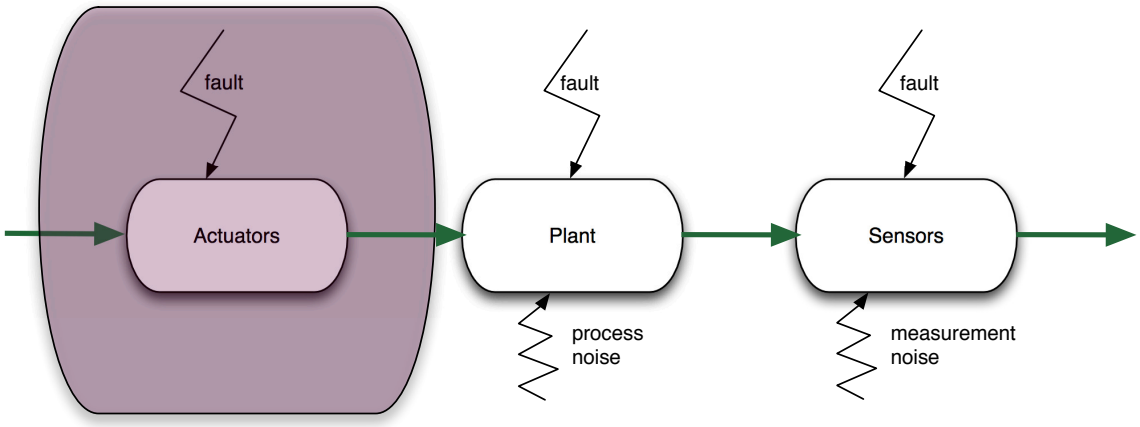

Figure 1-1: Faults altering the system

and Diagnosis, meaning that identification is added to Fault Detection and Isolation [46].

One of the first attempts to unify the terminology is carried out by IFAC SAFEPROCESS technical committee in 1996 and published by [26]. 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

1.3 Conventions for a Safe Flight

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 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 [6].

RECONFIGURE project of FP7 [20] 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 [35] 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].

1.4 Methods for Fault Tolerant Control Systems

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 [6].

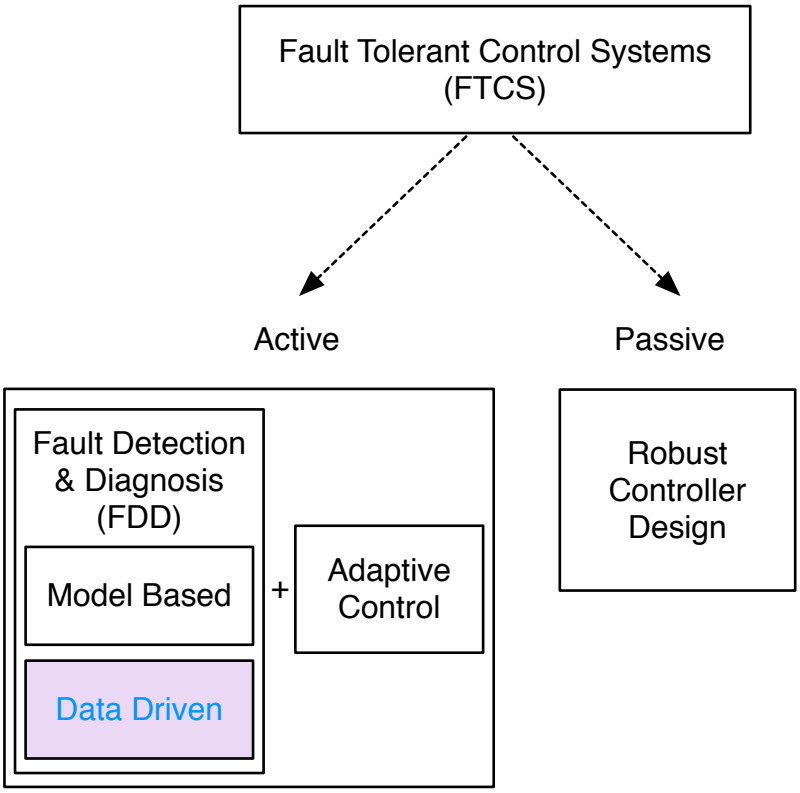

Figure 1-2: Variations of fault tolerant control systems

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. 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. 1-2, 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.

1.4.1 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 [26]. 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.

Model Based Methods

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 resid-

uals 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 [26].

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 andtwo-probe t-test. Fault detection is only supported by simple threshold logic or hypothesis testing in most of the applications [26].

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 [24]. Parameter identification via H_∞ filter is used to indicate icing in [30].

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 [9]. 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 [38]. [39] 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.

Data-Driven Methods

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 [41].

[21] argues artificial intelligent methods for fault detection of complex systems. Comparison between PCA and model based stochastic parity space approaches is given in [23]. In [28], 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.

Chapter 2

Safe Integration of Drones into Airpace

Air safety authorities are forced to develop regulations for UAS due to incidents disturbing public safety and demands from companies who desire to utilize them. There has been numerous studies, from both the FAA in US and EASA in Europe, but none of them decided on a regulations set for the UAVs to satisfy. Improvement of the reliability of the flight is considered to be one of the main obstacles for 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 and failures to emerge. To tackle the safety challenges and help the regulation development, NASA is currently carrying out a four years research program (up to 2019) to enable Unmanned aircraft traffic management solutions which are structured yet flexible when needed. To ensure safety, this integration needs to be achieved through airspace management and UAS reliability.

The preliminary airspace designs, like the one proposed by Amazon, identify different zones depending on the UAS capabilities, population density and altitude. Plus, different national rules and their progressive refinement pushes to cope with a variety of requirements. Open source and modular architectures are key to adapt these requirements. As a specific example, NASA's UTM builds, later to be refined by FAA, make modularity essential for UAS software to follow their evolution.

Concerning reliability, current regulations focus on flight constraints but they 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. This could put too many constraints on UAS manufacturers who desire to access the G airspace.

In Europe, Regulation (EC) No 216/2008 mandates the European Aviation Safety Agency to regulate Unmanned Aircraft Systems (UAS) and in particular Remotely Piloted Aircraft Systems (RPAS), when used for civil applications and with an operating mass of 150 Kg or more.

2.1 UAVs populating airspace

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 [14, 19]. 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 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.

2.2 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.

2.2.1 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. 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. 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. 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.

2.2.2 National Regulations

Circulation of UAS internationally is somewhat prevented by the Chicago convention unless an agreement holds between Contracting States [16]. 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.

2.2.3 Accommodating evolution of regulations

Prescriptive rules seem to cause some difficulties since the technical area on UAS systems develop rapidly [16]. 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.

2.3 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" [34]. 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.

2.3.1 4-D Trajectory Management

As noted in [17], 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 [42]. In clear, it implies that UTM detects future conflicts along the trajectories of all UAS and deconflicting them as safely and early as possible.

2.3.2 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.

2.3.3 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. [18] proposed a participative dynamical airspace management method: the air-parcel 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.

2.3.4 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 [43] 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.

2.4 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 [37]. 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 sys-

tems 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.

2.4.1 SMEs 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 [36]. 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.

2.4.2 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.

2.4.3 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.

2.4.4 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.

To conclude, 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.

2.5 Certification of analytical approaches NOT FINISHED

Due to the lack of the modeling for interaction in-between different FPE (Flight Parameter Estimation), FDD and FTC modules in simulations, leaves them free from practical limitations, offering a long way to go for these analytical methods to be certified.

[REF : certificationOfFDD] [REF : clearanceOptimBased]

safety issues

http://www.techrepublic.com/article/12-drone-disasters-that-show-why-the-faa-hates-drones/

Regulations EU : https://easa.europa.eu/unmanned-aircraft-systems-uas-and-remotely-piloted-aircraft-systems-rpas

The increased cost will be inevitable for the demands of certification. REF : UAVreliabilityStudy

This increment could be even beyond expectations that it could lead to a cancellation of project: REF : http://www.defenseindustrydaily.com/euro-hawk-program-cleared-for-takeoff-03051/

Regulations US :

US Delivery Drones (News from Guardian http://www.theguardian.com/technology/2015/nov/06/a and-facebook-compete-with-secret-drone-plans) ———— Even if the companies solve the technical challenges of keeping drones aloft for long periods, sharing data via lasers and serving city-sized areas, both Alphabet and Facebook still face regulatory hurdles. Neither company has been granted a waiver to the current FAA blanket ban on the commercial operation of unmanned aircraft.

Alphabet has applied for an exemption, called ?333? after a section of the FAA regulations, for its Project Wing delivery drones, but that is yet to be granted and in any case would only clear operation to a maximum height of 400ft. Google has also been testing its delivery drones in the US under a Certificate of Waiver or Authorization (COA) with Nasa, which permitted flights intended to help Nasa develop an automated air traffic control system for low-flying drones. This would not apply to drones flying far above other manned and unmanned aircraft.

?There?s not a lot to run into between 60,000 and 90,000 feet,? says Cummings. ?But I?m sure regulators would be deeply suspicious if Google and Facebook were flying these over the US.?

Facebook and Alphabet would not comment.

What's really standing in the way of drone delivery? ———————————— ———————— http://www.theverge.com/2016/1/16/10777144/delivery-drones-regulations-

safety-faa-autonomous-flight

SORASORA SORA SORA Fig. 2-1

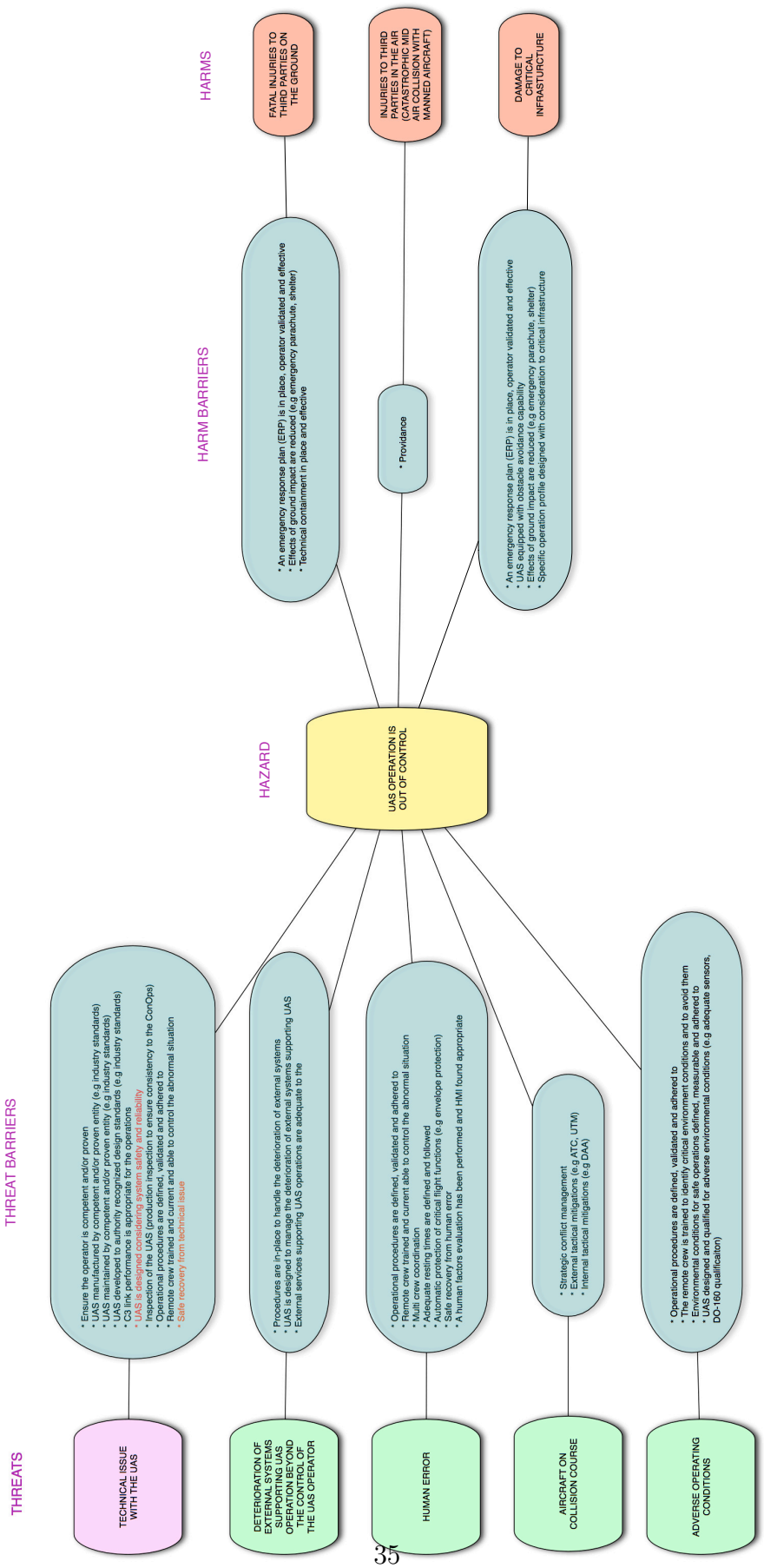

Figure 2-1: SORA structure: threats, threat barriers, hazard, harm barriers, harms

Chapter 3

State of the Art

Chapter 4

Nonlinear Aircraft Model

The motion of the aircraft is conventionally modeled by two different aspects. First the aircraft is assumed to be a point mass and this point's position and mass in space is given in time. These equations are said to represent the translational motion of the aircraft. Secondly, the aircrafts attitude is investigated by no longer assuming it as a point mass but a rigid body in space. The attitude and angular velocities are studied with these equations namely the attitude motion equations. Since some of the states in these equations is dependent on the other states in the other set, these two motions are coupled and should be solved simultaneously.

4.1 Attitude motion modeling

Attitude motion is represented with kinematic and dynamics equations. First different parametrizations, the 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.

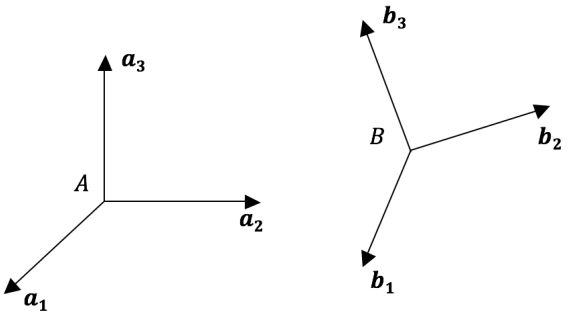

Figure 4-1: Attitude representation is simply specifying the orientation of aircraft body axes b_1, b_2, b_3 in the reference frame A

4.1.1 Attitude representations

Let us say triad b_1, b_2, b_3 of unit vectors representing an orthogonal coordinate system is attached to a rigid body, such that

$$\boldsymbol{b_1} \times \boldsymbol{b_2} = \boldsymbol{b_3} \tag{4.1}$$

The rigid body is composed of points, which do not experience any distance change in between during motion of the body. Problem of representing attitude can simply be thought of specifying the orientation of this triad with respect to some reference frame. See Fig. 4-1.

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 such that

$$\begin{aligned} \boldsymbol{b_1} &= C_{11}\boldsymbol{a_1} + C_{12}\boldsymbol{a_2} + C_{13}\boldsymbol{a_3}, \\ \boldsymbol{b_2} &= C_{21}\boldsymbol{a_1} + C_{22}\boldsymbol{a_2} + C_{23}\boldsymbol{a_3}, \\ \boldsymbol{b_3} &= C_{33}\boldsymbol{a_1} + C_{32}\boldsymbol{a_2} + C_{33}\boldsymbol{a_3}. \end{aligned} \tag{4.2}$$

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 written in matrix form,

$$\begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{11} & C_{12} & C_{13} \\ C_{11} & C_{12} & C_{13} \end{bmatrix} \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix} = \boldsymbol{C}^{B/A} \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix} \tag{4.3}$$

Here is called the direction cosine matrix, also known as rotation matrix or coordinate transformation matrix to B from A [44]. The direction cosines, the elements of the direction cosine matrix, are not all independent [44].

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

$$\boldsymbol{C}^{-1} = \boldsymbol{C}^{\mathrm{T}} \tag{4.4}$$

and

$$\boldsymbol{C}\boldsymbol{C}^{\mathrm{T}} = \boldsymbol{C}^{\mathrm{T}}\boldsymbol{C} = \mathbb{1} \tag{4.5}$$

When the orientation is preserved, an additional condition occurs

$$\Big|C\Big| = \mathbb{1} \tag{4.6}$$

Matrices satisfying the last two properties belong to special orthogonal group and denoted by 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 to B from A and $\boldsymbol{C}^{A/B}$, the direction cosine matrix of A relative to B, or the direction cosine matrix to A from B.

$$\begin{aligned} [\boldsymbol{C}^{A/B}]^{-1} &= [\boldsymbol{C}^{A/B}]^{\mathrm{T}} = \boldsymbol{C}^{B/A} \\ [\boldsymbol{C}^{B/A}]^{-1} &= [\boldsymbol{C}^{B/A}]^{\mathrm{T}} = \boldsymbol{C}^{A/B} \end{aligned} \tag{4.7}$$

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:

$$\begin{aligned} \boldsymbol{H} &= H_1\boldsymbol{a_1} + H_2\boldsymbol{a_2} + H_3\boldsymbol{a_3} \\ &= H_1^{'}\boldsymbol{b_1} + H_2^{'}\boldsymbol{b_2} + H_3^{'}\boldsymbol{b_3} \end{aligned} \tag{4.8}$$

Utilization of the direction cosine matrix in the following equation, components of the arbitrary vector $\boldsymbol{H}$ is transformed to B from A.

$$\begin{bmatrix} H_1^{'} \\ H_2^{'} \\ H_3^{'} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b_1}\cdot\boldsymbol{a_1} & \boldsymbol{b_1}\cdot\boldsymbol{a_2} & \boldsymbol{b_1}\cdot\boldsymbol{a_3} \\ \boldsymbol{b_2}\cdot\boldsymbol{a_1} & \boldsymbol{b_2}\cdot\boldsymbol{a_2} & \boldsymbol{b_2}\cdot\boldsymbol{a_3} \\ \boldsymbol{b_3}\cdot\boldsymbol{a_1} & \boldsymbol{b_3}\cdot\boldsymbol{a_2} & \boldsymbol{b_3}\cdot\boldsymbol{a_3} \end{bmatrix} \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} = \boldsymbol{C}^{B/A} \begin{bmatrix} H_1 \\ H_2 \\ H_3 \end{bmatrix} \tag{4.9}$$

Euler Angles

One of the approaches to describe the attitude is using Euler angles. It is a procedure of rotating three times successively about one of the axis of the rotated body fixed reference frame. First rotation is about any of the fixed body axes. The second rotation is about any of the other two axis which have not been used in the first rotation. Third rotation is about one of the axis which have 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 A frame describing the orientation of the body frame B with respect to reference frame A can be represented as

$$\begin{aligned} &\boldsymbol{C}_3(\theta_3) : A^{'} \leftarrow A, \\ &\boldsymbol{C}_2(\theta_2) : A^{''} \leftarrow A^{'}, \\ &\boldsymbol{C}_1(\theta_1) : A^{'} \leftarrow A^{''}. \end{aligned} \tag{4.10}$$

The direction cosine matrix relative to can be given as;

$$\boldsymbol{C}^{B/A} = \boldsymbol{C}_1(\theta_1)\boldsymbol{C}_2(\theta_2)\boldsymbol{C}_3(\theta_3) \tag{4.11}$$

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 axis of the body frame. The orientation of with respect to A is given as

$$\begin{aligned}
\begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} &= \boldsymbol{C}^{B/A} \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 \\ 0 & -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & 0 & -\sin\theta_2 \\ 0 & 1 & 0 \\ \sin\theta_2 & 0 & \cos\theta_2 \end{bmatrix} \begin{bmatrix} \cos\theta_3 & \sin\theta_3 & 0 \\ -\sin\theta_3 & \cos\theta_3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta_2\cos\theta_3 & \cos\theta_2\sin\theta_3 & -\sin\theta_2 \\ \sin\theta_1\sin\theta_2\cos\theta_3 - \cos\theta_1\sin\theta_3 & \sin\theta_1\sin\theta_2\cos\theta_3 + \cos\theta_1\cos\theta_1\cos\theta_3 & \sin\theta_1\cos\theta_2 \\ \cos\theta_1\sin\theta_2\cos\theta_3 + \sin\theta_1\sin\theta_3 & \cos\theta_1\sin\theta_2\sin\theta_3 - \sin\theta_1\cos\theta_3 & \cos\theta_1\cos\theta_2 \end{bmatrix}
\end{aligned} \tag{4.12}$$

If the set of rotations is selected in a different way, such as

$$\begin{aligned}
&\boldsymbol{C}_2(\theta_2) : A^{'} \leftarrow A, \\
&\boldsymbol{C}_3(\theta_3) : A^{''} \leftarrow A^{'}, \\
&\boldsymbol{C}_1(\theta_1) : A^{'} \leftarrow A^{''}.
\end{aligned} \tag{4.13}$$

then the direction cosine matrix would differ from the previous one.

Quaternions

The idea behind this representation is the Eulerś 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

$$\boldsymbol{C}\boldsymbol{e} = \boldsymbol{e} \tag{4.14}$$

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

$$\begin{aligned} \boldsymbol{e} &= e_1\boldsymbol{a_1} + e_2\boldsymbol{a_2} + e_3\boldsymbol{a_3} \\ &= e_1\boldsymbol{b_1} + e_2\boldsymbol{b_2} + e_3\boldsymbol{b_3} \end{aligned} \tag{4.15}$$

Rotating a rigid body about this axis , rotation from any given orientation to any other orientation can be achieved. Such a rotation is named after the Swiss mathematician and theoretical physicist Leonard Euler (1707-1783), Euler axis. Euler symmetric parameters (also known as quaternions) are defined as:

$$\boldsymbol{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\phi/2) \\ e_1 \sin(\phi/2) \\ e_2 \sin(\phi/2) \\ e_3 \sin(\phi/2) \end{bmatrix} \tag{4.16}$$

Where ϕ is the Euler rotation angle. Hamilton (1805-1865) is the discoverer of the quaternions. For ease of the mathematical representations, we define the a vector such as

$$\boldsymbol{q} = \boldsymbol{e} \sin \phi/2 \tag{4.17}$$

Euler symmetric parameters are not independent, and satisfy the constraint

$$\boldsymbol{q}^{\mathrm{T}}\boldsymbol{q} + q_0^2 = q_0^2 + q_1^2 + q_1^2 + q_3^2 = 1 \tag{4.18}$$

The direction cosine matrix can also be written in terms of Euler symmetric parameters as below

$$
\begin{aligned}
C(\boldsymbol{q}) &= \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_3q_0) & 2(q_1q_3-q_2q_0) \\ 2(q_1q_2-q_3q_0) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3+q_1q_0) \\ 2(q_1q_3+q_2q_0) & 2(q_2q_3-q_1q_0) & -q_1^2-q_2^2+q_3^2+q_0^2 \end{bmatrix} \\
&= (q_0^2-\boldsymbol{q}^{\mathrm{T}}\boldsymbol{q})\mathbb{1}+2\boldsymbol{q}\boldsymbol{q}^{\mathrm{T}}-2q_4\boldsymbol{Q}
\end{aligned}
\tag{4.19}
$$

where

$$
\boldsymbol{Q} = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix}
\tag{4.20}
$$

Given the direction cosine matrix, Euler symmetric parameters can be calculated as

$$
\begin{aligned}
\boldsymbol{q} &= \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \frac{1}{4q_0}\begin{bmatrix} C_{23}-C_{32} \\ C_{31}-C_{13} \\ C_{12}-C_{21} \end{bmatrix} \\
q_0 &= \pm\frac{1}{2}(1+C_{11}+C_{22}+C_{33})^{\frac{1}{2}}
\end{aligned}
\tag{4.21}
$$

Attitude parametrization selection

In 1776, Euler showed that SO(3) has three dimensions [40]. Representations with more than three parameters subject to constraints. Also, [40] states that no parameter set can be both global and nonsingular. So, we are faced with choosing the representation parameters either singular or redundant. Euler angle representation has its advantages like having a clear physical interpretation and minimum parameter set achieving no redundancy. But due to their important disadvantageous, the possibility of having singularities when describing motion, Euler angles are not selected.

The Gibbs vector is not very often used, and can be thought that it is an interval step on the way of quaternion parametrization. There are other representation types, which are not very often preferred, such as the axis-azimuth representation. So there are not mentioned in this thesis. Quaternion (Euler symmetric parameters) representation takes the lead for this mission because of their usefulness in simulations, no singularities. Also as we will mention in a short time, the kinematic equations are linear in terms of quaternions. Quaternion multiplications offers a useful way of expressing composite rotations. With all these preferable properties, quaternions are the choice for attitude representations for many attitude control missions. 4.1.1 shows a brief comparison of attitude representations [4]

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

In the previous sections, various ways to represent rotations are identified and inherent properties of each technique have been examined. Now it is time to find out how to represent attitude depending on time, namely the equations of motion. Equations of motion is generally presented in two sections: the kinematic equations and dynamic equations. The kinematic equations provide the relations between the time derivative of the attitude representation and the angular velocity, while the dynamics (or kinetics) describe the development of angular velocities under influence of external moments.

4.1.2 Attitude kinematics

The time evalution of attitude parameters is identifed by a set of first order differential equations called the kinematic equations. For each attitude parametrization choice, the corresponding set of kinematic equation differs.

The angular velocity of a reference frame B with respect a reference frame A, given in the B frame can be written as follows

$$\boldsymbol{\omega} = \omega_1 \boldsymbol{b_1} + \omega_2 \boldsymbol{b_2} + \omega_3 \boldsymbol{b_3} \tag{4.22}$$

Recalling Eq. 4.23 and orthonomality property in Eq. 4.4

$$\begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix} = \left[\boldsymbol{C}_B^{B/A}\right]^{-1} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} \tag{4.23}$$

Taking the time derivative with respect to frame A

$$\begin{aligned} \frac{d}{dt} \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \\ \boldsymbol{a_3} \end{bmatrix}\Bigg|_A &= \left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} + \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \dot{\boldsymbol{b_1}} \\ \dot{\boldsymbol{b_2}} \\ \dot{\boldsymbol{b_3}} \end{bmatrix} = \left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} + \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{\omega} \times \boldsymbol{b_1} \\ \boldsymbol{\omega} \times \boldsymbol{b_2} \\ \boldsymbol{\omega} \times \boldsymbol{b_3} \end{bmatrix} \\ &= \left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} + \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}} \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} \end{aligned} \tag{4.24}$$

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

$$\mathbf{\Omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{4.25}$$

and rearrange the terms

$$\left[\left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} - \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}}\mathbf{\Omega}\right] \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \\ \boldsymbol{b_3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.26}$$

Implies

$$\left[\left[\dot{\boldsymbol{C}}_B^{B/A}\right]^{\mathrm{T}} - \left[\boldsymbol{C}_B^{B/A}\right]^{\mathrm{T}}\mathbf{\Omega}\right] = \mathbf{0} \tag{4.27}$$

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

$$\dot{\boldsymbol{C}}_B^{B/A} + \mathbf{\Omega}\boldsymbol{C}_B^{B/A} = \mathbf{0} \tag{4.28}$$

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

$$\begin{aligned} \omega_1 &= \dot{C}_{21}C_{31} + \dot{C}_{22}C_{32} + \dot{C}_{23}C_{33} \\ \omega_2 &= \dot{C}_{31}C_{11} + \dot{C}_{32}C_{12} + \dot{C}_{33}C_{13} \\ \omega_3 &= \dot{C}_{11}C_{21} + \dot{C}_{12}C_{22} + \dot{C}_{13}C_{23} \end{aligned} \tag{4.29}$$

From that point, derivation depends on the attitude parameters used. When the direction cosines and their derivatives are substituted with their equals in terms of Euler angles, the result will give the time dependence of Euler angles. But in this study, due to the reasons discussed before, quaternions are selected as to represent the attitude, so we will keep going with substituting the direction cosines in terms of

quaternions,

$$\begin{aligned}
\omega_1 &= 2(\dot{q}_1 q_0 + \dot{q}_2 q_3 - \dot{q}_3 q_2 - \dot{q}_0 q_1) \\
\omega_2 &= 2(\dot{q}_2 q_0 + \dot{q}_3 q_1 - \dot{q}_1 q_3 - \dot{q}_0 q_2) \\
\omega_3 &= 2(\dot{q}_3 q_0 + \dot{q}_1 q_2 - \dot{q}_2 q_1 - \dot{q}_2 q_3)
\end{aligned} \tag{4.30}$$

Fourth equation comes from the differentiation of the quaternion norm constraint;

$$0 = 2(-\dot{q}_0 q_0 + \dot{q}_1 q_1 + \dot{q}_2 q_2 - \dot{q}_3 q_3) \tag{4.31}$$

Writing in matrix form;

$$\begin{bmatrix} 0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} = 2 \begin{bmatrix} q_1 & q_2 & q_3 & q_0 \\ q_0 & q_3 & -q_2 & -q_1 \\ -q_3 & q_0 & q_1 & -q_2 \\ q_2 & -q_1 & q_0 & -q_3 \end{bmatrix} \begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} \tag{4.32}$$

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 [44]

$$\begin{aligned}
\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} &= \frac{1}{2} \begin{bmatrix} -q_1 & -q_2 & -q_3 & q_0 \\ q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & -q_1 & q_4 & q_3 \end{bmatrix} \begin{bmatrix} 0 \\ \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \\
&= \frac{1}{2} \begin{bmatrix} -\omega_1 & -\omega_2 & -\omega_3 & 0 \\ 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}
\end{aligned} \tag{4.33}$$

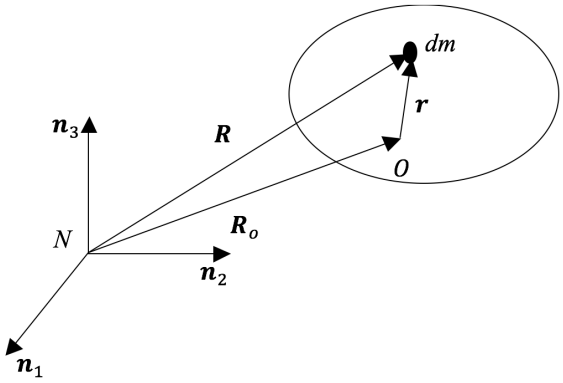

Figure 4-2: Rigid body rotating about an arbitrary point O, given in the N frame

And in a more compact form;

4.1.3 Attitude dynamics

The equation describing the rotational motion of a rigid body moving relative to an inertial frame can written as [44]

$$\int \boldsymbol{r} \times \ddot{\boldsymbol{R}} dm = \boldsymbol{M_0} \tag{4.34}$$

Here the rotation of the rigid body takes place about an arbitrary point O. Let us take an infinitisimal mass element dm. In Fig.4-2, r is the position vector of 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, $\boldsymbol{M_0}$ is the total external torque about point O.

Now, let us take a body fixed frame B originated at the center of mass of the rigid body as shown below. In Fig.4-3, $\boldsymbol{\rho}$ is the position vector of dm mass with respect to 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 $\boldsymbol{\omega} \equiv \boldsymbol{\omega}_B^{B/N}$ inertial frame N is denoted as . 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 $\boldsymbol{H}$ a rigid body about

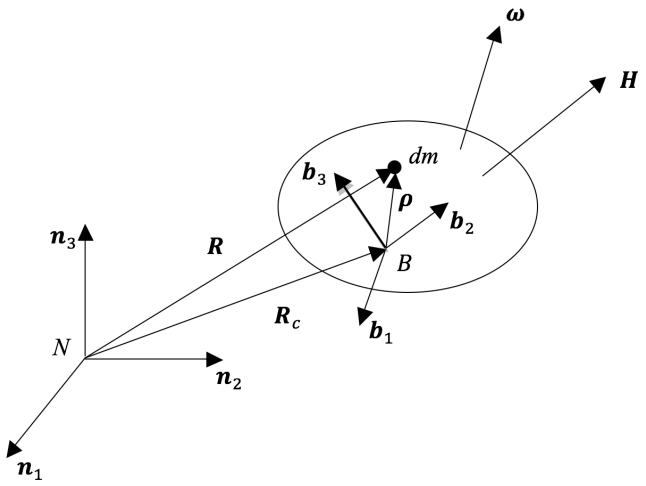

Figure 4-3: Rigid body rotating about an arbitrary point O, given in the N frame

its center of mass given as,

$$\boldsymbol{H} = \int \boldsymbol{\rho} \times \dot{\boldsymbol{R}} dm \tag{4.35}$$

When $\boldsymbol{R}_C$ is constant

$$\dot{\boldsymbol{R}} = \dot{\boldsymbol{R}}_C + \dot{\boldsymbol{\rho}} = \dot{\boldsymbol{\rho}} \tag{4.36}$$

And from rigidity of the body,

$$\dot{\boldsymbol{\rho}} \equiv \left\{ \frac{d\boldsymbol{\rho}}{dt} \right\}_N = \left\{ \frac{d\boldsymbol{\rho}}{dt} \right\}_B + \boldsymbol{\omega} \times \boldsymbol{\rho} = \boldsymbol{\omega} \times \boldsymbol{\rho} \tag{4.37}$$

Then the angular momentum is given as;

$$\boldsymbol{H} = \int \boldsymbol{\rho} \times \dot{\boldsymbol{R}} dm = \int \boldsymbol{\rho} \times \dot{\boldsymbol{\rho}} dm = \int \boldsymbol{\rho} \times (\boldsymbol{\omega} \times \boldsymbol{\rho}) dm \tag{4.38}$$

The components of $\boldsymbol{\rho}$ and $\boldsymbol{\rho}$ in the body frame B is written as;

$$\begin{aligned} \boldsymbol{\rho} &= \rho_1 \boldsymbol{b}_1 + \rho_2 \boldsymbol{b}_2 + \rho_3 \boldsymbol{b}_3 \\ \boldsymbol{\omega} &= \omega_1 \boldsymbol{b}_1 + \omega_2 \boldsymbol{b}_2 + \omega_3 \boldsymbol{b}_3 \end{aligned} \tag{4.39}$$

Then the angular momentum can be written as;

$$\boldsymbol{H} = \boldsymbol{I}\boldsymbol{\omega} \tag{4.40}$$

With components in frame B

$$\boldsymbol{H} = H_1 \boldsymbol{b}_1 + H_2 \boldsymbol{b}_2 + H_3 \boldsymbol{b}_3 \tag{4.41}$$

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

Now, it is time to express the rotational equations of motion ? also known as Eulerś equations of motion ? for a rigid body. Remember the angular momentum equation

$$\boldsymbol{M} = \dot{\boldsymbol{H}} \tag{4.42}$$

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;

$$\dot{\boldsymbol{H}} = \left\{ \frac{d\boldsymbol{H}}{dt} \right\}_N = \left\{ \frac{d\boldsymbol{H}}{dt} \right\}_B + \boldsymbol{\omega}_B^{B/N} \times \boldsymbol{H} = \boldsymbol{M} \tag{4.43}$$

By taking the time derivative of Eq. 4.40 and assuming the inertia is not dependent on time, attitude dynamics can be written as

$$\dot{\boldsymbol{\omega}}_B^{B/N} = \boldsymbol{I}_B^{-1} (\boldsymbol{M}_B - \boldsymbol{\omega}_B^{B/N} \times \boldsymbol{I}_B \boldsymbol{\omega}_B^{B/N}) \tag{4.44}$$

Figure 4-4: MAKO

4.1.4 UAVs simulated

In this section parameters for two different drone is given, ETH model aircraft and MAKO model.

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 4-5, and measurements performed by timing the oscillation period for each axis. The resultant moment of inertias are given Table 4.2.2.

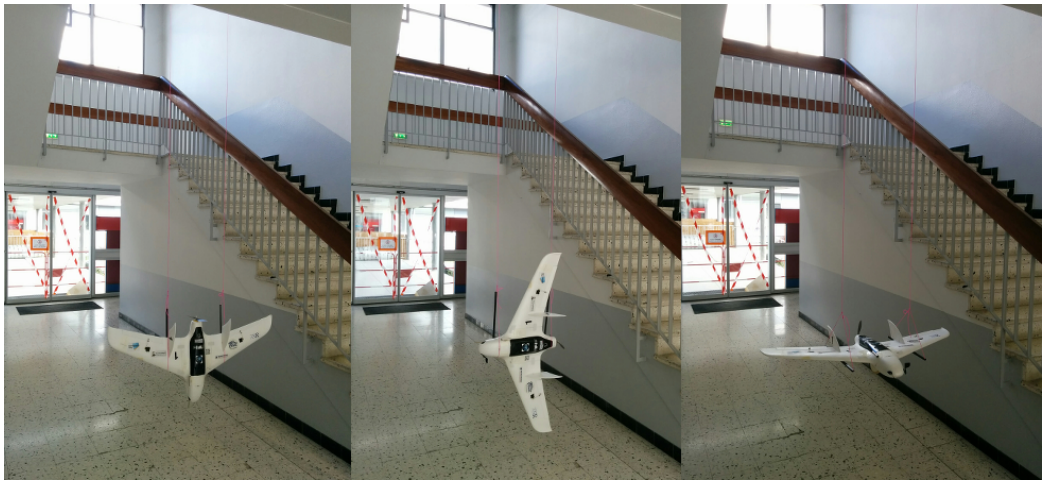

Figure 4-5: Moments of inertia measurements for each axis, I_{xx}, I_{yy}, I_{zz}.

Table 4.1: General specifications of MAKO [8]

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	$[kg \cdot m^2]$

4.1.5 Attitude kinematics of aircraft

Since the general equations of attitude motion of a rigid body, by introducing the frames of interest and angles standardized for aircraft attitude motion can be investigated next.

Frames of interest ECI ECEF Navigation Frame BODY

As mentioned before, the sequence of the transformation between frames effects the final transformation matrix so does the attitude kinematics equation. In the course of the study, the sequence is assumed to be yaw - pitch - roll as is given in Fig. 4-10 and the corresponding transformation matrix from the navigation frame to

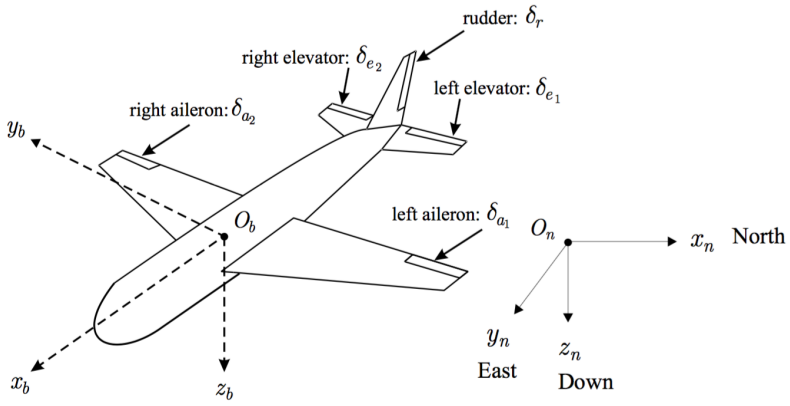

Figure 4-6: Body fixed frame and North East Down (NED) frame representations [15]

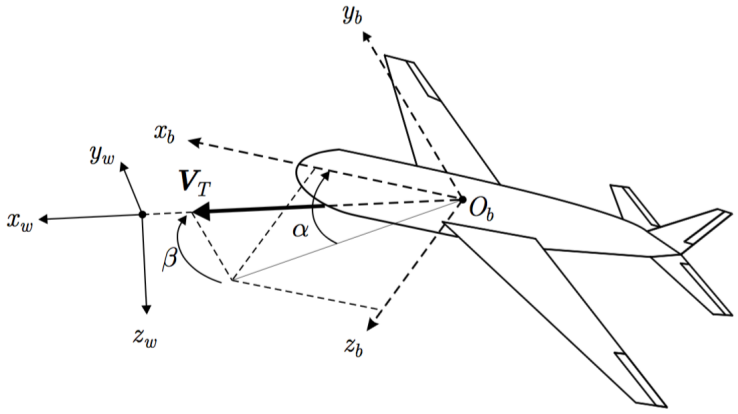

Figure 4-7: Wind frame, airspeed vector $\boldsymbol{V}_T$, angle of attack α and side slip angle β representation [15]

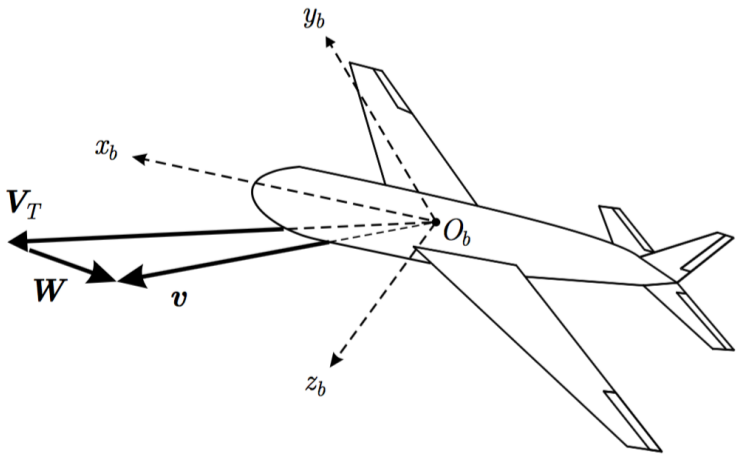

Figure 4-8: Relation revealed between the inertial velocity vector $\boldsymbol{v}$, airspeed vector $\boldsymbol{V}_T$ and wind disturbance $\boldsymbol{W}$ [15]

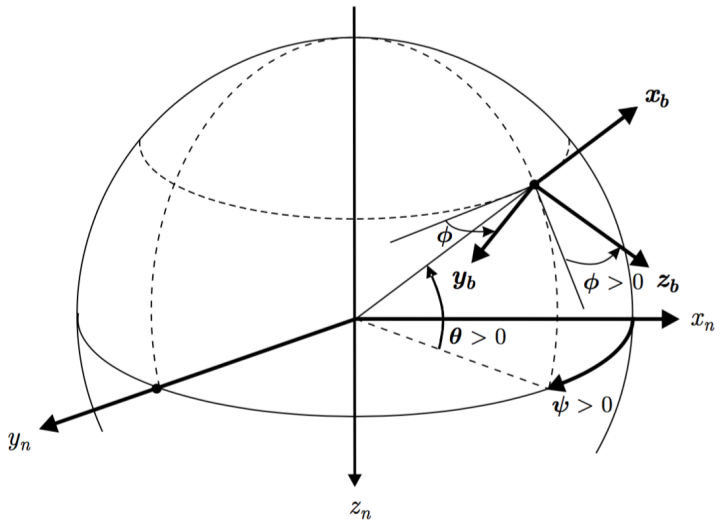

Figure 4-9: Euler angle sequence [15]

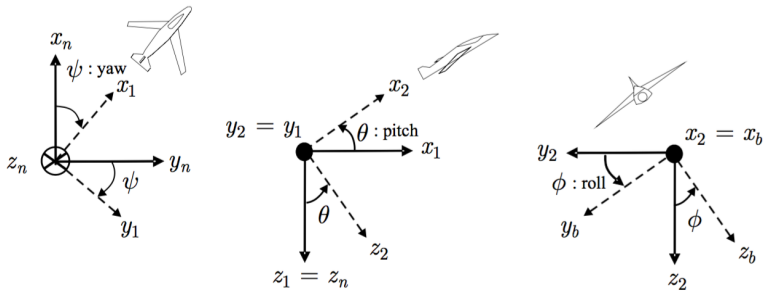

Figure 4-10: Euler angle sequence [15]

body-fixed frame is given in Eq. 4.45.

$$C_N^B = \begin{bmatrix} 1-2(q_2^2+q_3^2) & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 1-2(q_1^2+q_3^2) & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & 1-2(q_1^2+q_2^2) \end{bmatrix} \tag{4.45}$$

And finally the kinematics equation of motion for the aircraft can be written as in Eq. 4.46

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.46}$$

4.1.6 Attitude dynamics of aircraft

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.

Model of aerodynamic moments

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

$$\boldsymbol{M}_B = \begin{bmatrix} M_{Bx} \\ M_{By} \\ M_{Bz} \end{bmatrix} = \begin{bmatrix} L_B \\ M_B \\ N_B \end{bmatrix} \tag{4.47}$$

The torques can be given under three subclasses; roll torque L_B, pitch torque M_B, and yaw torque N_B.

Roll torque

$$L_B = \bar{q}\, S\, b\, C_L \tag{4.48}$$

$$C_L = C_{L_{a1}}\, \delta_{a1} + C_{L_{a2}}\, \delta_{a2} + C_{L_{e1}}\, \delta_{e1} + C_{L_{e2}}\, \delta_{e2} + C_{L_{\tilde{p}}}\, \tilde{p} + C_{L_{\tilde{r}}}\, \tilde{r} + C_{L_\beta}\, \beta \tag{4.49}$$

$$C_L = C_{L_a}\, \delta_a + C_{L_{\tilde{p}}}\, \tilde{p} + C_{L_{\tilde{r}}}\, \tilde{r} + C_{L_\beta}\, \beta \tag{4.50}$$

Pitch torque

$$M_B = \bar{q}\, S\, \bar{c}\, C_M \tag{4.51}$$

$$C_M = C_{M_1} + C_{M_{a1}}\, \delta_{a1} + C_{M_{a2}}\, \delta_{a2} + C_{M_{e1}}\, \delta_{e1} + C_{M_{e2}}\, \delta_{e2} + C_{M_{\tilde{q}}}\, \tilde{q} + C_{M_\alpha}\, \alpha \tag{4.52}$$

$$C_M = C_{M_e}\, \delta_e + C_{M_{\tilde{q}}}\, \tilde{q} + C_{M_\alpha}\, \alpha \tag{4.53}$$

Table 4.2: Stability derivatives for ETH UAV [15]

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

Yaw torque

$$N_B = \bar{q}\, S\, b\, C_N \tag{4.54}$$

$$C_N = C_{N_{\delta r}}\, \delta_r + C_{N_{\tilde{r}}}\, \tilde{r} + C_{N_{\beta}}\, \beta \tag{4.55}$$

$$C_N = C_{N_a}\, \delta_a + C_{N_{\tilde{p}}}\, \tilde{p} + C_{N_{\tilde{r}}}\, \tilde{r} + C_{N_{\beta}}\, \beta \tag{4.56}$$

$$\bar{q} = \frac{\rho V_T^2}{2} \tag{4.57}$$

where $\bar{q}$ is the dynamic pressure given in Eq. 4.59, 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.

Table 4.3: Stability derivatives for MAKO extracted from AVL program at $14m/s$ equilibrium cruise speed [8]

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

4.2 Translational motion modeling

4.2.1 Translational kinematics

4.2.2 Translational dynamics

Model of aerodynamic forces

Lift force

$$Z^w = \bar{q}\, S\, C_Z(\alpha) \tag{4.58}$$

$$\bar{q} = \frac{\rho V_T^2}{2} \tag{4.59}$$

$$C_Z(\alpha) = C_{Z_0} + C_{Z_\alpha}\alpha \tag{4.60}$$

Drag force

$$X^w = \bar{q}\, S\, C_X(\alpha, \beta) \tag{4.61}$$

Table 4.4: Aerodynamic force derivatives for ETH UAV [15]

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

$$C_X(\alpha, \beta) = C_{X_1} + C_{X_\alpha}\alpha + C_{X_{\alpha 2}}\alpha^2 + C_{X_{\beta 2}}\beta^2 \tag{4.62}$$

$$X^w = \bar{q}\, S\, C_Z(\alpha, \beta) \tag{4.63}$$

$$C_X(\alpha) = C_{X_1} + C_{X_k}\, C_Z^2 = C_{X_1} + C_{X_k}\, (C_{Z_1} + C_{Z_\alpha}\alpha)^2 \tag{4.64}$$

Lateral force

$$Y^w = \bar{q}\, S\, C_Y(\beta) \tag{4.65}$$

$$C_Y(\beta) = C_{Y_\beta}\beta \tag{4.66}$$

$$C_Y(\beta,\, \tilde{p},\, \tilde{r},\, \delta_a) = C_{Y_\beta}\beta + C_{Y_{\tilde{p}}}\, \tilde{p} + C_{Y_{\tilde{r}}}\, \tilde{r} + C_{Y_a}\, \delta_a \tag{4.67}$$

Thrust force model

$$F_T = \rho n^2 D^4 C_{F_T} \tag{4.68}$$

$$C_{F_T} = C_{F_{T1}} + C_{F_{T2}} J + C_{F_{T3}} J^2 \tag{4.69}$$

Table 4.5: Aerodynamic force derivatives for MAKO extracted from AVL program at $14m/s$ equilibrium cruise speed [8]

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 4.6: Thrust force coefficients for propeller ETH UAV [15]

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

$$J = \frac{V_T}{n\pi D} \tag{4.70}$$

$$C_{F_T} = C_{F_{T1}} + C_{F_{T2}} J' + C_{F_{T_{rpm}}} \frac{n}{60} \tag{4.71}$$

$$J' = \frac{V_T}{nD} \tag{4.72}$$

4.3 Shortcut to modeling

Nonlinear aircraft flight dynamics for translational and attitude motion can be given as a system of first order differential equations

Table 4.7: Thrust force coefficients for propeller APC SF 9×6 from wind tunnel experiments [7]

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

$$
\begin{aligned}
\dot{\boldsymbol{x}}_n &= \boldsymbol{C}_b^n \boldsymbol{v}^b \\
\dot{\boldsymbol{v}}^b &= \frac{1}{m}\big[m\boldsymbol{g}^b + \boldsymbol{F}_t^b + \boldsymbol{F}_a^b\big] - \boldsymbol{\omega}_{b/n}^{b\,\times}\boldsymbol{v}^b \\
\dot{q}_0 &= -\frac{1}{2}\boldsymbol{q}_\nu^T \boldsymbol{\omega}_{b/n}^b \\
\dot{\boldsymbol{q}}_\nu &= \frac{1}{2}\Big(\boldsymbol{q}_\nu^\times + q_0 \boldsymbol{I}_3\Big)\boldsymbol{\omega}_{b/n}^b \\
\boldsymbol{J}\dot{\boldsymbol{\omega}}_{b/n}^b &= \boldsymbol{M} - \boldsymbol{\omega}_{b/n}^{b\,\times}\boldsymbol{J}\boldsymbol{\omega}_{b/n}^b
\end{aligned}
\tag{4.73}
$$

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

$$\boldsymbol{x}^{\times} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{4.74}$$

$$\begin{aligned} L &= \bar{q}\, S\, b\, C_L \\ M &= \bar{q}\, S\, \bar{c}\, C_M \\ N &= \bar{q}\, S\, b\, C_N \end{aligned} \tag{4.75}$$

4.4 Verification with Matlab Simulink 6DOF block

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. 4-11. 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.

4.5 Sensor Models

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.

$$\boldsymbol{z}_{gyro} = \boldsymbol{k}_{gyro}\boldsymbol{\omega}^{b}_{b/i} + \boldsymbol{\beta}_{gyro} + \boldsymbol{\eta}_{gyro} \tag{4.76}$$

$$\boldsymbol{z}_{acc} = \boldsymbol{k}_{acc}\boldsymbol{\omega}^{b}_{b/i} + \boldsymbol{\beta}_{acc} + \boldsymbol{\eta}_{acc} \tag{4.77}$$

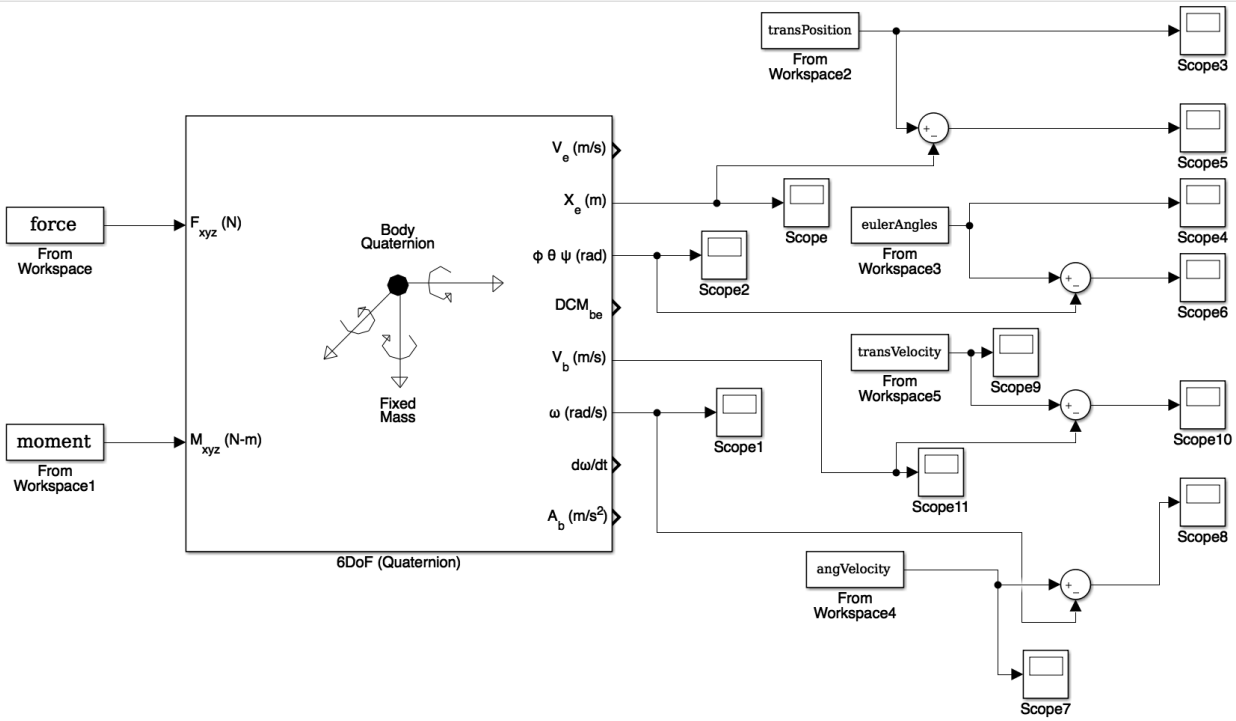

Figure 4-11: Validation with Simulink 6DOF aircraft model

Table 4.8: Specifications of the sensor suit InvenSense MPU-9250 Nine-axis (Gyro + Accelerometer + Compass) MEMS MotionTracking Device[10]

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

Here $\boldsymbol{\beta}$ is the bias, and $\boldsymbol{\eta}$ is the zero mean Gaussian process with $\boldsymbol{\sigma}^2$ variance and given in Table 4.8.

4.6 Fault Models

$$\boldsymbol{u}\,(t) = \left[\delta_a \;\; \delta_e \;\; n\right]^{\mathrm{T}} \tag{4.78}$$

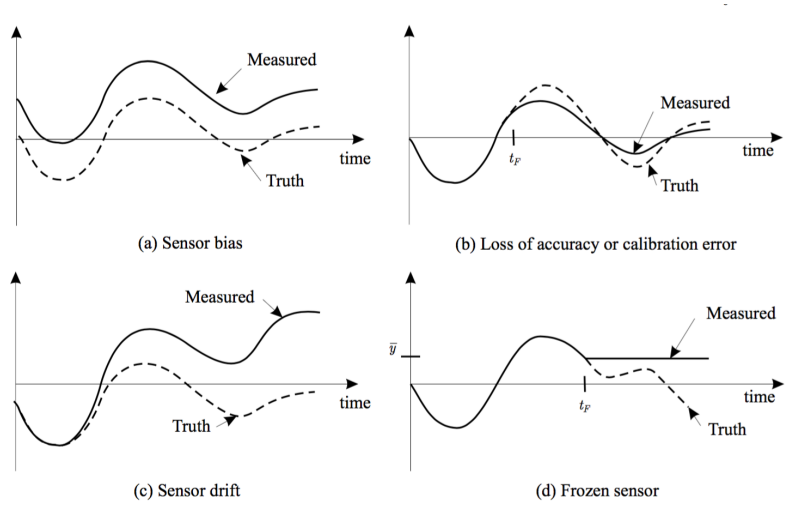

Figure 4-12: Probable sensor faults [15]

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

$$\boldsymbol{u}(t) = \boldsymbol{E}\boldsymbol{u}_c + \boldsymbol{u}_f \tag{4.79}$$

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. 4-13.

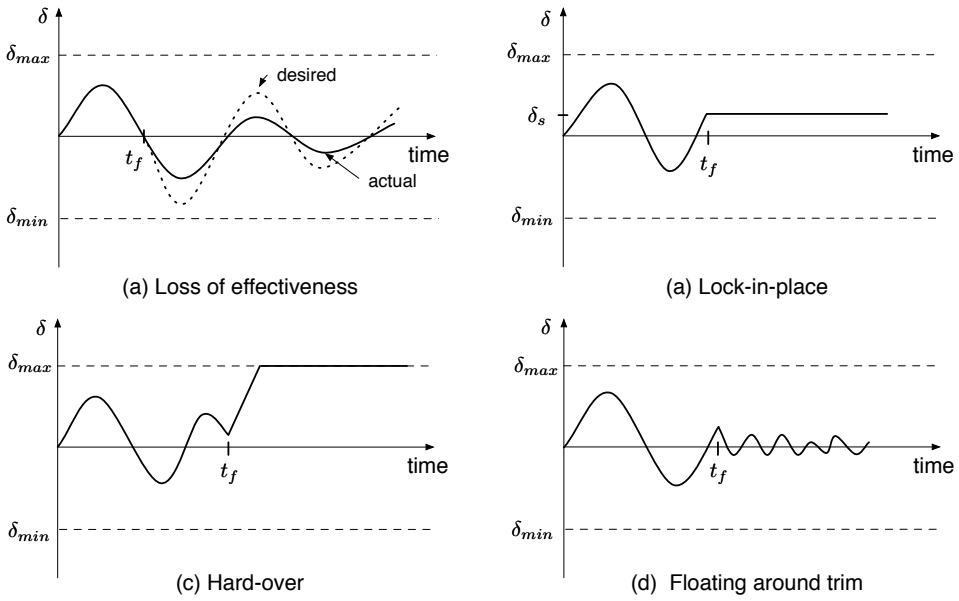

Figure 4-13: Probable actuator faults [15]

Chapter 5

Methodology

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.

5.1 Machine Learning

According to Arthur Samuel, name father of machine learning, "Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed". A rather more to the point but complicated definition offered by Tom Mitchell in 1998, quite new compared to Samuel's in 1959, " 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".

5.1.1 Introduction

Machine learning methods are mainly handled under two main categories: supervised and unsupervised learning as shown in Fig. 5-1. In supervised learning, *"right answer"* for each example in the data set is available to the user apriori. Whereas for unsupervised learning, the *"right answer"* are not available. Although not as common as these methods, there are other machine learning algorithms serving specific purposes such as reinforcement learning and recommender systems. The common

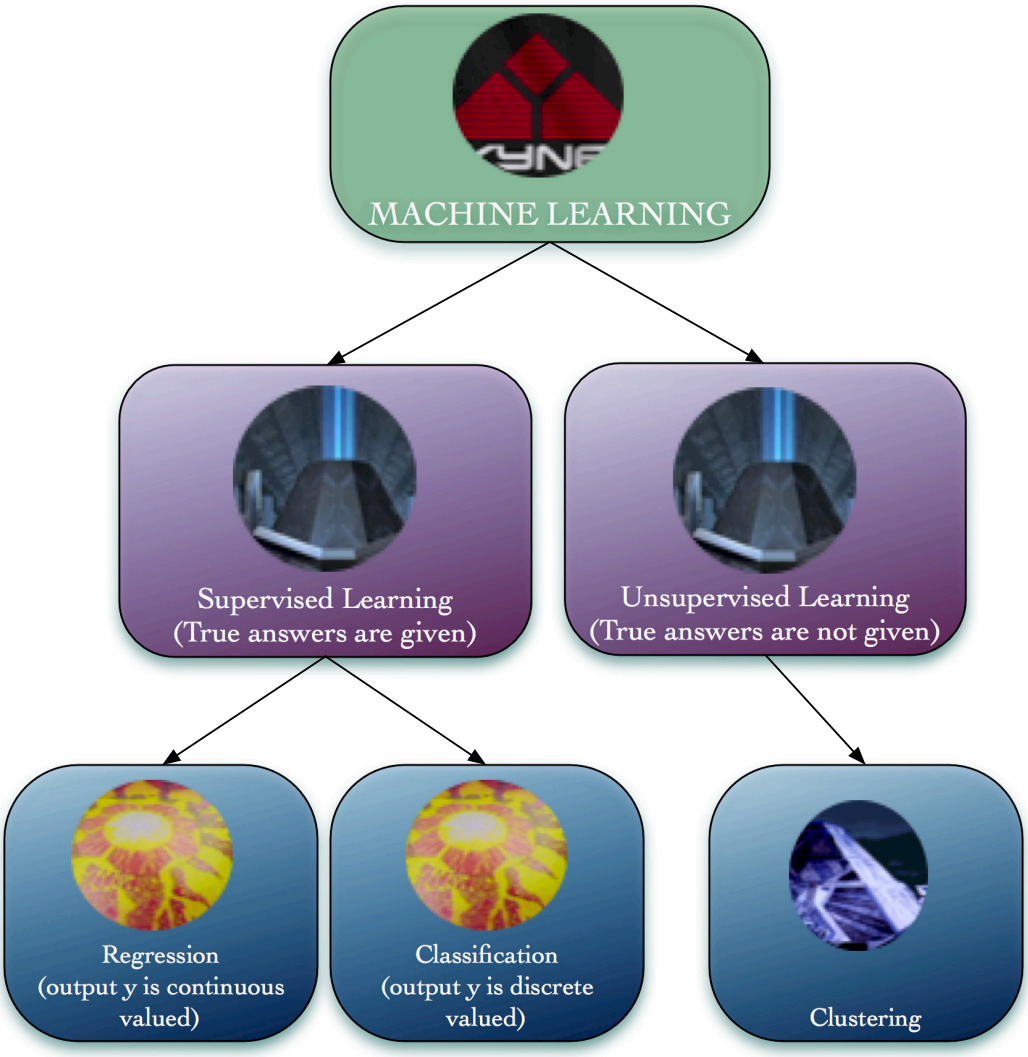

Figure 5-1: Common machine learning methodologies

methodology in machine learning is that there are two phases, the learning and the prediction phases as in Fig. 5-2.

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 will be the output of the system for a given input, depending on the knowledge about the system that you gained via learning phase.

The most common types of supervised leaning is the regression and classification problems. Since these are both of supervised learning types, the *"right answer"* (right

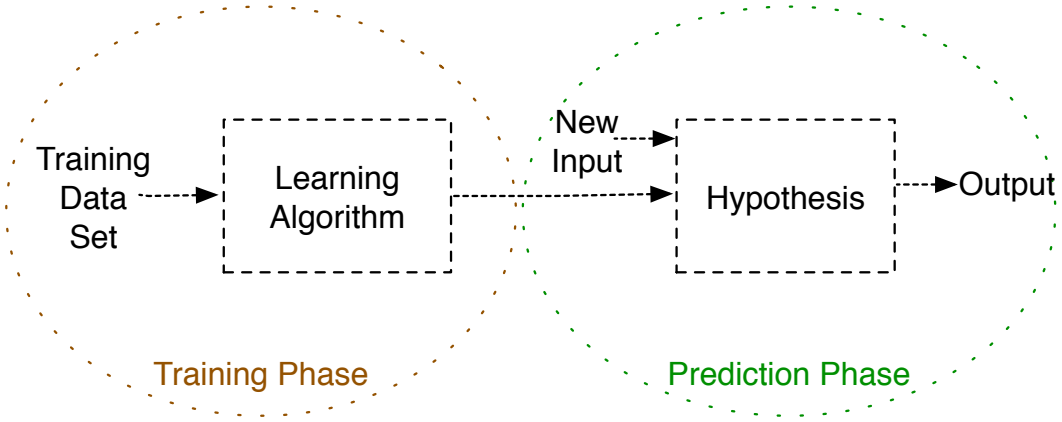

Figure 5-2: Supervised learning basics

Table 5.1: 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
$\mathbf{h_\theta}(\mathbf{x})$	training output
$\mathbf{J}(\boldsymbol{\theta})$	cost function

values of the output y) for each example is assumed to be known.

5.1.2 Terminology

Since being clear about the terminology is essential, a basic introduction to mostly used terms is given in this section. Table 5.1 shows representations of commonly used variables in machine learning. Although representations could differ from one reference to the other, throughout this thesis, one coherent representation given in Table 5.1 is followed.

Table 5.2: Training set $(\mathbf{x}, \mathbf{y})$ of housing prices - one-feature example

training example index (i)	**Size in** $feet^2$ $(\mathbf{x})$	**Price in** $1000\$s$ $(\mathbf{y})$
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 have defaults settings and easy to be used by beginners. Although they would most probably not achieve optimized results which would be the case in the hands of an experienced user, a beginner can easily start using those 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 might change from one to other but still there is 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 Andrew Ng's machine learning course) is given to show the way to constitute the input and output matrix. Table 5.2 uses a supervised learning regression example to show input and output vector representations. The aim of the problem in this example is to predict price of houses for given surface area. For that, known examples of houses with surface area and corresponding price have been given. Since the aim of the problem is to predict the price of a house given the surface area, price of house is the output of the problem. And to predict the price, the available information that is known to effect the price of a house is the surface area of the house, making it the input variable. So, in Table 5.2 each row corresponds to a different house, second column (surface area of house) constitute 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 5.3: 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 would not hold in the real case. In real problems, it is more common that the output would depend not only one variable but more, or many. For that reason, the input vector in a realistic example would rather be an input matrix having different features in its columns as given in Table 5.3. Here in this example, the output is still the price of house, but now the features which corresponds to each column of an input matrix (or sometimes called the feature matrix) are surface area, number of bedrooms, and can be enlarged until n features. It should be kept in mind the selection of features that would lead to a better prediction of the output is a challenging problem and usually requires experience on the system of interest.

5.1.3 Steps towards the learning machine

Visualizing the data

Having a preliminary glance at data might give the designer an idea about the ways to tackle the problem since it is the designer to select the features, the hypothesis, and the type of cost function. It is true that the algorithms are optimizing some parameters of the hypothesis and the cost function but still the structure itself is usually supplied by a human. Although there are tricks to select those in a better sense, machines still have a way to go towards Skynet.

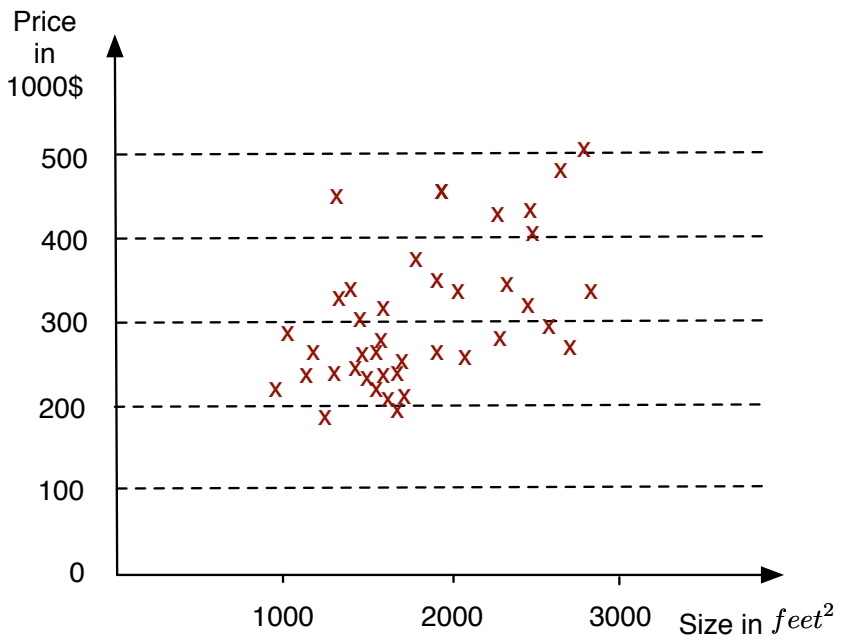

Figure 5-3: Linear regression example - Housing prices as a function of area of the house

Fig. 5-3 shows the housing price depending on the surface area corresponding to the data given in Table 5.2. In this example, price of the houses 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 output y is price which is a continuous value (thus not a label representing a class) this problem is called a regression problem. Plotting the data given shows that the problem could to be modeled by a linear hypothesis (a linear line to fit the data given). Then linear regression can be applied.

Since an example for regression problem has been given, now a preliminary look at classification problem will be presented. Fig. 5-4 is 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).

And the information about the output is represented with the colors of the samples in the feature space. Fig. 5-5 shows the output vs. inputs of a classification problem. This is just to show the difference between regression and classification problem fig-

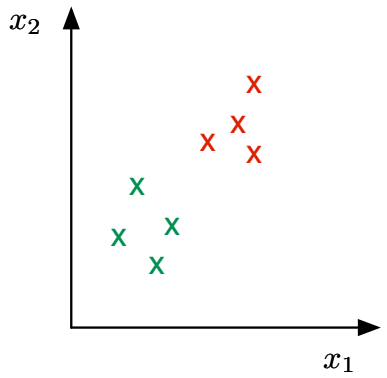

Figure 5-4: Classification example

ures. Usually in regression problems the y-axis represents the output. Here there is a direct analogy by representing a classification problem just as a regression problem, x-axis representing the input whereas y-axis representing the output. But usually in classification problems, value of y (which class it corresponds to) is represented with different colors as in Fig. 5-4 or with different signs such as O and X.

By just plotting the samples, in this example, it seems that logistic regression could yield satisfactory results since the data seems to be separated by a linear decision boundary.

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

In case the number of feature set is small but yet the visualization gives that the model for regression or classification problem can not be represented by a linear model (or linear decision boundary) feature mapping could be applied. Then linear regression or logistic regression (can only fit linear models unless features are mapped) could be applied to nonlinear systems with mapped features.

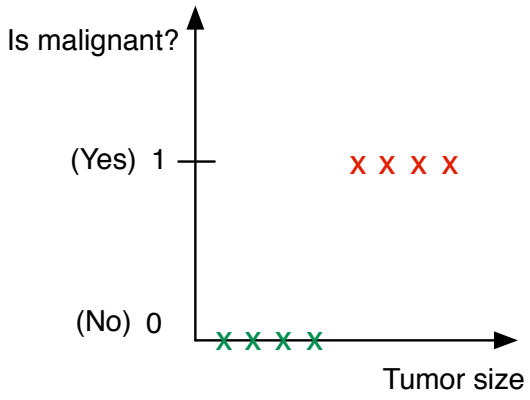

Figure 5-5: Classification example

Feature Mapping

This section applies to problems with smaller feature sets and to fit a nonlinear model with linear or logistic regression. Depending on the results from visualizing data, it might be necessary to map the features since a straightforward implementation of linear / logistic regression will end up in linear model, linear decision boundary respectively. So for cases that the model or decision boundary being linear will not be enough to satisfactorily describe the training data, feature mapping should be applied.

The housing price problem can be revisited here to show how could feature mapping be done in that specific example. The data given in Table 5.2

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\ &= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3 \end{aligned} \tag{5.1}$$

where

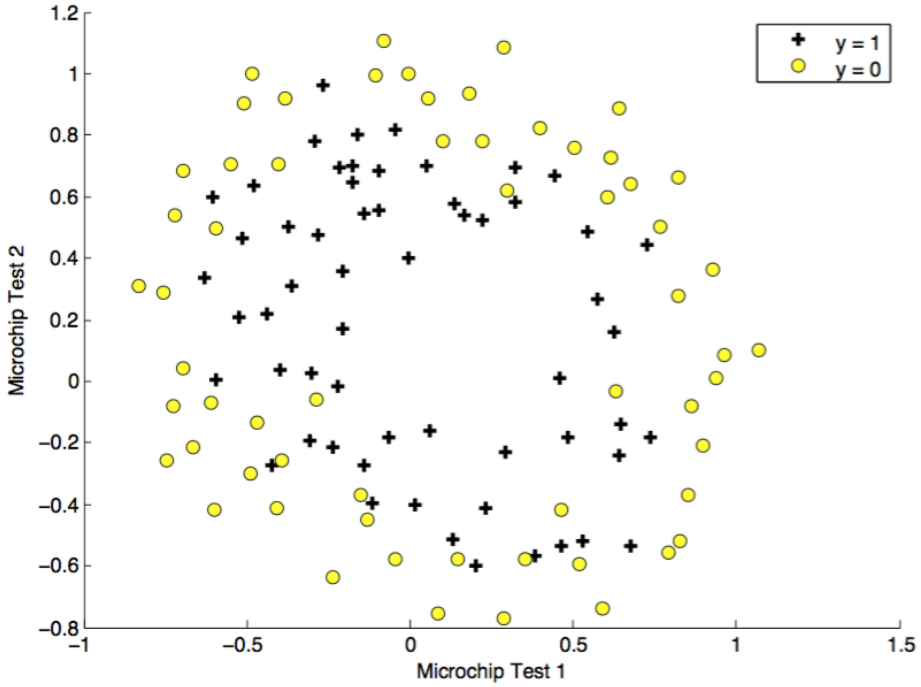

Figure 5-6: Classification example

$$\begin{aligned} x_1 &= size \\ x_2 &= size^2 \\ x_3 &= size^3 \end{aligned} \tag{5.2}$$

Below you can see a data set that will be used for logistic regression for the purpose of classification of a microchip assessment. It is seen from the plot that a linear decision boundary will not serve appropriately. So we might map our features as below

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} 1 & x_1 & x_2 & x_1^2 & x_1x_2 & x_2^2 & x_1^3 \cdots & x_1x_2^5 & x_2^6 \end{bmatrix}^T \tag{5.3}$$

To give an example of how many features you will inject into the problem, assume a binary classification problem with 100 features originally. Since the basic version

of the logistic regression can fit a linear decision boundary only, let's say that we will map the feature set such that only we take the quadratic terms, the new feature set becomes

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} x_1^2 & x_1x_2 & x_1x_3 & \cdots & x_1x_{100} & x_2^2 & x_2x_3 & \cdots & x_3^2 & x_3x_4 \cdots \end{bmatrix}^T \tag{5.4}$$

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

If you consider the 3rd order terms, the new feature set becomes

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} x_1^3 & x_1x_2x_3 & x_1^2x_2 & \cdots & x_{11}x_{13}x_{17} & \cdots \end{bmatrix}^T \tag{5.5}$$

But keep in mind that with a higher dimensional feature vector, overfitting might become a problem. Which can be handled via regularization (Step 6). And also it could be computationally expensive as well.

To fit a model or decision boundary of a circular or elliptical shape, you might consider only the subset of features such that

$$\boldsymbol{x_{mapped}} = \begin{bmatrix} x_1^2 & x_2^2 & x_3^2 & \cdots & x_N^2 \end{bmatrix}^T \tag{5.6}$$

So the number of features will be less but it can not fit complex models or boundaries such as in Fig. 5-7. You have to consider more terms such as given in Equ. 5.4 or you can use Neural Networks. A scheme to help select the strategy for mapping is given in Fig. 5-8.

Selecting Model Structure

If you have more than one features, it might be difficult to have a sense of which model to choose. The idea behind selecting the model or realizing if your hypothesis is overfitting or not is to realize that when you fit some parameter by using some data, it might not be fair to judge if it is a good fit or not by using the same data set. Because in that case, the result could end up by modeling only that data well

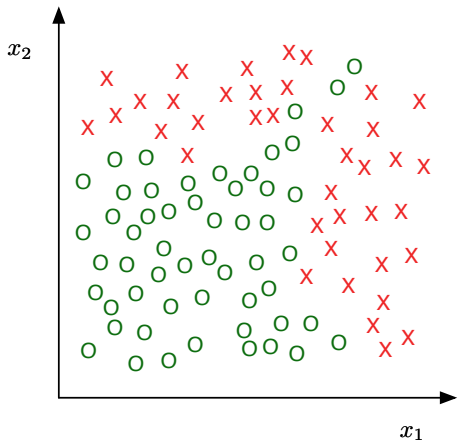

Figure 5-7: Complex decision boundary

but not able to generalize to the new data which is our aim in the first place. So the idea is to have different subsets of data to train the model (calculating the model), to train the dimension of the model and to test if we trained them accurate enough to generalize to the new data set. The difference between calculating the test set error for overfitting (to distinguish if the model is over-fit or not) and calculating the test set error here for model structure selection is that we split the data set to 2 and 3 respectively. The difference in numbers is due to a common sense that when we fit a parameter, we should test it on a different data set to not to have a biased sense of error (the fitting check will give better results on the same data set it is trained on). In model structure selection problem we have an extra parameter we fit (unless it is not obvious that you fit that kind of an extra parameter.) This parameter is the model structure number (d). So when we select which model we choose we should calculate the error of our choice with a different data set. So we need 3 distinct data sets for that problem, to train parameters theta, select the model, and associate an error value with that choice. If we had two data set, and we calculate the test set error with the same data that we have used for selecting the model, than the error for selecting that model will be biased since it is the same data we used for selecting it.

Fig. 5-9.

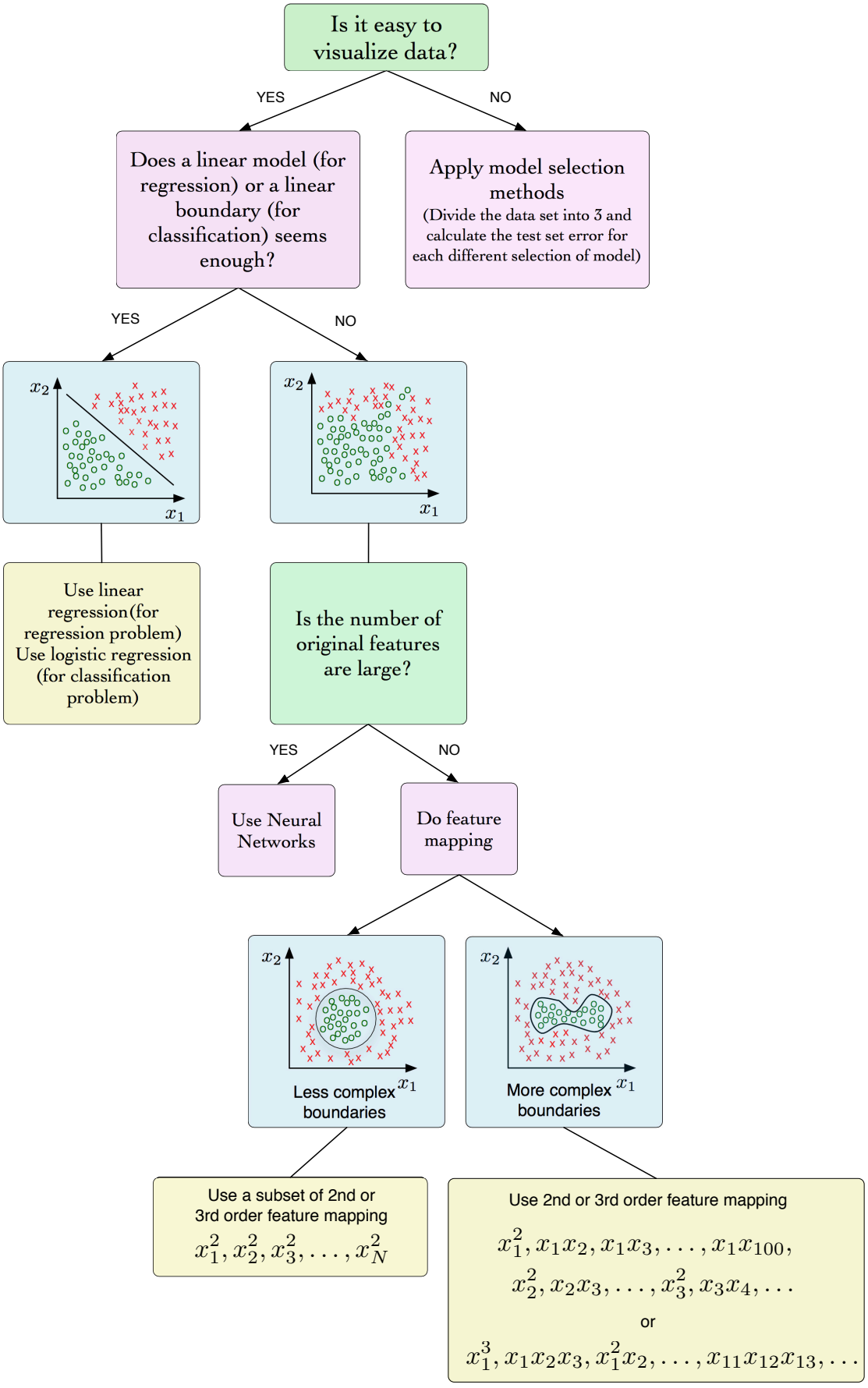

Figure 5-8: Guide for feature mapping

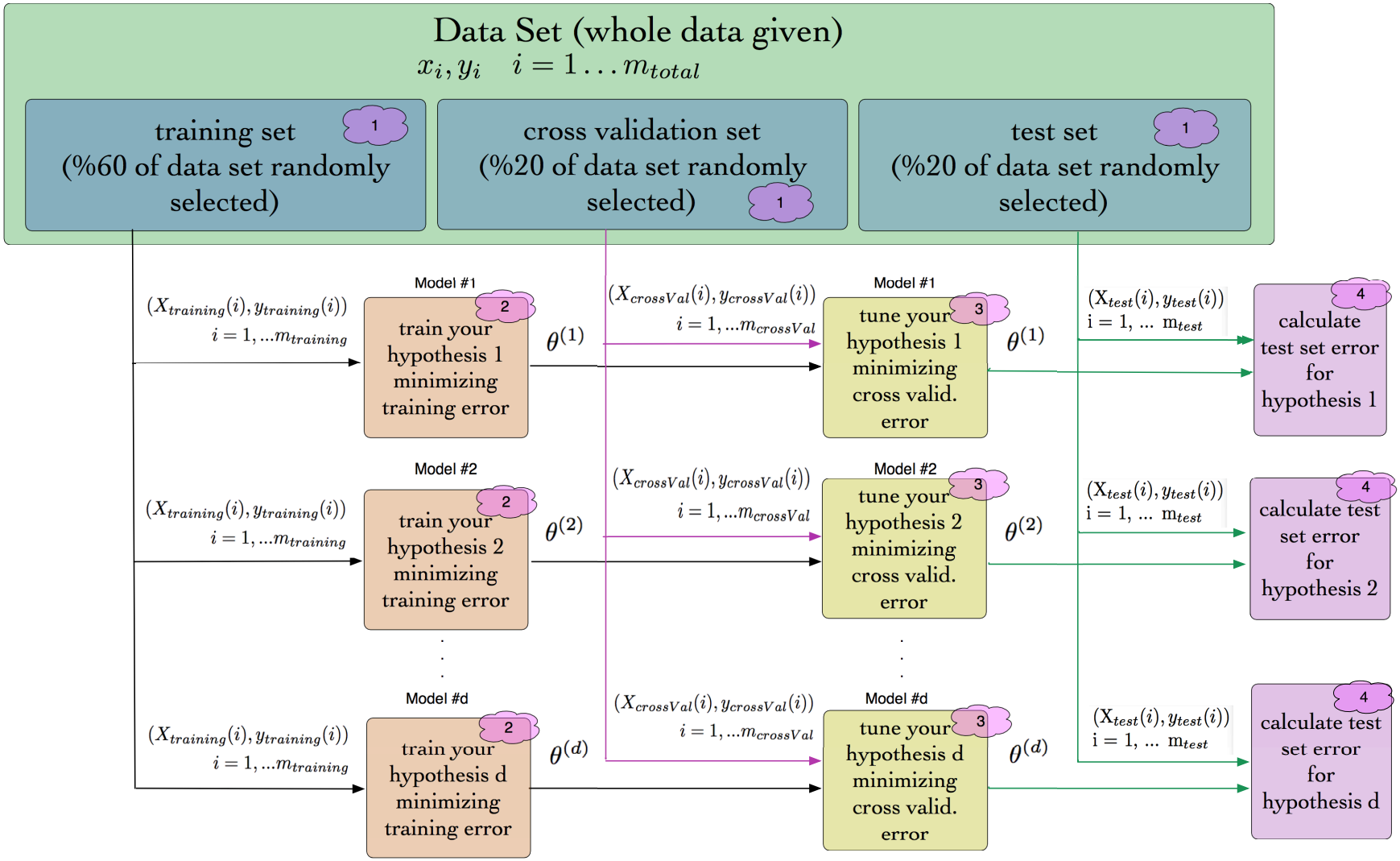

Figure 5-9: Guide for feature mapping

Here the idea is to

1. Train the hypothesis for each model (d = 1,2,3 .. 10 in the example below is the degree of polynomial) by using the training set (%60 of whole data, randomly selected) which will give you the parameters of the hypothesis for the selected model. So you will end up with x number of parameter matrices, where x is the number of models you try (in this example x = 10).

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

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} \left(h_\theta(x_{cv}^{(i)}) - (y_{cv}^{(i)}) \right)^2 \tag{5.7}$$

3. Then by using the cross validation data set (%20 of whole data, randomly selected) calculate cross validation error for each model and select the one (d) with the smallest error.

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - (y_{test}^{(i)}) \right)^2 \tag{5.8}$$

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

Select the d which makes $J_{cv}(\theta^{(k)})$ minimum and then estimate generalization error for the test set $J_{test}(\theta^{(4)})$.

Scaling

The next step is to normalize the features of the data to make the values of features change with the same order of magnitude. The reason is about the calculation of parameters of the hypothesis via an optimization algorithm and its convergence rate. Although there are different methods for normalization, a common one is to manage it

$$\bar{x}_j = \frac{x_j - \mu_j}{s_j} \tag{5.9}$$

where the mean and range is given as

$$\begin{aligned} \mu_j &= \frac{\sum_{i=1}^{m} x_j^i}{m} \\ s_j &= max(x_j) - min(x_j) \end{aligned} \tag{5.10}$$

An important point is to keep that values μ_j, s_j and may be standard deviation if it is used instead of range s_j. During prediction phase, the data first should be scaled with these values attained from learning data. If you have already added artificial feature $x_0 = 1$ (step 4), do not apply scaling to the artificial feature.

Add Artificial Feature

This is for the purposes of generalizing the hypothesis for more features and the ability to write it in vectorized form.

Such that a multiple feature

$$\vec{\boldsymbol{x}} = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_n \end{bmatrix}^T \tag{5.11}$$

where $x_0 = 1$ with a parameter vector

$$\vec{\boldsymbol{\theta}} = \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_n \end{bmatrix}^T \tag{5.12}$$

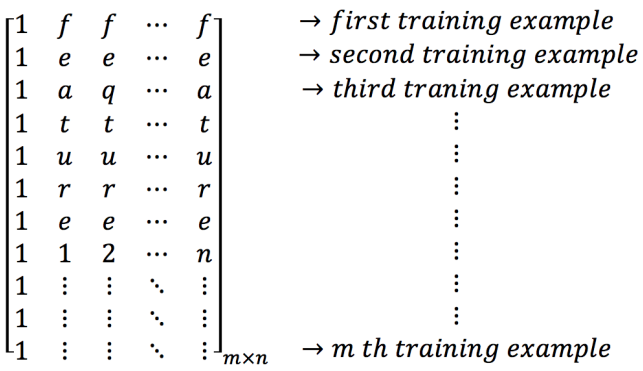

Figure 5-10: Adding an artificial feature of 1s.

since the hypothesis for linear regression is

$$h_\theta(x) = \theta_0 + \theta_1 x_1 \tag{5.13}$$

for one feature and

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \tag{5.14}$$

for multiple features. The more common way to denote hypothesis in an compact generic form is

$$h_\theta(x) = \vec{\boldsymbol{\theta}}^{\mathsf{T}} \tag{5.15}$$

Fig. 5-10.

Selection of the Cost Function and Calculating the Gradient

Cost function and the gradient are usually the inputs of the optimization algorithms by which you will calculate the optimized θ values to fit a model or a decision boundary by using the data set given to you. The selection of cost function is another issue but for standard applications there are widely used cost functions for each type of machine learning (e.g linear regression, logistic regression).

Some examples of cost functions is given as an example below. Cost function for

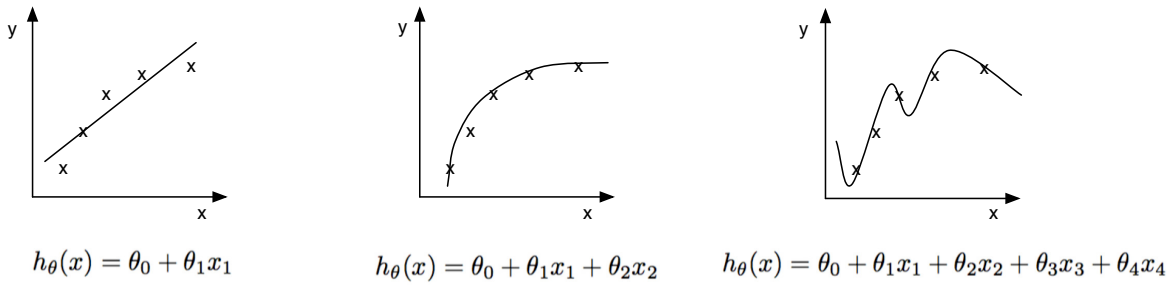

Figure 5-11: Guide for feature mapping

Linear Regression

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \Big(h_\theta(x^{(i)}) - (y^{(i)}) \Big)^2 \tag{5.16}$$

Cost function for Logistic Regression (Classification) - a subcase of two class e.g $y \in \{0, 1\}$.

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \Big[- y^{(i)} log(h_\theta(x^{(i)})) - (1 - (y^{(i)})) log(1 - h_\theta(x^{(i)})) \Big] \tag{5.17}$$

Modify the Cost Function by Adding Regularization Term

Regularization is done to avoid overfitting which can be roughly explained that it is when you end up with a very complicated model or decision boundary, especially when you have nonlinear feature terms such as below;

$$\boldsymbol{x_{mapped}} = \Big[1 \quad x_1 \quad x_2 \quad x_1^2 \quad x_1 x_2 \quad x_2^2 \quad x_1^3 \cdots \quad x_1 x_2^5 \quad x_2^6 \Big]^T \tag{5.18}$$

Overfitting might end up by fitting very well with the training set, but fail to generalize to the new data for the prediction phase. This means that the training set error is not a good predictor for how well the hypothesis would do on new examples.

Fig. 5-11 it is easy to realize that the hypothesis is under-fit, just right or over-

fit since there is only one feature (plot and see). But for problems in which you have more features, it might not be that obvious or even not possible to plot and understand that the hypothesis is over-fit or not. For that cases the standard way to evaluate the hypothesis

1. Split the data set you have to two portions (%30 for the test set - %70 for the training set). Note that if your data set is randomly ordered than you can use the first %70 for training and the rest for test, but if your data is not randomly ordered (let's say that we have data set dependent on time), any kind of pattern in between the data set, you should be randomly selecting this %70 - %30)

2. Learn parameters theta from the new training set (% 70 of the whole data)

3. Calculate the test set error by using test set (%30 of whole data) with the parameters you learned by utilizing training set (% 70 of the whole 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 utilizing the new training set (% 70 of the whole data)

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)}) - (y_{test}^{(i)}) \right)^2 \tag{5.19}$$

   For logistic regression you either use the test set error

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y_{test}^{(i)} \, log(\, h_\theta(\, x_{test}^{(i)})) + (1 - (y_{test}^{(i)})) \, log(\, h_\theta(\, x_{test}^{(i)})) \right] \tag{5.20}$$

   or the misclassification error (0/1 misclassification error)

$$test\ error = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left(err(\, h_\theta(\, x_{test}^{(i)}), (y_{test}^{(i)}) \right) \tag{5.21}$$

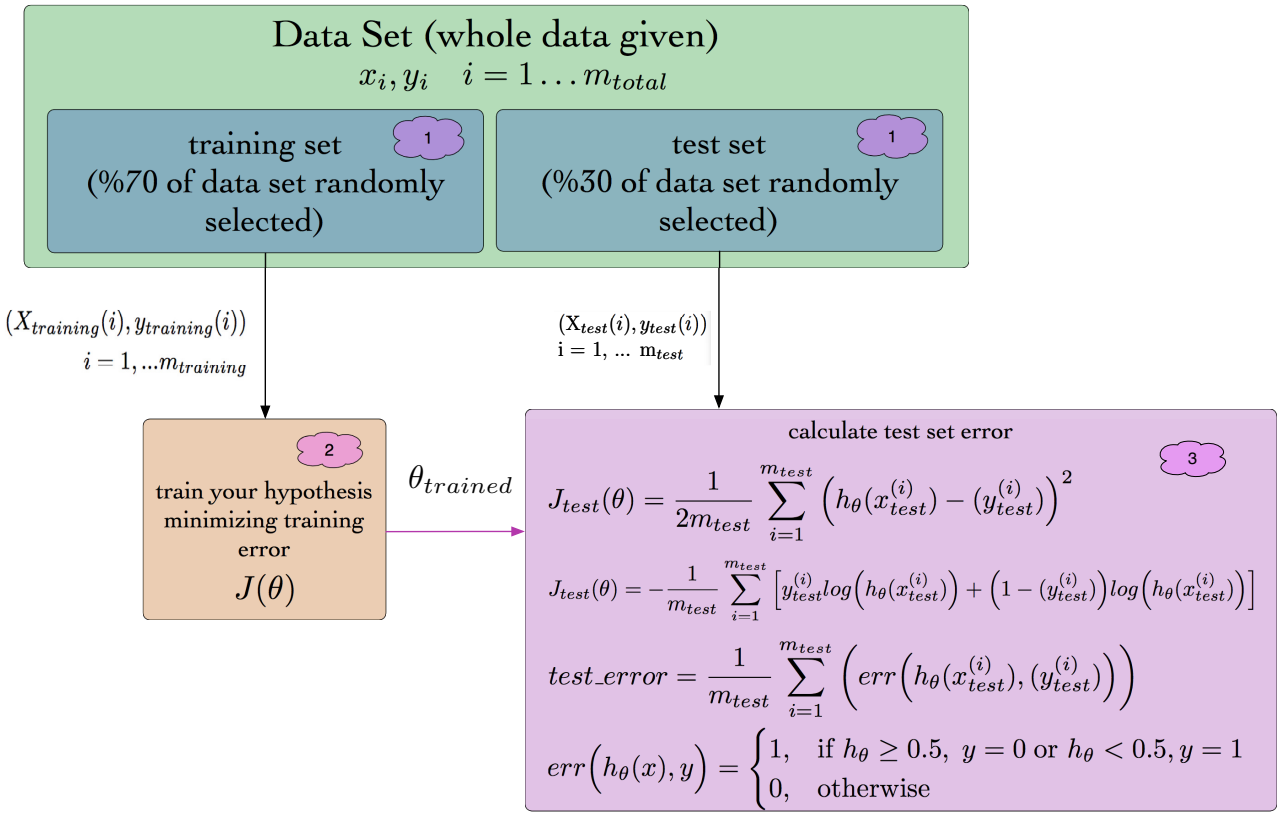

Figure 5-12: Procedure to detect overfitting

$$err(h_{theta}(x), y) = \begin{cases} 1 & if\ h_\theta \geq 0.5,\ y = 0\ or\ if\ h_\theta < 0.5,\ y = 1 \\ 0 & otherwise \end{cases} \tag{5.22}$$

To visualize the procedure, refer to Fig. 5-12.

Regularization:

You add a term to cost function to penalize parameters $\vec{\boldsymbol{\theta}}$ by being large (i.e to make $\vec{\boldsymbol{\theta}}$ as small as possible), but keep in mind that $\vec{\boldsymbol{\theta}}$ is not penalized (by convention).

$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - (y^{(i)})\right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2\right] \tag{5.23}$$

Here, the art is to find the appropriate value for lambda which serves like a weight

between the usual part of the cost function and the second part, which penalize theta vector by being big. So it decreases the values of theta. But if this lambda is bigger than it supposed to be then you end up with

$$h_\theta(x) = \theta_0 \tag{5.24}$$

since you penalized all terms except (as we earlier said, is not penalized by convention). Such that

$$h_\theta(x) = \theta_0 + \cancel{\theta_1 x} + \cancel{\theta_2 x^2} + \cancel{\theta_2 x^3} + \cdots + \cancel{\theta_n x^n} \tag{5.25}$$

Thus you have an under-fit to the model such as

Cost Function Minimization

Here is the point to use the optimization algorithms to minimize cost function by changing the parameters $\vec{\boldsymbol{\theta}}$ (i.e not by changing x and y) which is represented mathematically as

$$\underset{\theta_0, \theta_1}{minimize}\ J(\theta_0, \theta_1) \tag{5.26}$$

For that there are different optimization algorithms. One of them is the Gradient Descent. Others are Conjugate Gradient, BFGS, L-BFGS. Most of them needs the cost function and its gradient to minimize the cost function. The selection of the optimization algorithm is another subject so for now you can select Gradient Descent.

Gradient Descent:

The gradient descent algorithm will be first given for one feature for simplicity for a linear regression model given as in Equ. 5.27

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 \\ J(\theta_1, \theta_2) &= \frac{1}{2m} \sum_{i=1}^{m} \Big(h_\theta(x^{(i)}) - (y^{(i)}) \Big)^2 \end{aligned} \tag{5.27}$$

1. The gradient descent for one feature is given below. Beware that you have to simultaneously update the thetas, meaning that you need to evaluate hypothesis in the summation just above with the previous theta values. So during one iteration, you should use the same theta values to substitute into $h_\theta(x)$, in order to calculate next values of θ_0, θ_1.

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
            $grad0 \leftarrow grad0 + (costFunc(x) - y)$
            $grad1 \leftarrow grad1 + (costFunc(x) - y) * x$
        end for
        $theta0 \leftarrow theta0 - alpha * grad0$
        $theta1 \leftarrow theta1 - alpha * grad1$
    end for
end function

2. Choosing initial value of θ ($theta\_init$)

   A conventional approach is to assign $theta_init = 0$ A point to know is that for different initial values your theta might converge to a different value (local minima but not the global one) as in Fig. 5-13. But for linear regression it is shown that the cost function that we have selected in the notes above is convex, meaning that it has no local minima. It has only one global minima.

   An example of different initial conditions will converge to different solutions (slightly different choice of initial theta might make you converge to the local minima, which gives you $J(\boldsymbol{\theta})$ values smaller than some parts but does not give you the smallest value of $J(\boldsymbol{\theta})$.

3. Choosing learning rate, α ($alpha$)

Algorithm 2 Gradient Descent in a general sense

function GRADDES($X, y, theta\_init, alpha, num\_iters$)
    Inputs: X - training inputs,
        y - training outputs,
        theta_init - initial guess for the optimized parameter theta,
        alpha - learning rate,
        num_iters - number of iterations to find the optimized theta
    Outputs: theta - theta_optimized/ theta vector that minimizes the cost function
        J - the values of the cost function calculated during the course of iterations
    repeat until convergence
    $\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \cdots)$
    For correct implementation, update simultaneously e.g
        $temp0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1, \cdots)$
        $temp1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1, \cdots)$
            $\vdots$
        $\theta_0 = temp0$
        $\theta_1 = temp1$
            $\vdots$
end function

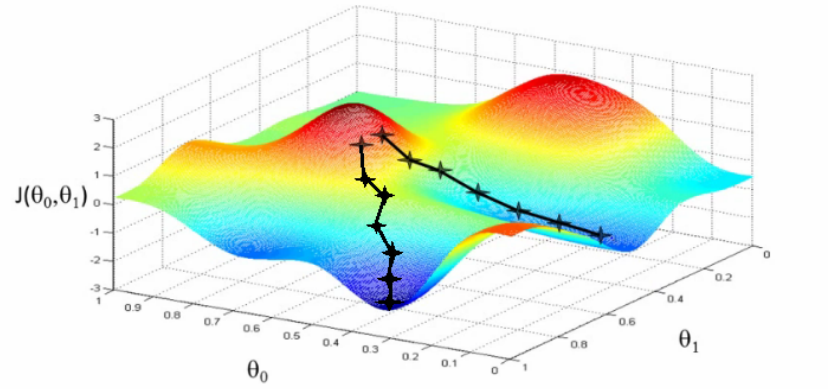

Figure 5-13: Gradient descent convergence dependance on $theta_i nit$. Two different but close choice of $theta_i nit$ might converge to local or global minima

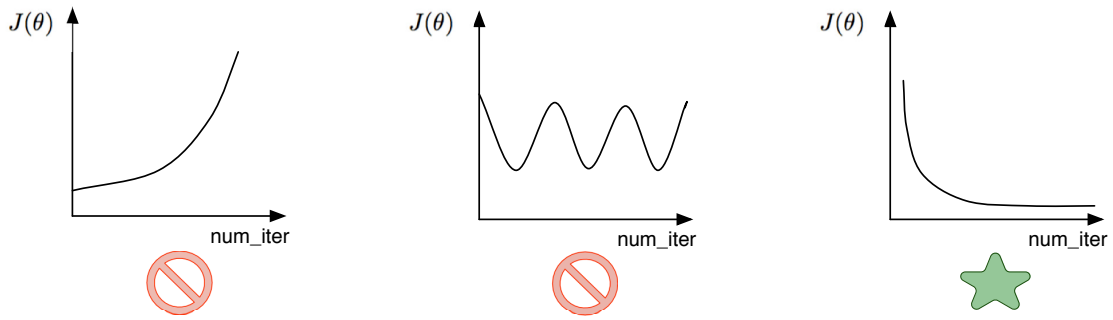

Figure 5-14: Evolution of $J(\theta)$ with respect to number of iterations. Left two figures showing that the optimization problem is not converging. The figure in the far right is the $J(\theta)$ evolution expected

Alpha, learning rate gives the gradient descent its step size, so a bigger value means a faster convergence. But keep in mind for a big alpha, your solution might not converge or even diverge. And too small value might end up the optimization to converge very slowly. So you can try in order these values of $alpha = 0.001, 0.01, 0.1, 0.003, 0.03, 0.3$

4. Choosing number of iterations (num_iter)

   You might start with $num\_iter = 1000$ and then by visualizing the cost function check if the cost function converged to a steady value or not. If it has not yet converged but decreasing, try a bigger value for num_iter

Visualizing the cost function

Check converge by visualizing the cost function as a function of iterations of the optimization algorithm. Normally you expect this plot to be decreasing, never increase, converging to a steady value at the end of the optimization algorithm.

To be able to observe how the cost function $J(\theta)$ during the iterative process of optimization, you should save the cost function values at each iteration. Two examples of a bad optimization implementation can be seen in Fig. 5-14.

Not a good sign! Decrease alpha(learning rate) What you expect is something like this Fig. 5-14.

WRITE SOME HERE Fig. 5-15

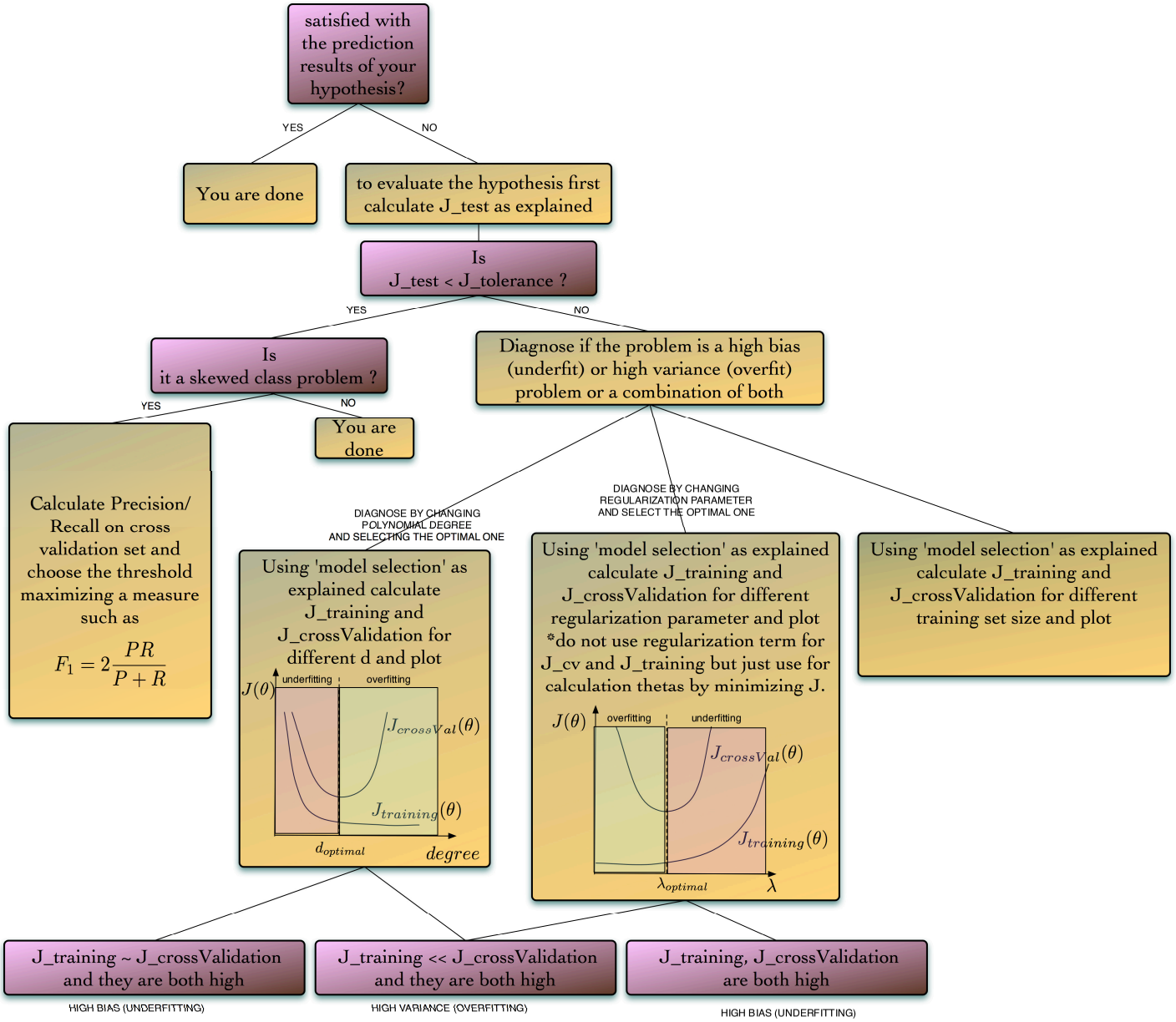

Figure 5-15: Diagnosis the problem of machine learning by comparing the cost function for the training and cross-validation data sets

Visualizing the model/classifier (decision boundary) with respect to training data

Overfitting can be roughly explained that it is when you end up with a very complicated model or decision boundary. This ends up by fitting very well with the training set, but fail to generalize to the new data for the prediction phase. If overfitting occurs apply regularization meaning; Go step 6.

Prediction

For a new set of input data, now you will predict the outcome thanks to the model/-classifier you trained. Now you should first normalize the new data X_new by using the same mean and standard deviation values we had previously calculated from the training set. And then evaluate h(theta) by the optimized theta and regularized new data set X_new_regul.

5.2 Support Vector Machines

5.2.1 Introduction

Let $E = \Big\{\big(\vec{\boldsymbol{x}}_1, y_1\big), \big(\vec{\boldsymbol{x}}_2, y_2, \cdots, \big(\vec{\boldsymbol{x}}_m, y_m\big)\big)\Big\}$, where $\vec{\boldsymbol{x}}_i \in \mathbb{R}^n$ and $y_i \in \big\{0, 1\big\}$ be a training example set. Assuming the training data linearly separable,

SVM is a relatively new approach for classification offering better generalization property thanks to its foundations on the structural risk minimization principle [22, 45] 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

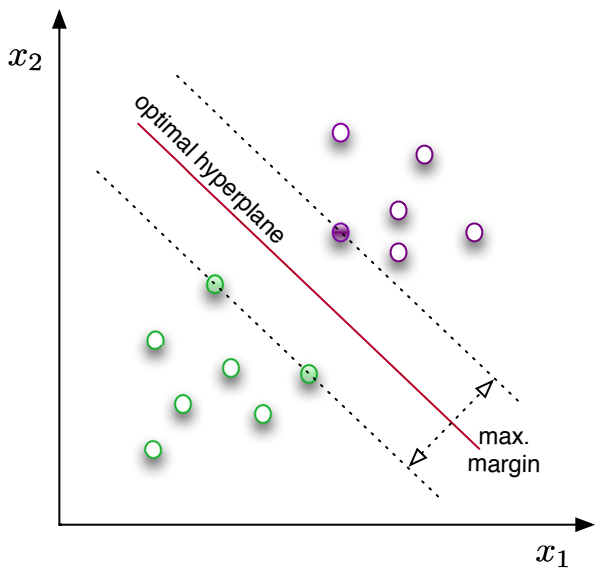

Figure 5-16: SVM working principle

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. 5-16. 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 separable 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 one of the control surface stuck at 0°. SVM being a supervised classification algorithm has two main phases as shown in Fig. 6-4. In the training phase, the model is learned as a fit to the labeled data that is fed to the SVM algorithm. This phase is usually followed with a tuning phase where some of the parameters of SVM 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

$$min_{\gamma,\omega,b} \quad \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{m}\xi_i \tag{5.28}$$

$$s.t. \quad y^i(\omega^T x^(i) + b) \geq 1 - \xi_i,\ i = 1, \cdots, m \tag{5.29}$$

$$\xi_i \geq 0,\ i = 1, \cdots, m \tag{5.30}$$

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 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 [33] is used to calculate the posterior probabilities giving the probability that the new measurements belongs to faulty mode. Results shows as in Fig. 6-5 that proper tuning achieves very accurate and instant detection for the drone fault.

5.2.2 Application

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

SVM being a supervised classification algorithm has two main phases as shown in Fig. 6-4: training and prediction. The labeled data set 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.

Training of the classifier

In the training phase, the model is learned as a fit to the labeled data that is also an input to the SVM algorithm. 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 number of 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 next step is to normalize the features of the data to make the values of

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

The aim of SVM is to find an optimal hyperplane maximizing the margin by solving the optimization problem for non-linearly separable datasets. SVM implements the idea of having a confidence in the prediction by using the concept of separating data with large margin. Some other classification methods, such as logistic regression, outputs the probability of a new instance's class, giving the confidence of the prediction as output inherently. On the other hand, SVM does only output if the new instance belongs to a class not necessarily pointing the confidence on this decision. Rather than including this confidence information as an output in terms of probabilities, this information is introduced with the functional and geometric margins. These definitions also serve for mathematically convenience, so that the optimization problem can be represented as a convex optimization problem. Furthermore, introducing the Lagrange duality to obtain the dual form of the optimization problem, use of kernels, to work efficiently in higher dimensional spaces, is eased. The dual form also allows to utilize efficient optimization solvers such as Sequential Minimal Optimization (SMO) [32] which is the solver used in this work as well. Conventionally, training phase of SVM requires to solve a large quadratic programming (QP) problem. Especially for large training set, computational heaviness of this phase might limit the applicability of SVM to specific problems sets. To overcome this constraint, SMO breaks the QP problem into a series of smaller QP problems which can be solved analytically. Default settings for SVM binary classification, included in Matlab Statistics and Machine Learning Toolbox, utilizes SMO for optimization if outlier fractions has not been specified during the function call.

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

as

$$K(x,z) = \phi(x)^T \phi(z) \tag{5.31}$$

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

$$K(x,z) = exp\bigg(- \frac{\|x-z\|^2}{2\sigma^2} \bigg) \tag{5.32}$$

Tuning of the classifier

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

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

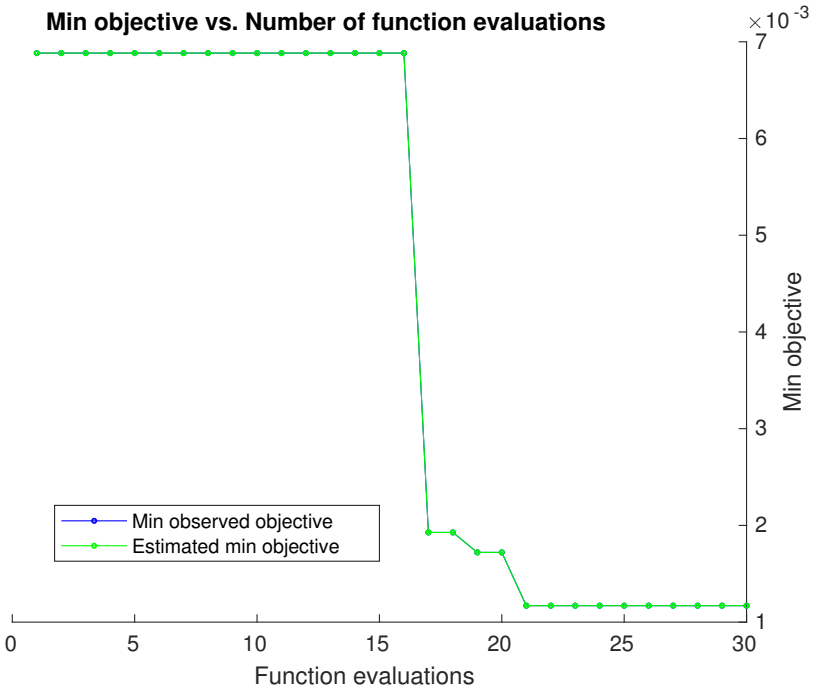

Figure 5-17: Convergence of minimum objective function

Box constraint (C) and kernel scale (sigma) are tuned considering the presence of the outliers to generalize the distribution of the data rather than resulting in fine fits for each individual data in the training set.

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

Bayesian optimization tool from Matlab can be used in conjunction with the classification tool to optimize box constraint and the kernel scale. This tool outputs minimum objective value as a function of number of function evaluations (max. number of function evaluations can be changed by passing arguments to the function) as shown in Fig. 5-17. Fig. 5-21 shows the objective function values for a variety

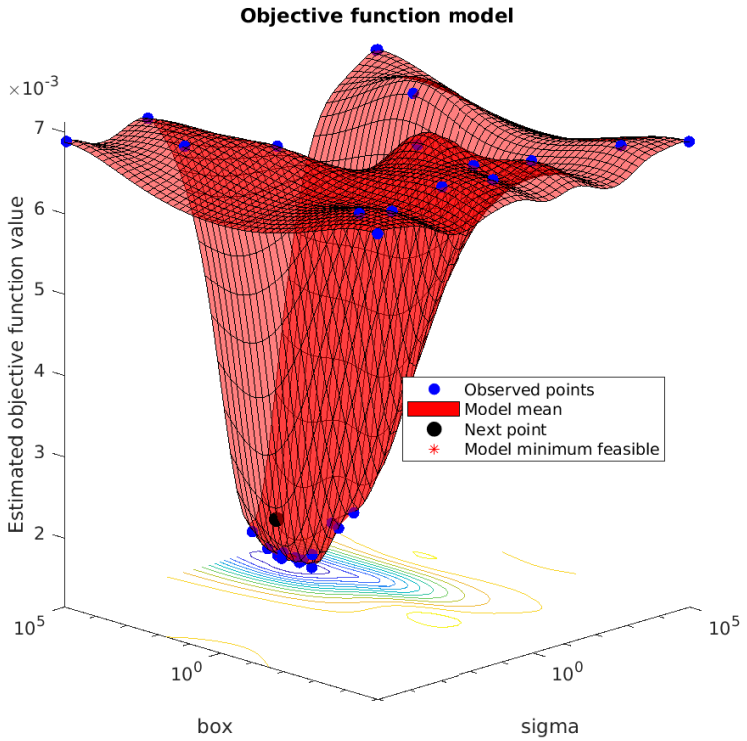

Figure 5-18: Objective function values for different box parameter and sigma values

```
Optimization completed.
MaxObjectiveEvaluations of 30 reached.
Total function evaluations: 30
Total elapsed time: 1670.0722 seconds.
Total objective function evaluation time: 1650.8993

Best observed feasible point:
    sigma      box
    ______    ______
    14.358    8929.2
Observed objective function value = 0.0011013
Estimated objective function value = 0.0011015
Function evaluation time = 58.5071

Best estimated feasible point (according to models):
    sigma      box
    ______    ______
    9.7729    2027.5
Estimated objective function value = 0.0011015
Estimated function evaluation time = 22.4874

results =
  BayesianOptimization with properties:

                     ObjectiveFcn: [function_handle]
             VariableDescriptions: [1×2 optimizableVariable]
                          Options: [1×1 struct]
                     MinObjective: 0.0011
                  XAtMinObjective: [1×2 table]
            MinEstimatedObjective: 0.0011
          XAtMinEstimatedObjective: [1×2 table]
           NumObjectiveEvaluations: 30
                 TotalElapsedTime: 1.6701e+03
                        NextPoint: [1×2 table]
                           XTrace: [30×2 table]
                   ObjectiveTrace: [30×1 double]
                 ConstraintsTrace: []
                    UserDataTrace: {30×1 cell}
      ObjectiveEvaluationTimeTrace: [30×1 double]
               IterationTimeTrace: [30×1 double]
                       ErrorTrace: [30×1 double]
                 FeasibilityTrace: [30×1 logical]
       FeasibilityProbabilityTrace: [30×1 double]
              IndexOfMinimumTrace: [30×1 double]
           ObjectiveMinimumTrace: [30×1 double]
    EstimatedObjectiveMinimumTrace: [30×1 double]
```

Figure 5-19: Objective function values for different box parameter and sigma values

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	sigma	box
1	Best	0.0068833	5.4704	0.0068833	0.0068833	1300.5	156.44
2	Accept	0.0068833	11.849	0.0068833	0.0068833	3.8141e-05	0.026371
3	Accept	0.0068833	5.7804	0.0068833	0.0068833	18.587	0.00033069
4	Accept	0.0068833	74.913	0.0068833	0.0068833	0.0016458	6284
5	Best	0.0013078	329.69	0.0013078	0.0013078	8.472	74623
6	Accept	0.0013767	256.94	0.0013078	0.0013078	6.5809	49767
7	Best	0.0011013	58.507	0.0011013	0.0011013	14.358	8929.2
8	Accept	0.0013078	95.294	0.0011013	0.0011013	31.364	61151
9	Accept	0.001239	11.969	0.0011013	0.0011013	4.9097	262.14
10	Accept	0.0011013	22.669	0.0011013	0.0011013	9.7729	2027.5
11	Accept	0.0068833	5.0633	0.0011013	0.0011013	0.024383	1.0133e-05
12	Accept	0.0011702	37.825	0.0011013	0.0011014	9.3873	4382.3
13	Accept	0.0017896	7.0261	0.0011013	0.0011014	15.5	226.66
14	Accept	0.0028221	6.5309	0.0011013	0.0011014	0.86327	416.18
15	Accept	0.001239	20.717	0.0011013	0.0011013	6.9169	1016.2
16	Accept	0.0011702	6.2389	0.0011013	0.0011013	2.4378	8.1869
17	Accept	0.0011702	7.253	0.0011013	0.0011013	3.2296	26.858
18	Accept	0.0029598	7.8493	0.0011013	0.0011013	0.69763	0.54765
19	Accept	0.0013078	20.199	0.0011013	0.0011014	14.755	3342
20	Accept	0.0011702	292.51	0.0011013	0.0011014	17.65	96435

Iter	Eval result	Objective	Objective runtime	BestSoFar (observed)	BestSoFar (estim.)	sigma	box
21	Accept	0.0022026	5.5539	0.0011013	0.0011014	4.7413	4.9864
22	Accept	0.0014455	7.1498	0.0011013	0.0011014	1.7326	19.251
23	Accept	0.0011702	122.78	0.0011013	0.0011014	16.137	28424
24	Accept	0.0027533	38.785	0.0011013	0.0011015	2.0161	98226
25	Accept	0.0013078	10.156	0.0011013	0.0011015	8.417	301.45
26	Accept	0.001652	50.602	0.0011013	0.0011015	69.585	87379
27	Accept	0.001239	9.3285	0.0011013	0.0011015	2.8662	102.24
28	Accept	0.0011702	107.06	0.0011013	0.0011015	11.912	16671
29	Accept	0.0068833	5.1546	0.0011013	0.0011015	91781	1.0261e-05
30	Accept	0.002065	10.042	0.0011013	0.0011015	292.18	96946

Figure 5-20: Objective function values for different box parameter and sigma values

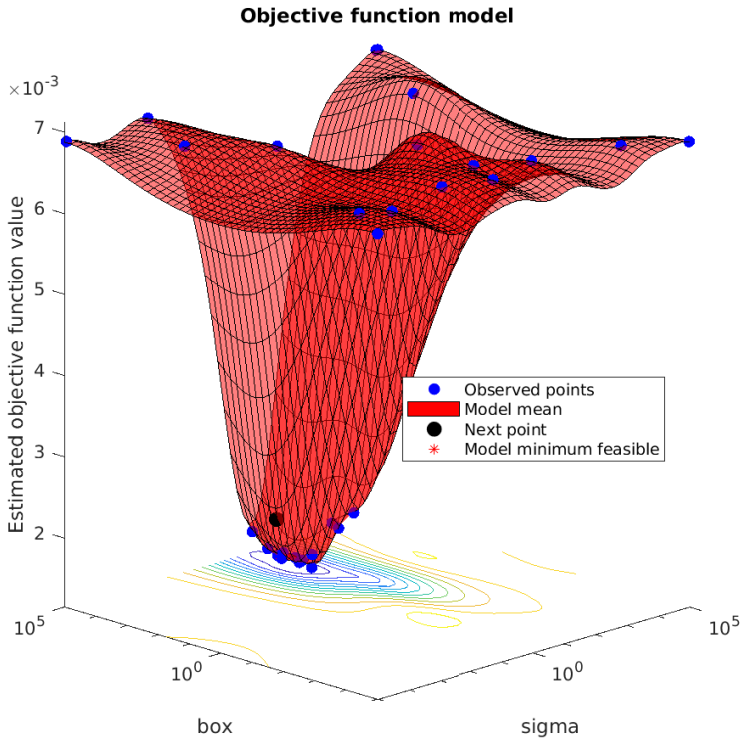

Figure 5-21: Objective function values for different box parameter and sigma values

of box constraint and kernel scale values. The optimized values for box constraint and kernel scale have also been presented in the tables for each classification problem considered.

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

Evaluating the classifier

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

Another means to evaluate performance of a classifier, available under Matlab Statistics and Machine Learning Toolbox, is via observing the *classification edge*. The *edge* is the weighted mean of the *classification margins*. Here, the *classification margin* for a binary classifier is defined, for each observation, as the difference between the *classification score* for the *true class* (faulty measurements in the considered problem) and the *classification score* for the *false class* (nominal measurements in the considered problem). In this definition, *classification score* is considered as the signed distance from each observation to the decision boundary.

While all these variables would be satisfactory for performance analysis of classifiers, due to the skewed-class nature of the problem, another means of evaluations is necessary. When the number of instances for different classes in a data set has a big difference, it is named as a skewed-class problem. The inherent trickiness to evaluate

classifiers for such problems lies on the fact that, predicting only the more frequent class might lead to a misunderstanding that the classifier gives superior performance although it might not be even learning at all. For the problem of fault detection of a control surface would serve as a good example to clarify more. Since nominal data is not difficult to generate in real flight while it is difficult to fly faulty, the nominal data is much vast compared to the faulty data

For such problems, in order to define a single metric to evaluate the abilities of classification, *precision* and *recall* should be defined. Fig. 5-22 shows the confusion matrix which is used to calculate the precision and recall. Here, in general, the class indicated by 1 is the skewed class, corresponding to the fault class in this study. True positive refers to the number of instances that are predicted faulty and are actually faulty, false positive refers to the number of nominal instances which are miscalculated or predicted as faulty, false negative is the number of instances that are actually faulty but the classifier predicted that they are nominal, and finally the true negative is the number of instances that are predicted as nominal and are actually nominal. Precision gives a measure on the fraction that was actually faulty of all the measurements that was predicted as faulty. Precision can be related to number of false alarms which is cautiously avoided in flight control systems. Precision is defined as

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \tag{5.33}$$

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

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \tag{5.34}$$

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

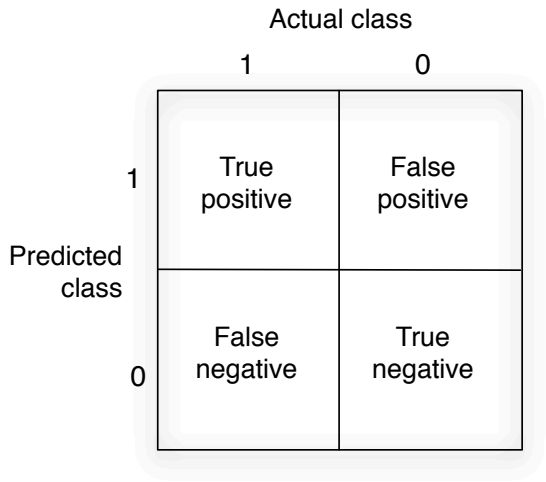

Figure 5-22: Confusion matrix

score is defined as

$$f1Score = \frac{2 * precision * recall}{precision + recall} \tag{5.35}$$

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

Chapter 6

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 starts with the explanations on the path to generate faults in real flight thoroughly for two reasons, the importance of having knowledge on data, how it is generated/labeled, and also constructing a guide for researches to realize their own faulty flight campaigns. After that, classification for control surface stuck and loss of effectiveness faults have been investigated separately.

6.1 Fault detection from simulated flight data

In this section, model of an aircraft is simulated in Matlab using the equations of motion given Equ. (Summarized Equations of motion reference). This drone model, will not be used for the design of FDD algorithms, but to generate data that will be used by the FDD algorithms. After the equations of motion of drone have been solved numerically, accelerometer and gyro measurements have been simulated based on the statistics of the sensors onboard. 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 uses a ZAGI drone.

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

$$\boldsymbol{u}(t) = \begin{bmatrix} \delta_a & \delta_e & n \end{bmatrix}^{\mathrm{T}} \tag{6.1}$$

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

$$\boldsymbol{u}(t) = \boldsymbol{E}\boldsymbol{u}_c + u_f \tag{6.2}$$

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 additive actuator fault. This model makes it possible to simulate all four types of actuator faults shown in Fig. 4-13.

Most of the FDD algorithms are implemented to open-loop systems, ignoring the probable influences of the controller might cause on the detection performance [31]. Throughout this first section, the system is open-loop as well. Here, a step by step approach is followed. In this first section the effect of controller is ignored while in the next section, in which real flight data is utilized, diagnosis is achieved aside a functioning controller.

The code for simulations in this section can be reached in *Github*[1]. First, the measurements are simulated for faulty and and nominal flight conditions. As an example to an input in a loss of efficiency faulty condition could be an actual aileron command of $\delta_a^a = 1^\circ$ corresponding to a desired $\delta_a^d = 4^\circ$. The measurements are labeled correspondingly.

It is always important to visualize the features to have a grasp of data structure before applying machine learning algorithms. For that reason, available observations forms the 6-dimensional pattern space, $\vec{\boldsymbol{y}} = \begin{bmatrix} a_x & a_y & a_z & \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ can be visual-

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

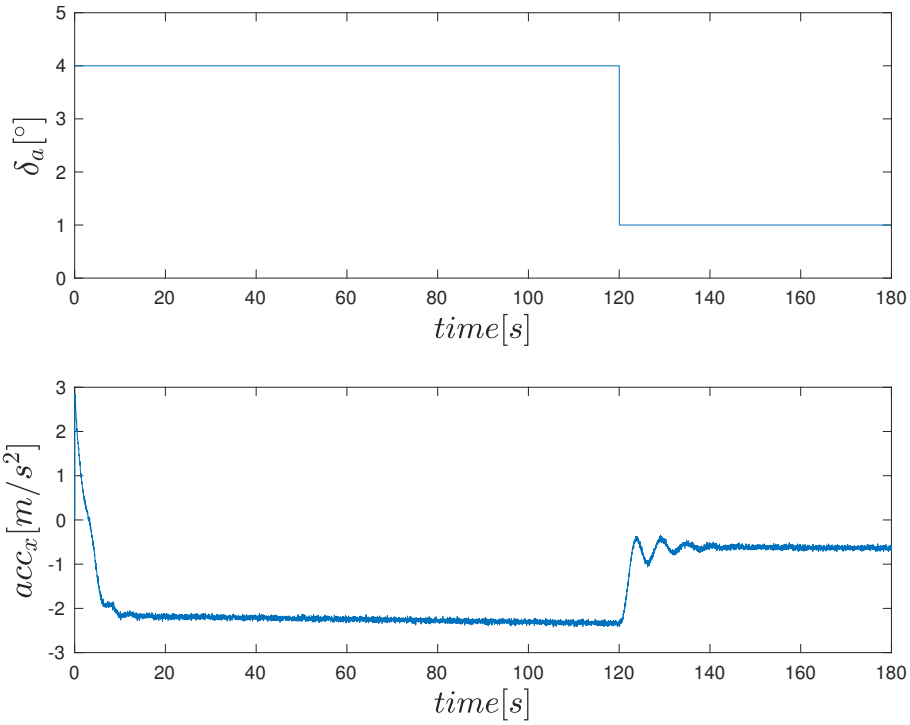

Figure 6-1: Loss of effectiveness fault simulation in aileron command and corresponding accelerometer x axis measurement

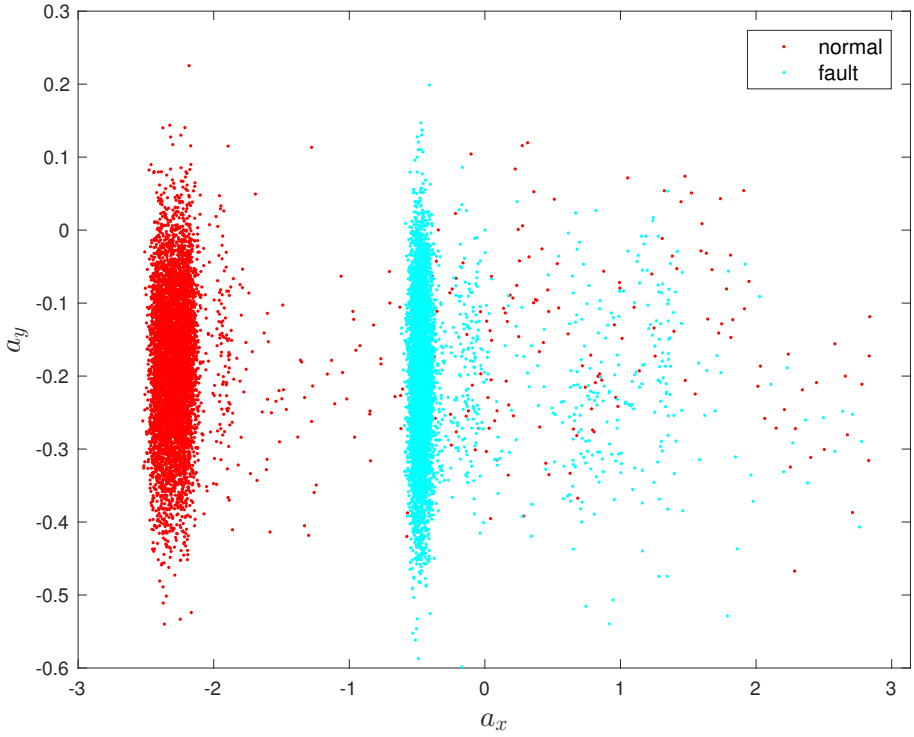

Figure 6-2: Accelerometer simulation a_x vs a_y

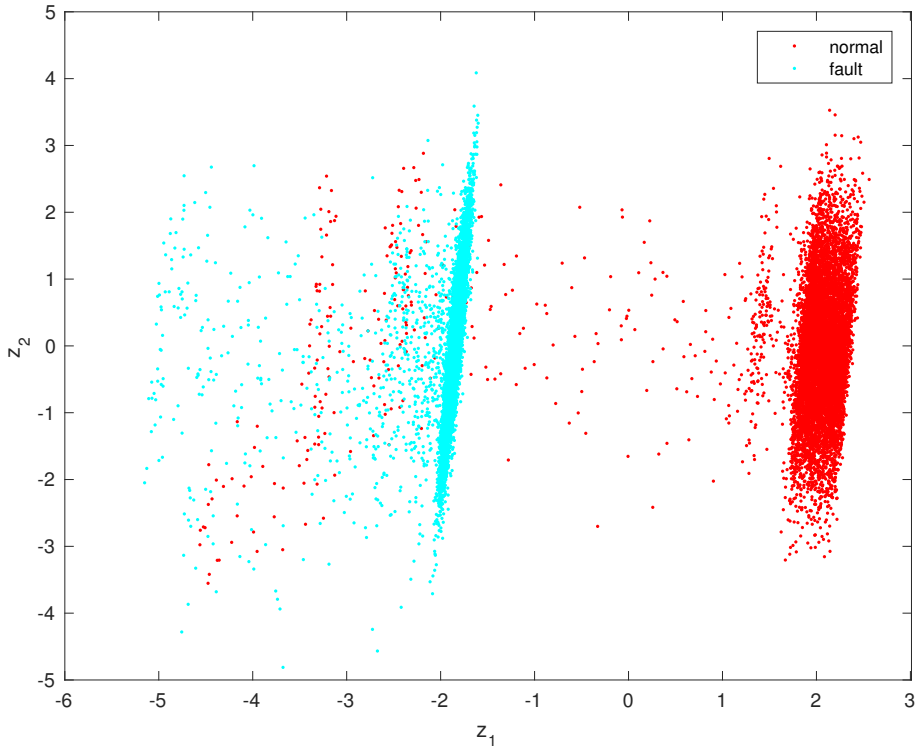

Figure 6-3: Reduced dimensional space features z_1 vs z_2

ized in pairs such as Fig. 6-2. There are further methods to visualize multidimensional data such as *Tours*,[3, 11, 12], GGobi data visualization system [13]. In this work, dimensionality reduction technique, Principle Component Analysis (PCA), is utilized for visualization. If very 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 faulty and nominal measurements' distribution in feature space as shown in Fig. 6-3.

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. With default values of MATLAB SVM toolbox, a hypothesis is trained by fitcsvm function%70 of whole data.

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

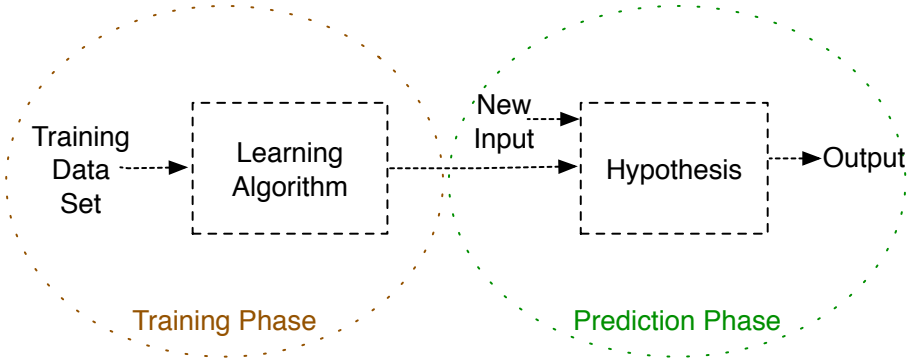

Figure 6-4: Supervised learning basics

Fig. 6-4. In the training phase, the model is learned as a fit to the labeled data that is fed to the SVM algorithm. This phase is usually followed with a tuning phase where some of the parameters of SVM 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 $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ 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.

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

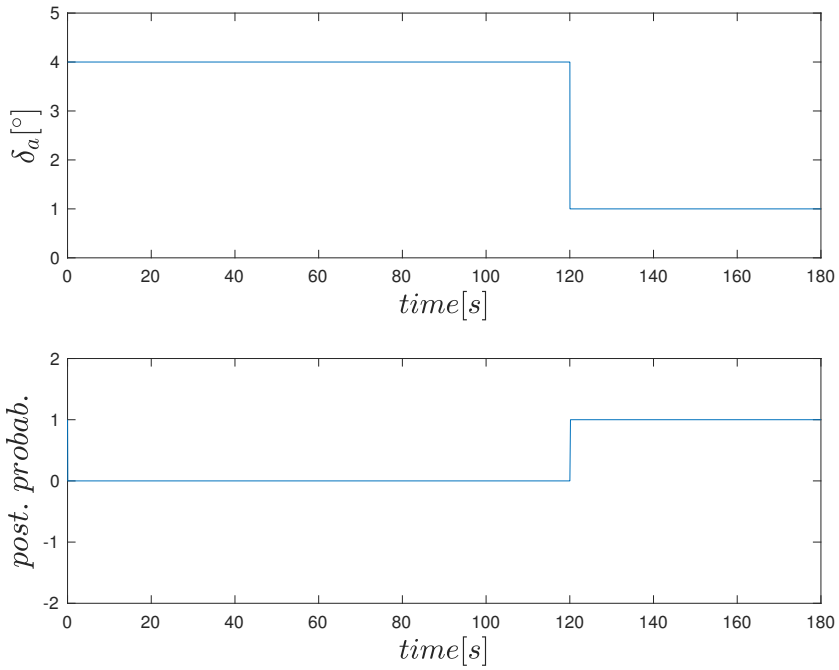

Figure 6-5: Posterior probability of loss in effectiveness fault for test set when a fault is injected at t = 120s.

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 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 [33] is used to calculate the posterior probabilities giving the probability that the new measurements belongs to faulty mode. Results shows as in Fig. 6-5 that prober tuning achieves very accurate and instant detection for the drone fault.

6.2 Fault detection from real flight data

This second section of the chapter discusses the classification results using real flight data. First, the injection of faults to the flights have been explained throughly. The Paparazzi GCS has been altered to inject faults real-time, controllers in some of flight modes have been changed to accommodate faults. After, the selection of the faults and nominal phases and also labeling data have been presented. Finally, the labeled data have been used to train classifiers for elevon stuck and loss of efficiency faults separately and the classifiers have been evaluated. A variety of techniques implemented to ameliorate the performance such as feature engineering and tuning the classifiers.

6.2.1 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: Injecting the faults real-time from GCS, and editing the controller onboard so that the sent faulty input values configures the servos as manipulated from the GCS.

For the former part, a slider is added to the ground station to set the fault during flight and set it back to normal flight conditions if necessary. Fig. 6-6 shows the GCS view with the fault settings open. This pane can be found under *Settings* > *FAULT* as highlighted in pink in Fig. 6-6. The four row configuration represents from top to bottom, the multiplicative error in the right elevon, the multiplicative error in the left elevon, additive error in the right elevon, and finally the additive error in the left elevon. The nominal condition where there is no fault, the configuration is $[right \quad left \quad right\_offset \quad left\_offset] = [1.0 \quad 1.0 \quad 0 \quad 0]$.
This configuration lets the user to realize all types of actuator faults, such as control surface inefficiency or stuck. To generate the fault of right elevon stuck at its nominal position, setting the first slider ($right$) to zero is enough. The generate stuck fault at other positions, $right\_offset$ slider should be changed to desired stuck position while keeping the first slider at zero.

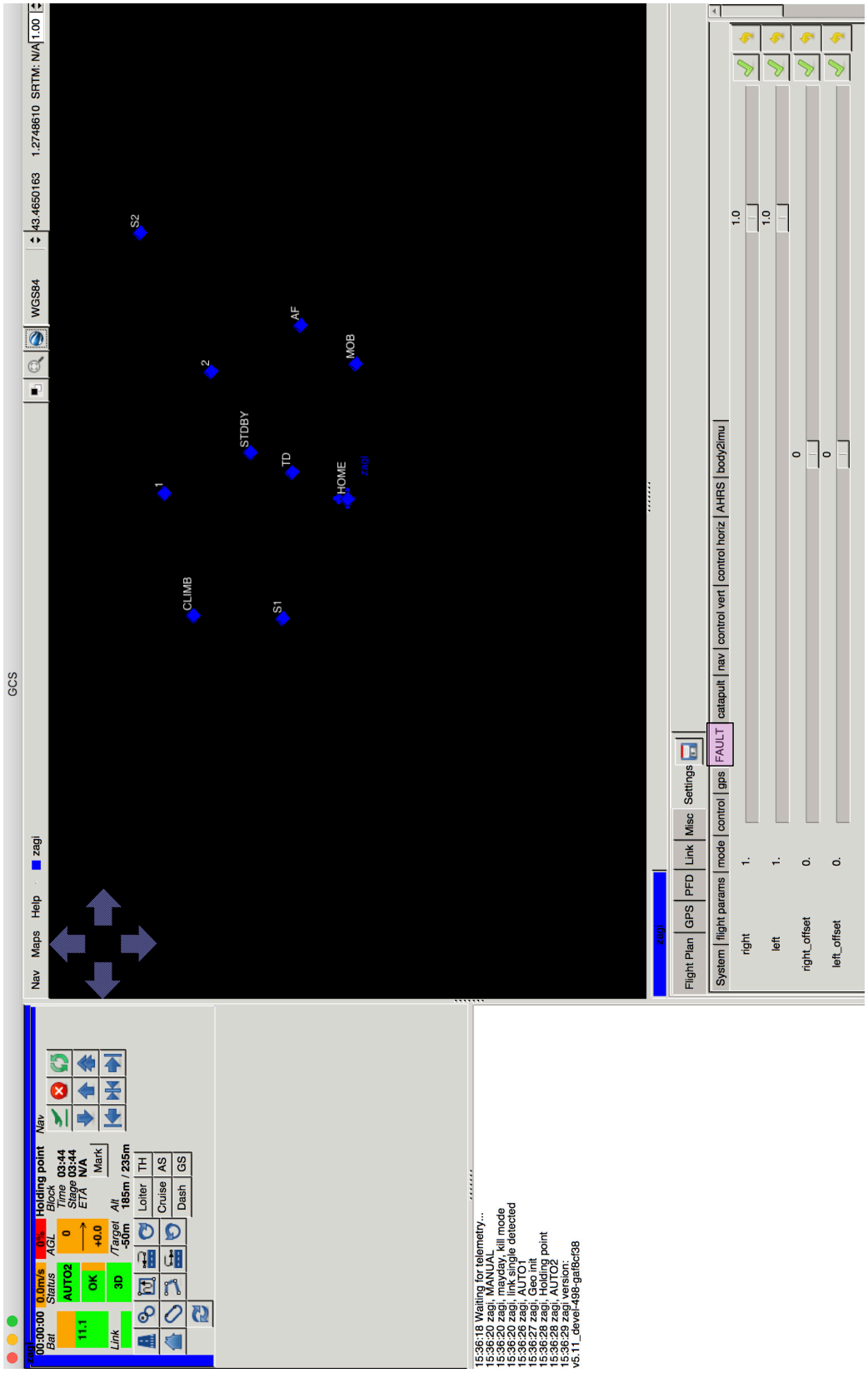

Figure 6-6: View of fault injection tool in Paparazzi ground control station view of in during flight

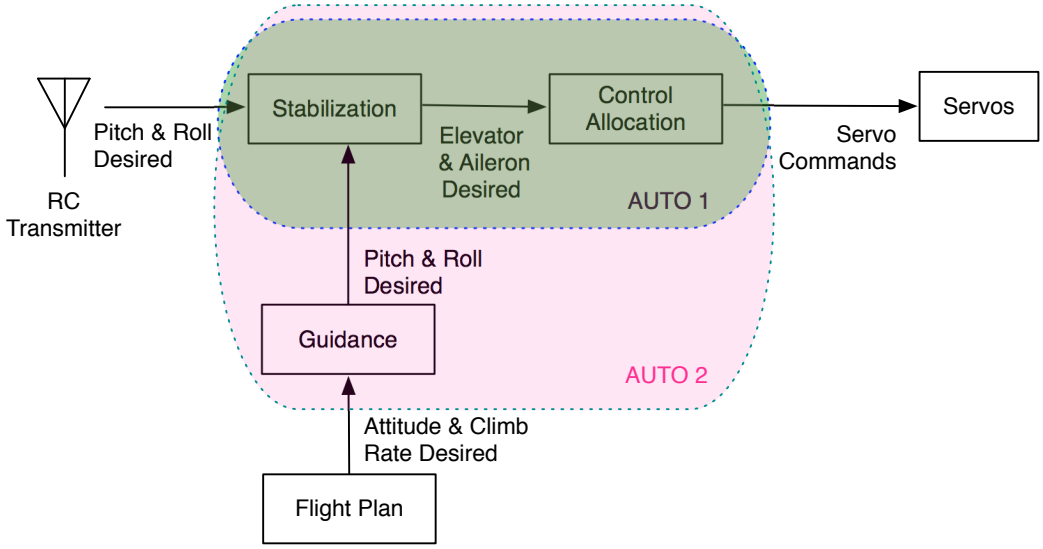

Figure 6-7: Paparazzi autonomy modes

6.2.2 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, a look at the *Paparazzi* flight modes is necessary. Most of the times, there are three modes for fixed-wings from control perspective: *Auto 1*, *Auto 2*, *Manual*. In *Auto 1*, the pilot is still in the loop and gives the desired pitch and roll values to the controller and the desired elevator and aileron commands are calculated by the autopilot and passed to control allocation where the final desired servo commands are sent to servos as highlighted *Auto 1* in Fig. 6-7. In the *Auto 2* mode, there is no need for the pilot since the navigation is also held by the autopilot for a given flight plan. This mode is also given as *Auto 2* in Fig. 6-7. In *Manual* mode, the pilot gives the desired elevator and aileron commands and desired servo commands are calculated in the autopilot's control allocation phase. So still there is a very low sense of autonomy in the manual phase.

Flying with faults is a challenge. The risk to crash is increased on purpose, so a back up plan is necessary to recover from faulty situations if the drone seems to be out of control and/or about to crash. For that purpose, the faults are only injected to

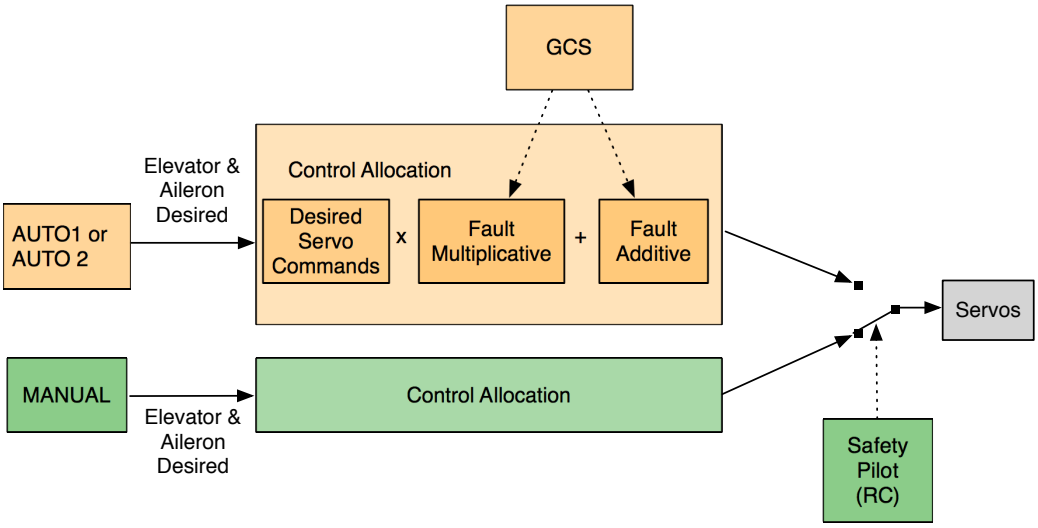

Figure 6-8: Modifications on the control modes of *Paparazzi* autopilot

Auto modes and *Manual* mode is always free of faults even a fault is given from the ground station. So, when the pilot sees a safety problem during the faulty operation, s/he can switch to *Manual* mode from the remote controller and have the control of the control surfaces free from faults. Depending on the nature of the fault injected, such as for some severe stuck control surface fault data gatherings, this might be a game changer since the drone would crash unless an action taken. This is shown in Fig. 6-8 with a switch initiated by the pilot's remote control. This figure shows that during the control allocation phase, which is the calculation of the desired servo commands from given desired elevator/aileron commands, faults are injected if the mode is *Auto 1* and *Auto 2* when fault multiplicative or fault additive values are changed from the GCS by the operator. When switched to *Manual* mode, the control allocation do not consider the injected faults.

The modifications in the control allocation code by adding additive and multiplicative faults as well as extracting *Manual* mode from those injected faults can be seen in the *Paparazzi* code given below.

```
<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>
```

6.2.3 Reading & labeling flight data

Flight data saved to the SD card onboard should be converted to .data format by using the *sd2log* program of *Paparazzi*. For that purpose, from the terminal, browse into the folder including the .LOG file, which is the file format for the onboard data saved in Paparazzi. Run the program under paparazzi/sw/logalize/sd2log and specify the file is to be converted to .data file and where to export it as shown in Fig. 6-9 ('.' means extract to current folder).

An example of a part of the flight data[2] is given in Fig. 6-10 and 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. 6-11. The duration of the flight was around an hour. The flight has been practiced under strong wind. 34 different faults are injected. During the flight, the effect of the faults on the drone was sometimes visible to human eye and sometimes not. For the control surface stuck faults, even for one control surface

[2] 17_07_06__10_21_07_SD.data

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

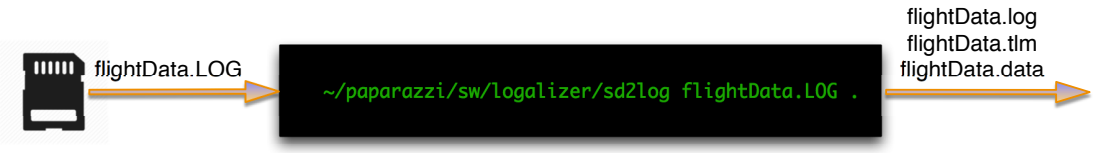

Figure 6-9: Conversion of raw flight data saved to SD card onboard to .data file to be used in further calculations

was stuck case, it immediately gets out of control and safety pilot takes the initiative. Thanks to the piloting skills, no crashes occurred. For ineffectiveness of control surface faults, where the controller has still an effect but not as efficient as before, error in navigation was observed.

Now that the .data file is ready, next is to detect the time stamps at which the faults are injected and then label all the data corresponding to this fault interval as designated with different colors in Fig. 6-10. Since, most of the times, there has been more than one fault type generated during flights, another step in data manipulation is to choose the faults to investigate.

The fault injection or change from fault condition to nominal mode is done via the GCS and there is a corresponding message saved to 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 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. 6-12. It means to multiply the value given by the controller by 1 and add 0, so does not change the values given by the autopilot. As soon as any of control surface effectiveness values is changed in the GCS, a new *Settings* value is saved to the file (Shown with arrows in Fig. 6-10).

To find the indexes where the nominal and faulty data starts and ends, the values of the *Settings* message is investigated. So when there is a *Settings* message in the flight, it should be either a fault generation or going back to nominal condition after a fault. If the message contains the *Settings* message equal to [1.0 1.0 0.0 0.0], this data

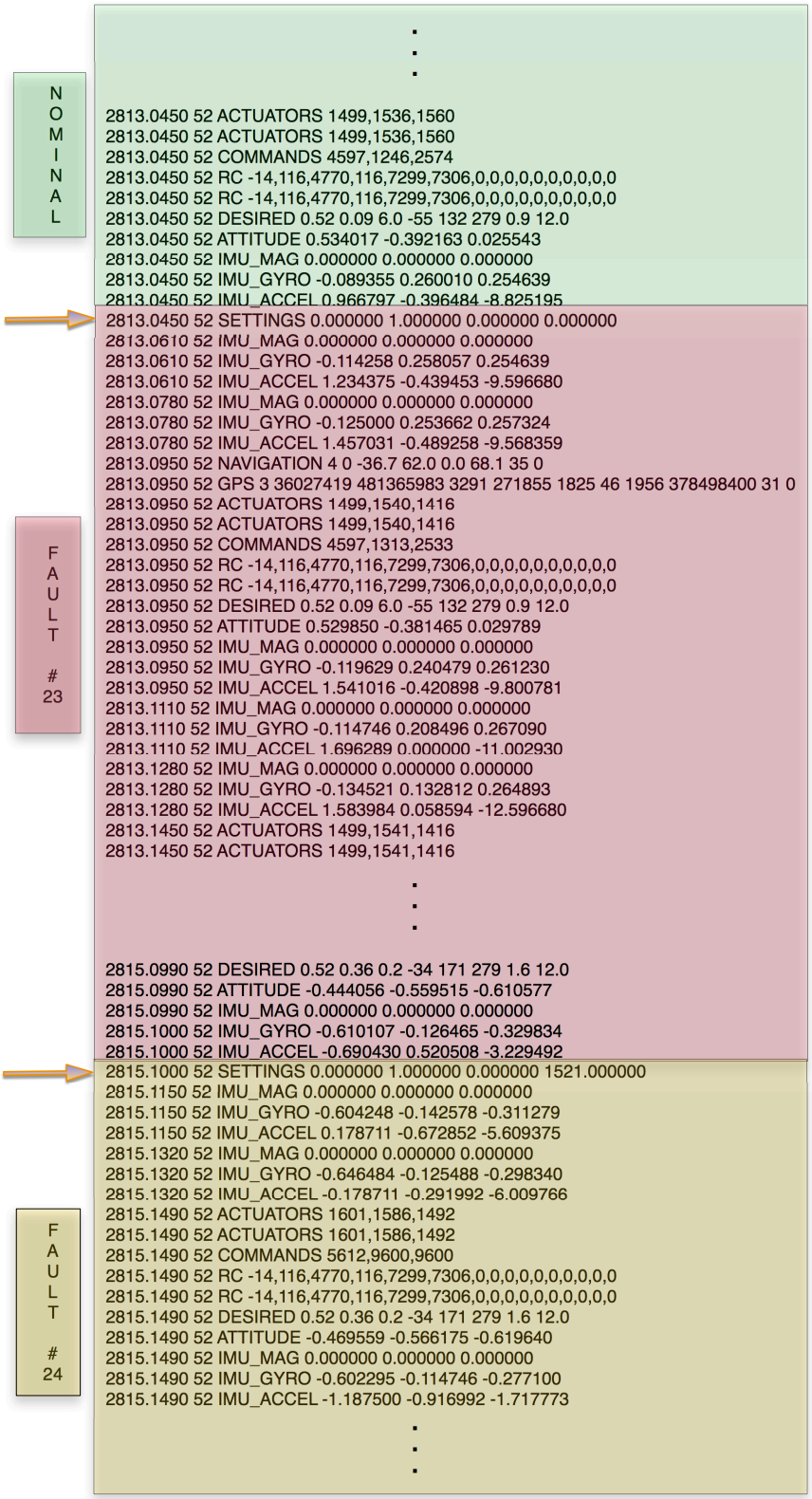

Figure 6-10: A small part of the flight data corresponding to nominal and two different fault phases of the flight. *Settings* message exists only if there is a change in the multiplicative and additive fault parameters via GCS

Figure 6-11: The flying-wing: *ZAGI*

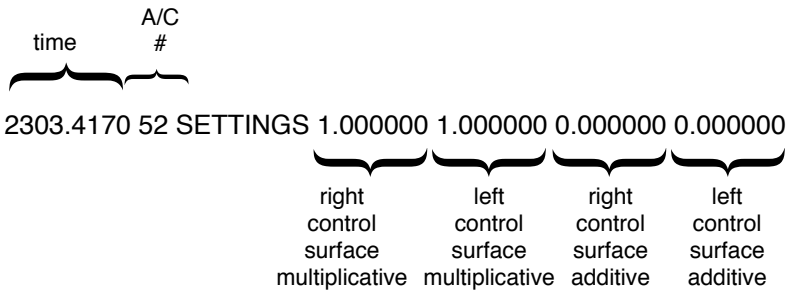

Figure 6-12: *SETTINGS Message* saves the multiplicative and additive fault values inserted from the GCS. [1.0 1.0 0.0 0.0] corresponds a command from the GCS to revert back to nominal phase

index is selected as the nominal condition start index and a previous index before next *Settings* message is the last index of this nominal phase. The matrix holding the start and end index for the nominal phases are saved as *nominal_start_stop* shown in Table 6.1 where the first row corresponds to start index of each nominal set and second row corresponds to the last index of the corresponding as nominal phase. For the fault indexes a similar approach is followed except that *Settings* messages selected are the ones which is different than [1.0 1.0 0.0 0.0]. An example is Fault #23 in Fig. 6-13 and its corresponding start and end indexes given in Table 6.2.

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 and followed with an extreme fault

Table 6.1: Nominal phase start stop indexes of the flight

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

Table 6.2: Faulty phase start stop indexes of the flight

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

2813.0450 52 SETTINGS 0.000000 1.000000 0.000000 0.000000

Figure 6-13: SETTINGS message corresponding to stuck of right control surface

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 6-14: Indexing SETTINGS, to find fault and nominal flight intervals, indexing GYRO measurements to AND with FAULT indexes to find the indexes of faulty gyro measurements

injection is selected. This corresponds to the 4th nominal phase of the flight followed by the 23th fault is selected. Fault #23 corresponds to right control surface stuck and can be seen in Fig. 6-10. The next is to select the measurement corresponding the these selected time intervals and this is done by AND ing the indexes of interest, such as the indexes of fault and gyro data measurements as shown in Fig. 6-14. The script for the 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 selected phases of the flight.

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

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

This work investigates only binary classification, not multi-class classification considering multiple different faults. But different faulty cases have been studied to further attack the problem of multi-class classification in the future studies. The classifier's abilities in different fault conditions have been investigated. The main two types of faults of interest in this study were control surface stuck and LOE. The flight data[4] and code[5] that has been developed are publicly available under *Github*.

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

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

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

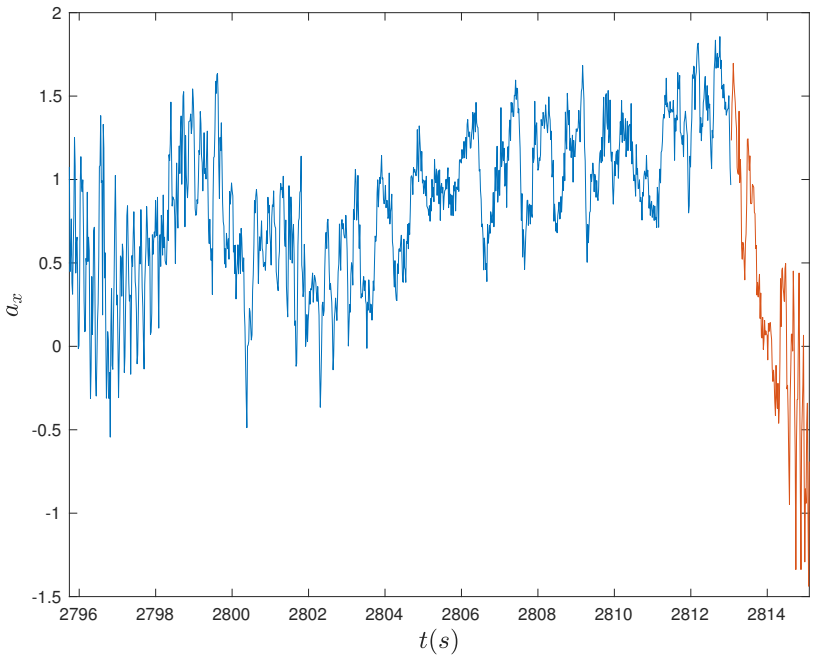

Figure 6-15: Accelerometer readings along x-direction during flight interval just before control surface stuck (nominal, represented with blue) and during control surface stuck at 0° until the safety pilot's intervention

6.2.4 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 manual recovery by triggering the safety switch shown in Fig. 6-8. So the time past starting from the ignition of the control surface stuck fault until the manual pilot's intervention is very short ($\sim$2 seconds) which can be seen from the accelerometer x-direction readings in Fig. 6-15. This causes the problem of skew-class classification where there is a big difference between the number of instances belonging to different classes and requires some special treatment as explained above under section 5.2.2.

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

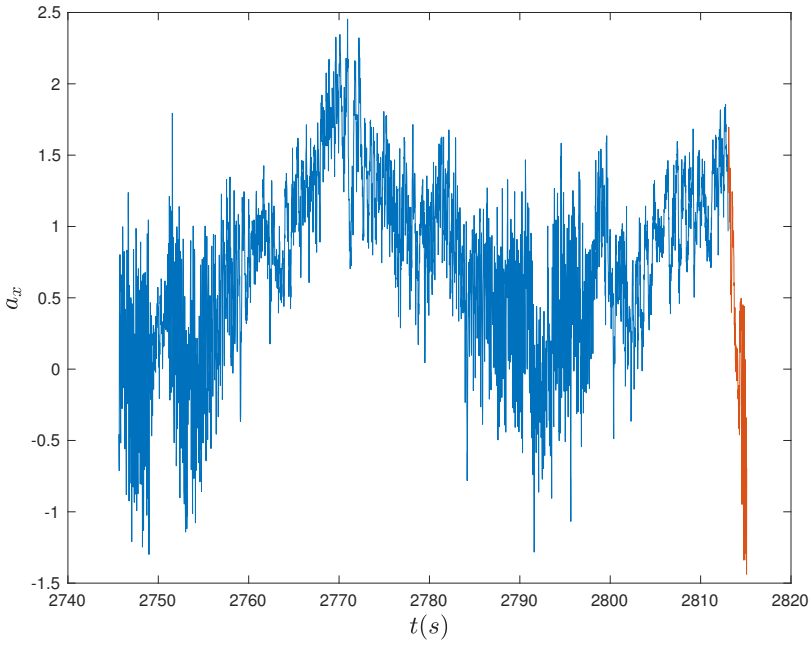

Figure 6-16: Accelerometer readings along x-direction during flight interval before control surface stuck (nominal, represented with blue) and during control surface stuck at 0° until the safety pilot's intervention

Having knowledge about data is critical for machine learning applications. Usually, the performance of the learning algorithms are quite dependent on the level of the engineer's experience since some of the critical decision are handled by her/him such as selection of features (not for all machine learning methods). Fig. 6-15 shows accelerometer x-axis measurements for a duration of ∼20 seconds. Measurements plotted in blue indicates the part of the flight where elevon works correctly while red part of the plot corresponds to the faulty phase. It can be interpreted from the Fig. 6-15 that the fault injected shows a distinct change in the x-axis accelerometer measurements when the fault is injected.

To investigate further, measurements have been plotted during a larger time scale involving ∼70 seconds of measurements as shown in Fig. 6-16. Here, it is seen that the accelerometer measurements corresponding to faulty phase could be observed during the nominal flight phase as well, thus it might be necessary to check the time change of translational acceleration to catch a difference in behavior. Fig. 6-17 and Fig. 6-18 show the accelerometer measurements along z-direction and angular

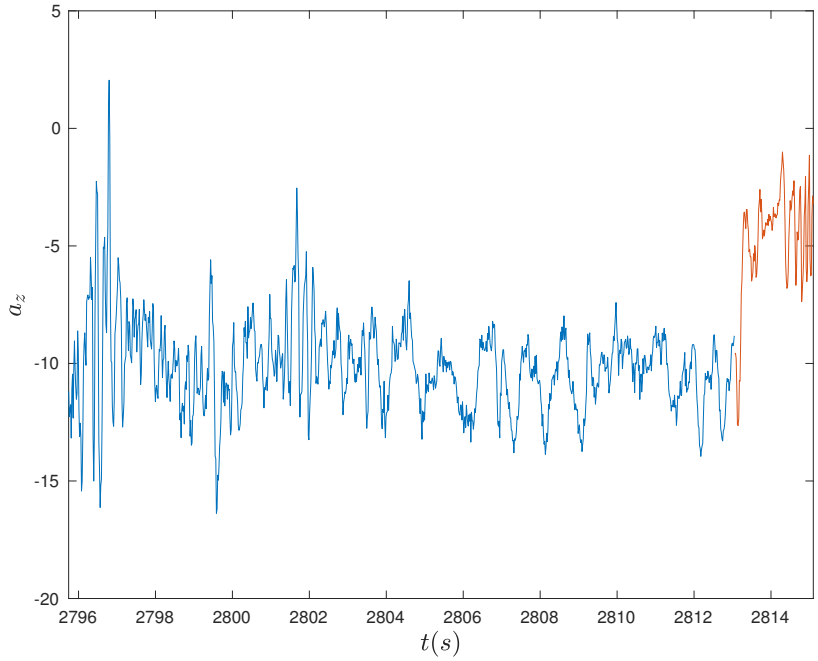

Figure 6-17: Accelerometer measurements along z direction a_z for faulty and nominal flight data

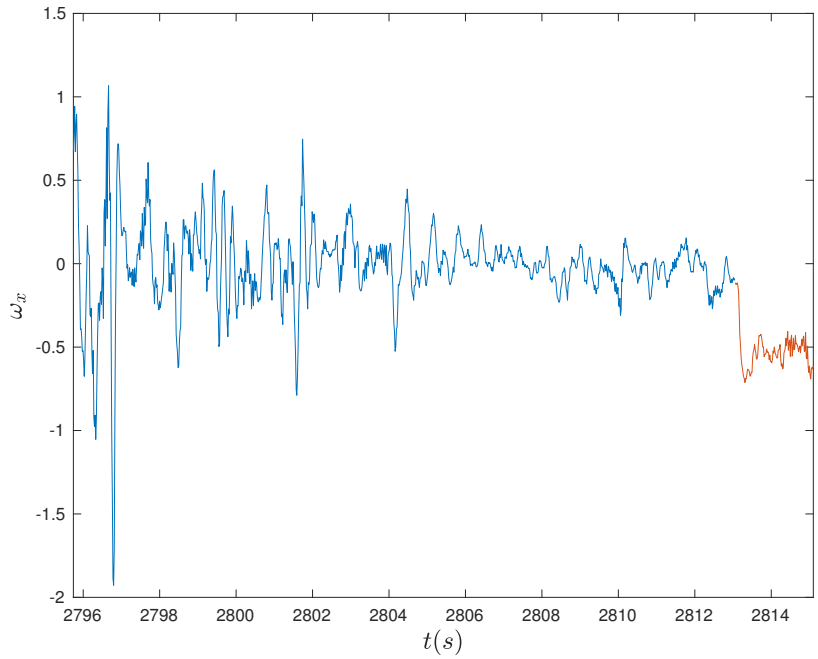

Figure 6-18: Angular velocity w_x for faulty nominal flight data

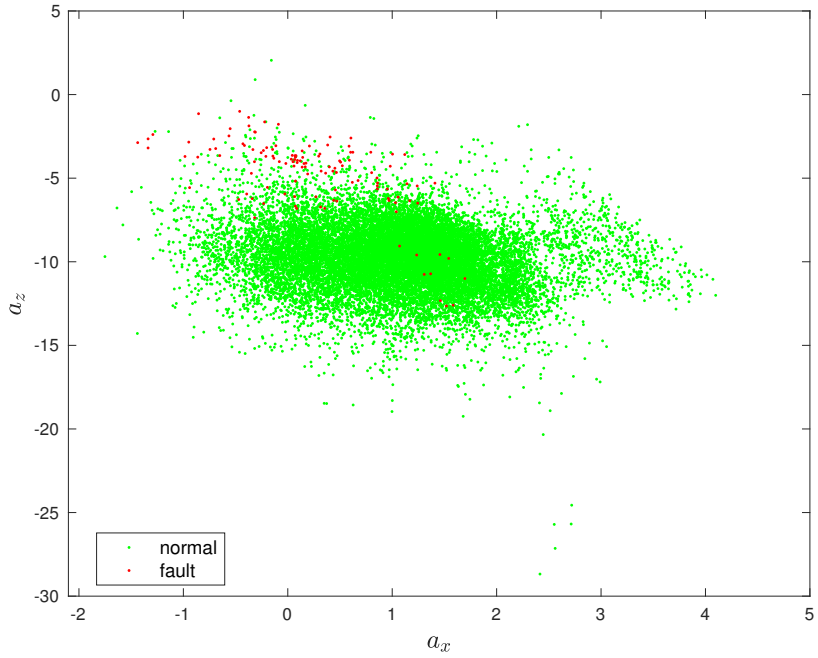

Figure 6-19: a_x vs a_z feature space for faulty nominal flight data

velocities along x-direction during a short time interval. Here, although an average of faulty measurements would result in an obvious difference, individual measures might correspond to nominal phase of the flight as well.

The measurements are also plotted in feature spaces a_x-a_z and a_x-a_y as shown in Fig. 6-19 and Fig. 6-20. It shows a_x-a_z space gives a distribution that would be easier to classify the faulty and nominal phases while in a_x-a_y feature space, the measures are quite similar and difficult to classify.

Table 6.3 shows the results of SVM classification for untuned linear kernel, untuned Gaussian kernel, and tuned Gaussian kernel via two methods (heuristic and Bayesian) respectively in its columns. Although a variety of variables used in the evaluation of classifiers have been presented to the reader for completeness, f1Score will be the main variable of concern for this study for the reasons explained before. The classifier with a *box constraint* $= 2.11$ and a *kernel scale* $= 1$ have been found to give the highest *f1Score* (*f1Score* $= 0.9545$).

Due to previous observations on data, feature set have been widened via additions of data from past measurements for each attribute. The idea is to introduce the time

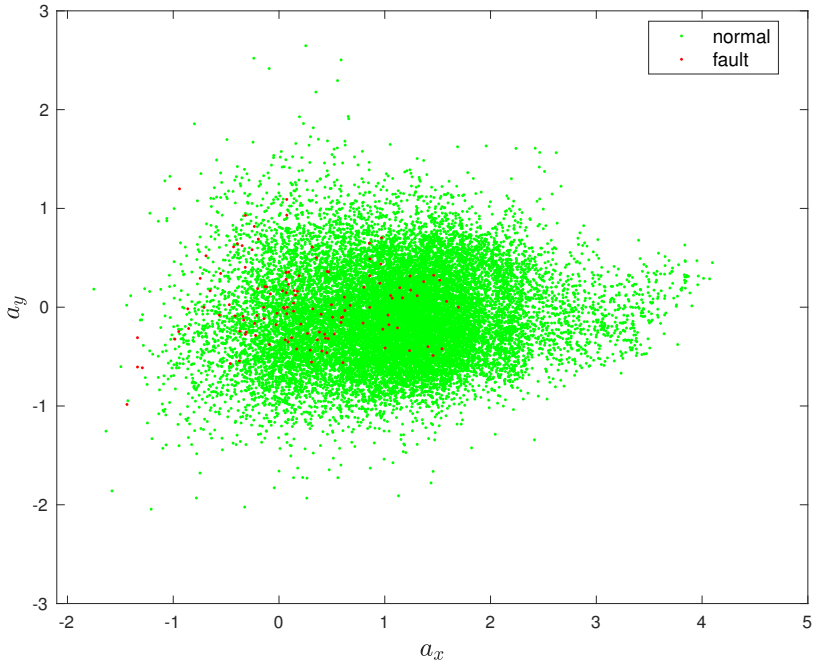

Figure 6-20: a_x vs a_y feature space for faulty nominal flight data

Table 6.3: 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

Table 6.4: 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

change behavior of data to be represented in each instance. The number of features have been increased to 24 in Table 6.4, which means only 3 previous measures for each different attribute have been added to the feature set ((3 past + 1 instant) * 6 axis). This addition to the feature set, in general, deteriorated the performance of the classifier, especially the classifier with an untuned Gaussian kernel (*f1Score*= 0.4167) ($\Delta f1Score = -0.5135$). For the classifiers with tuned Gaussian kernels, *f1Score* decreases by a small amount for the classifier tuned with heuristic method ($f1Score = 0.9444$) and increases by an even smaller amount($f1Score = 0.9143$) hence the effect is not really obvious.

Table 6.4 and first 3 columns of Table 6.5 are the results of the same algorithm with same number of features and percentage of training/test set ratio. Due to usage of random functions, the seed value has been set to a constant value for reproducibility. Stratified sampling has been implemented to assure fair amount of data from both classes in both sets (training and test). Finally, it has been discovered that even with the selection of same number of data for the training and test sets for both classes (faulty and nominal), some combinations of possible training/test set selections result in more precise classifiers than others. This can be seen by observing the tuned *f1Scores* (*f1Scores*=0.9444 in and *f1Score*= 0.9143) in Table 6.4 increasing to the *f1Score* (*f1Score*= 1) in Table 6.5.

Table 6.5: 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 6.5 shows the evaluation results of SVM classifiers trained with different percentage of training and test sets. First 3 columns present the results for a test to training set ratio of 1/4 while the last 3 columns present the ratio of 2/3. *f1Score* trained and tuned with %80 training data implies that this percentage is a good choice for this problem (with an *f1Score* of 1 for tuned classifiers).

6.2.5 Control surface loss of efficiency fault

Loss of efficiency fault was generally more difficult to diagnose. First, a fault of %10 loss of efficiency on the left elevon has been investigated (SETTINGS 1.0 0.9 0.0 0.0). This fault was the first injected fault in the course of the flight so the corresponding nominal phase is quite long. During those first minutes of the flight, the nominal mode kept quite long to gather more nominal data from the flight before initiating the fault sequences. The results showed the insufficiency of the method to diagnose the fault (see Table 6.6 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, since the control surfaces is not moving in a wide range during the nominal course of action, %10 degradation in the movement of the elevon does not really change the dynamic behavior of the drone yet not easy to pinpoint from the measurements. Another reason, likely to be the main driver, could be the compensation of the fault via the controller. The aim of the controller onboard is to minimize the error to track a given reference trajectory and corresponding attitude references. Hence, when this error is not minimized, for one reason or another, the controller re-calculates the control signal to minimize this error. So either an error due to a different wind condition or a fault in the control surface, the controller's duty is to minimize this error, so a well designed controller might handle that especially for the condition where the effect of disturbance is not catastrophic. This controller's compensation for the fault could be known by observing output of the controller and feeding this information as well in the set of instances. But, during the course of this study, the abilities of a classifier without the information from the controller is investigated. The main reason behind is to challenge the abilities of a small and

Table 6.6: %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

Table 6.7: %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

cheap set of health monitoring system that does not rely on the information from the autopilot.

The loss of efficiency of the left elevon is increased to %40 degradation to investigate if this time the SVM classifier will be able to classify the fault. This fault is coined as LOE2 in the Table 6.6. The results show no recuperation in the classification results for neither untuned nor tuned classifiers with *f1Score*= 0.0032 and *f1Score*= 0.2 respectively.

Since the results were very poor in terms of classifying faults, the faults have been increased to have %40 degradation in both elevon. The idea is to see at which level the result of classification will improve and also to search for any reasons behind this ineffectiveness except controller's compensation. Another issue to investigate was the effect of number of instances for the larger datasets (nominal class) to classification. For that purpose, three different size for nominal data set (∼15min, ∼6min, ∼3min)

Table 6.8: %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

was investigated in terms of their effects to classification performance for a faulty measurement data set of ∼1min. Classification results for two of the situations (∼15min, ∼3min) are given in Table 6.7. The reduction of the nominal data set was not random selection throughout the complete set, but only the last part of the nominal data set that is closer to the faulty data in terms of flight time instance was selected. The first simulation involved ∼15min nominal measurements, hence the most skewed classification. An untuned SVM classifier with a Gaussian kernel was not able to classify ($f1Score = NaN$) the fault. Even when tuned, the classification performance was too poor to be used for predicting faults ($f1Score = 0.12$) as shown in Table 6.7. Reducing the number of the larger class, gyro and accelerometer measurements during nominal phase, an untuned SVM classifier with Gaussian kernel still results poorly in classification ($f1Score = 0.0073$) and tuning does not help much ($f1Score = 0.13$). Compared to the larger nominal class measurements case, it gives the idea that reducing the number of instances of the bigger class might help to improve classification performance ($\Delta f1Score = 0.01$). A further reduction in the 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 observed compared to the wider nominal data set ($\Delta f1Score = 0.2616$). Tuning the classifier did not enhance the classification result noticeably ($f1Score = 0.2756$).

Since reducing the number of instances of the larger class (nominal phase measurements) in the classification does help but not result in satisfactory results for

classification, adding new attributes to the feature set is considered. For that, the previous measurements of the same attribute is added to the input matrix to have the time dependent change in the behavior of the observed physical variable. Results for a set of total number of 24, 100 and 300 features has been given in Table 6.8. To start with the discussion of the effect of adding previous measurements as separate features to the feature set, previous 3 measurements have been added to the input data set, resulting in 4 x 6 = 24 features. Adding 3 previous instance in time for each sensor measurement decreased the performance of the untuned classifier with a Gaussian kernel $f1Score = 0.1375$ which is an classification performance decrease of $\Delta f1Score = 0.1341$. Adding more features the classifier still not able to classify when untuned but by tuning the *f1Score* can be improved heavily producing a classifier with an *f1Score* of 0.9635 for 120 feature case and can be even more fined to a *f1Score* of 0.9896 for 300 features in total. So adding features advances the classification but only with proper tuning is achieved. Otherwise, untuned, the results are even worse than before adding features.

6.2.6 Use of spinors as attributes

In this study, until now, use of feature engineering to ameliorate the classification performance has been performed mainly utilizing the addition of new features to include previous measurements in all instances. The idea 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. For that, the kinematics equation, presented here again as a reminder, has to be solved numerically to attain quaternions from angular velocities.

For that, kinematics equations Eq. 6.3 have been solved numerically.

$$\begin{aligned}\dot{q}_0 &= -\frac{1}{2}\boldsymbol{q}_\nu^T\boldsymbol{\omega} \\ \dot{\boldsymbol{q}}_\nu &= \frac{1}{2}\Big(\boldsymbol{q}_\nu^\times + q_0\boldsymbol{I}_3\Big)\boldsymbol{\omega}\end{aligned} \tag{6.3}$$

$$\boldsymbol{q} = \begin{bmatrix} cos\Big(\frac{\theta}{2}\Big) & sin(\frac{\theta}{2})\boldsymbol{e}\end{bmatrix} \tag{6.4}$$

Given the quaternion definition in Eq. 6.4

$$log(\boldsymbol{q}) = \begin{bmatrix} 0 & \frac{\theta}{2}\boldsymbol{e}\end{bmatrix} \tag{6.5}$$

spinor are calculated with taking the logarithm of quaternion described as in Eq. 6.5.

Table 6.9: 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

Result given in Table 6.9 shows an apparent increase in classification with *f1Score* reaching up to 0.9915.

Untuned classification superiority

The first promising result by using spinors for classification is its advantage in untuned classifiers. This can be seen by checking from Table 6.10 the *f1Score* of the SVM classifier without tuning. As mentioned before, untuned classifier for added features results poorly although increasing the classification accuracy when tuned properly. 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. 6.6, *f1Score* even increases up to 0.9555.

$$K(x,z) = exp\left(- \frac{\|x-z\|^2}{2\sigma^2} \right) \tag{6.6}$$

Table 6.10: 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

Tuned classification Bayesian optimization efficiency

Table 6.11 shows the results except 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. But when spinors used, Bayesian optimization results in a better performance in tuning the classifier as shown under Table 6.11 in Tuned Gaussian Kernel spinors row.

Table 6.11: 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

Chapter 7

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.

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

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%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] Plamen Angelov. *Sense and avoid in UAS: research and applications*. John Wiley & Sons, 2012.

[3] Daniel Asimov. The grand tour: a tool for viewing multidimensional data. *SIAM journal on scientific and statistical computing*, 6(1):128–143, 1985.

[4] Thomas Bak. Spacecraft attitude determination. 1999.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Dianne Cook and Andreas Buja. Manual controls for high-dimensional data projections. *Journal of computational and Graphical Statistics*, 6(4):464–480, 1997.

[12] 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.

[13] Dianne Cook and Deborah F Swayne. *Interactive and dynamic graphics for data analysis: with R and GGobi*. Springer Science & Business Media, 2007.

[14] 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.

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

[16] Introduction of a regulatory framework for the operation of drones, 2015.

[17] Heinz Erzberger and Russel A Paielli. Concept for next generation air traffic control system. *Air Traffic Control Quarterly*, 10(4):355–378, 2002.

[18] 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.

[19] Ulrike Esther Franke. Civilian drones: Fixing an image problem? `http://isnblog.ethz.ch/security/civilian-drones-fixing-an-image-problem`, 2015.

[20] 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.

[21] Wei-hua Gui and Xiao-ying Liu. Fault diagnosis technologies based on artificial intelligence for complex process. *Basic Automation*, 4:000, 2002.

[22] Steve R Gunn et al. Support vector machines for classification and regression. *ISIS technical report*, 14:85–86, 1998.

[23] 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.

[24] 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.

[25] ICAO Cir 328, Unmanned Aircraft Systems (UAS), 2011.

[26] 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.

[27] 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.

[28] 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.

[29] Sherri Magyarits. FAA,s Approach to Unmanned Aircraft Systems (UAS) Concept Maturation, Validation & Requirements Development, 2015.

[30] 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.

[31] 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.

[32] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[33] 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.

[34] Laurent Probst, Laurent Frideres, Bertrand Pedersen, and Christian Martinez-Diaz. Uav systems for civilian applications. `http://ec.europa.eu/DocsRoom/documents/13430`, 2015.

[35] RECONFIGURE FP7 Project. `reconfigure.deimos-space.com/`, Accessed: 2016-07-19.

[36] Unmanned Aerial Vehicle Reliability Study, 2003.

[37] Unmanned Systems Roadmap 2005 - 2030, 2005.

[38] 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.

[39] 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.

[40] John Stuelpnagel. On the parametrization of the three-dimensional rotation group. *SIAM review*, 6(4):422–430, 1964.

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

[42] 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.

[43] 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.

[44] Bong Wie. *Space vehicle dynamics and control*. Aiaa, 1998.

[45] 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.

[46] Youmin Zhang and Jin Jiang. Bibliographical review on reconfigurable fault-tolerant control systems. *Annual reviews in control*, 32(2):229–252, 2008.