ACTUATOR FAULT DETECTION, ISOLATION AND FAULT TOLERANT CONTROL OF A HEXACOPTER UAV

Thesis submitted in partial fulfillment
of the requirements for the degree of

*Master of Science in **Electronics and Communication Engineering** by Research*

by

Aditya Mulgundkar
2020900037
aditya.mulgundkar@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
November 2023

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "ACTUATOR FAULT DETECTION, ISOLATION AND FAULT TOLERANT CONTROL OF A HEXACOPTER UAV" by Aditya Mulgundkar, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. Harikumar Kandath

To my family, my friends, and my labmates in the Robotics Research Centre.

Acknowledgments

I would like to thank my advisor, Dr. Harikumar Kandath, for his constant support and guidance in my research.

I am thankful to Dr. Deepak Gangadharan for his mentorship and guidance. I also thank the professors working in the Robotics Research Center (RRC) - Prof. Madhava Krishna, Dr. Spandan Roy, and Dr. Nagamanikandan Govindan.

Alongside my friends and family, I express my profound gratitude to Prof. Kashyap Joshi for kindling my interest in robotics. His inspirational guidance has been instrumental in my academic journey, a debt I will forever appreciate.

This work was supported by TiHAN, IIT Hyderabad under the project number TiHAN-IITH/03/2021-22/33(4).

Abstract

Unmanned Aerial Vehicles (UAVs) continue to penetrate diverse sectors, including agriculture, disaster management, communication, transportation, and defense. The robustness and reliability of their operation have become paramount due to the increasing ubiquity of these systems across numerous sectors. UAVs can undergo faults in their sensors, actuators (motors), and even structure - all of these have different levels of severity and effects on the system. Motor faults can be of three main types, namely, Motor Locking, Loss of Efficiency (LoE), and Loss of Actuation (LoA). Motor locking is a case where the actuator keeps spinning at a given RPM and cannot be changed, whereas LoE is a case where the actuator degrades over time and is not able to generate enough torque due to lower RPM. In this thesis, however, we focus on LoA, where an actuator stops working completely (gets stuck at zero RPM) and can result in catastrophic failure in the UAV. This work delves into the dynamics of UAVs under actuator fault conditions, examines the resulting instability issues, and explores potential methods for identification and control reconfiguration for safe operation.

Once an actuator failure occurs, the Fault Detection & Isolation (FDI) module should be able to locate the fault and pinpoint it with high accuracy and in a very short time, to allow the UAV to stabilize the fault. Various approaches can be used for the FDI system, namely, rule-based, model-based, and data-driven. Implementation of these methods can be relying on RPM sensors, battery monitors, and software-only solutions. We propose a data-driven software module for this purpose, since it is UAV frame agnostic, does not require any additional hardware (software-only), and can be run with minimal setup. The proposed Rotation Forest based classifier can detect, classify, and report the fault within 120-250 milliseconds of occurrence of the fault. This is an acceptable delay, since we have observed that the vehicle cannot handle delays upwards of 500 milliseconds in simulations.

Once a fault is reported by the FDI module, the Fault Tolerant Control (FTC) module reconfigures the control system to stabilize the vehicle and continue the mission, or prevent a crash by peforming a safe landing. This thesis focuses on cases of complete actuator failure in Hexacopter UAVs, specifically, for single motor failure scenarios. The proposed FTC module performs UAV stabilization using control reconfiguration, after fault occurrence.

We perform a thorough analysis for the FDI and FTC modules separately, extensively in simulation, and demonstrate the same in real flight tests. We also combine the two modules and analyze the response in the simulation. In real flights, the classifier (FDI module) responds to the fault in 2-5 sensor data samples (at a 60ms rate per sample) and has a high true positive rate of 92.6%. Also, in real flights,

the performance of the FTC module is measured in terms of tracking error, percentage overshoot, and settling time.

Integration of the FDI and FTC modules is performed and simulation results are presented showing satisfactory operation of each of these modules with guaranteed stable flight.

Contents

List of Figures

List of Tables

List of Acronyms

CNN	Convolutional Neural Network
DCR	Dynamic Control Reallocation
DT	Decision Trees
FDI	Fault Identification & Isolation
FDI	Fault Tolerant Control
FT	Fault Tolerance
GCS	Ground Control Station
GNB	Gaussian Naive Bayes
GPS	Global Positioning System
LoA	Loss of Actuation
LoE	Loss of Efficiency
LSTM	Long Short Time Memory
PCA	Principal Component Analysis
PID	Proportional–Integral–Derivative
RF	Rotation Forest
RPM	Revolutions per minute
SVM	Support Vector Machines
UAV	Unmanned Aerial Vehicle
WCA	Weighted Control Allocation

Chapter 1

Introduction

As Unmanned Aerial Vehicles (UAVs) continue to become more prevalent, the robustness and reliability of their operation have become paramount. An integral element of UAV reliability is fault tolerance, the ability of these systems to continue the mission even when subjected to various types of faults or failures happening to its software or hardware components. Fault tolerance in UAVs is not just a matter of service quality, but also of safety and sustainability. An in-flight fault can lead to uncontrolled UAV behavior, possibly resulting in crashes that could damage property, harm individuals, depending on the UAV's size and operational environment. This is particularly relevant when considering the variety of faults UAVs can encounter, including mechanical faults such as propeller or motor failure, sensor faults such as GPS or altimeter malfunctions, or software faults in control algorithms or onboard firmware. Furthermore, UAV failures can cause environmental damage, as crashes can spread debris or leak potentially hazardous materials. In some cases, the unpredictability of a faulty UAV can also compromise missions, leading to financial losses and operational inefficiencies. Thus, the issue of fault tolerance in UAVs is not just a technical problem, but a complex, multidimensional challenge that calls for rigorous research and innovative solutions.

This work specifically focuses on the challenges posed by actuator or motor faults in a multirotor UAV, which are common but potentially disastrous problems in UAV operations. Actuator faults generally manifest in three main forms: loss of efficiency, which results in decreased output; loss of actuation, where the motor stops completely; and motor locking, where the motor becomes stuck at a constant output. Among these, we focus primarily on the scenario of a complete loss of actuation. In such cases, the UAV is left with a critical deficiency in its control capabilities, making maintenance of stability a significant challenge and can lead to severe imbalance and a possible catastrophic failure if not properly managed. This work delves into the dynamics of UAVs under such critical fault conditions, examines the resulting instability issues, and explores potential methods for control and mitigation. Through a detailed understanding of these dynamics, this research aims to contribute to the development of more resilient, reliable, and fault-tolerant UAV systems.

Figure 1.1: A failure scenario occurring in a drone delivery mission can cause the drone to flip over and burn. Source: https://www.youtube.com/watch?v=bnC6pfwtVvI

This work focuses on the dynamics and implications of complete loss of single actuator in Hexacopter UAVs. When a UAV suffers from such a failure, it results in an instantaneous drop of control inputs to zero for the affected actuator, resulting in a sudden shift in the system dynamics. This loss manifests mathematically as a discontinuity in the system equations, disrupting the UAV's flight trajectory and potentially leading to erratic and uncontrollable movements. The objective is to understand the precise trajectory alterations, changes in the UAV's orientation, and shifts in the system's equilibrium points triggered by a complete actuator failure. In doing so, this work aims to provide a mathematical foundation for developing robust control strategies that can ensure stability and control, even in the face of total actuator loss.

Complete loss of actuation often introduces a sudden and dramatic imbalance in a UAV's control system, leading to significant instability issues. The UAV's flight dynamics are directly tied to the functioning of its actuators - when one or more actuators fail entirely, the symmetry in the control inputs is disrupted. This disruption can result in unpredictable and often amplified changes in the UAV's attitude and velocity, making it challenging to maintain a stable flight path. The UAV may enter a spin or tumble, or even fall from the sky, given the severity of the instability. Such instability can escalate quickly, especially without appropriate compensation strategies in place, resulting in a rapid deterioration of the UAV's operational state and potential catastrophic failure.

The occurrence of complete actuator loss necessitates the application of a dual-layered response system: i) Fault Detection and Isolation (FDI) and ii) Fault Tolerant Control (FTC). FDI performs real-time monitoring to identify and isolate any discrepancies in the system's mathematical models indicative of a fault. Post-fault identification, FTC assumes responsibility, deploying control strategies to counteract the system's imbalance caused by the actuator failure. These strategies are formulated based on the UAV's mathematical models, with an aim to maintain the UAV's operational stability and safety in the face of faults. In summary, FDI and FTC should work simultaneously to mitigate the impact of complete actuator failure and, in doing so, increase the fault tolerance of the UAV. The amount of

time taken from fault occurrence to its detection/classification plays a crucial role in the UAV's ability to stabilize post fault, since a UAV in midflight, with fault, most of the UAVs are typically left with a few milliseconds of time to counter and compensate for the resulting torque [31].

This work details the FDI module - a Rotation Forest Classifier-based technique, an ensemble learning method typically used in machine learning for classification tasks. This technique has been innovatively adapted to effectively detect and isolate actuator faults in UAVs, using its powerful decision-making capabilities to discern between normal and faulty operating conditions. Regarding FTC, a strategy based on Weighted Control Allocation (WCA) is used. This method dynamically reallocates control among the remaining operational actuators to counteract the effects of loss of actuation. Using this approach, the aim is to maintain the stability and controllability of the UAV despite the occurrence of critical faults. Therefore, this work pioneers the application of a Rotation Forest Classifier for FDI and a WCA-based strategy for FTC, significantly advancing the field of fault tolerance in UAVs.

The issue of complete actuator loss in Unmanned Aerial Vehicles (UAVs) is of significant relevance, since the functionality of UAVs depends heavily on their operational reliability. Actuator faults pose a significant threat to this. Complete actuator loss can cause severe instability, potentially leading to catastrophic failures, compromising property, human life, and the environment. Furthermore, these faults can lead to mission failures, leading to substantial financial losses and operational inefficiencies. As UAVs continue to become more integral to our societal infrastructure, addressing the challenge of complete actuator loss becomes increasingly critical. Therefore, this research focusing on robust fault detection and fault-tolerant control mechanisms for UAVs in severe fault conditions is highly relevant and timely.

1.1 Related Works

1.1.1 Fault Detection & Isolation

Existing fault detection and classification approaches focus on obtaining a high confidence value in a short period after the fault occurs to allow the proceeding control algorithm to compensate quickly and avoid a crash. In [30], a complete error detection, fault diagnosis, and system recovery architecture for a coaxial octorotor is presented. The diagnosis module uses motor speeds and currents measured by electronic speed controllers to reconfigure the control allocation matrix and distinguish between motor failures and propeller losses. The fault diagnosis method combines model-based thresholding and model-free classification using a support vector machine (SVM) algorithm. In [26], a combination of dynamic and data-driven models is used to generate training data and a deep neural network known as the long-short-time memory (LSTM) network to estimate the torque and thrust of faulty propellers. Flight datasets under normal and faulty scenarios are generated through simulation using the developed data-generative model. The article also proposes a fault classifier that uses a convolutional neural network (CNN) structure to identify and evaluate the degree of damage to the propellers. Both of these

approaches are infeasible since they require additional hardware in actual test flights. Our approach aims to improve this aspect by providing a software-based model-free solution that does not require additional sensors. Existing Note that our approach is designed for complete motor failures, while those compared before can also classify propeller loss to a reasonable extent. In [26], similar results are shown for propeller damage detection, running on a Random Forest algorithm and reporting an accuracy of 86.57%.

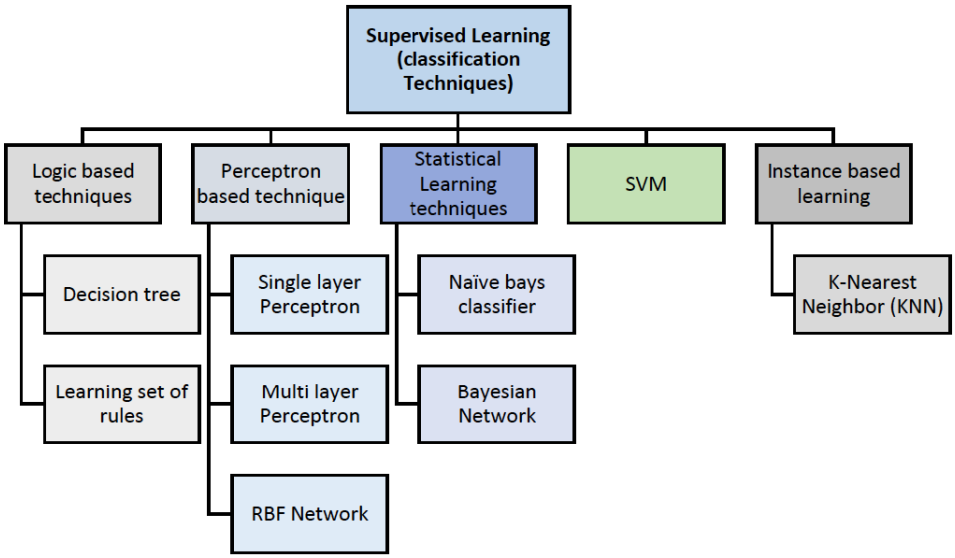

Figure 1.2: Different types of Supervised Learning methods for time-series classification. Source: https://doi.org/10.3390/drones6110330

As shown in Fig. (1.2), various supervised learning methods can be used for the detection [27] and classification [35] of motor failures in a UAV. Since SVMs underperform in large datasets with noise [3], such as the sensor data used as input in our work. Also, CNNs performance is greatly influenced by hyperparameter selection [2], we have taken a tree-based approach. Tree-based models are better in our case, because they can handle interactions between features and provide easy interpretability of the model's decision-making process [33]. This made Decision Trees an interesting choice over existing approaches for our problem statement, over other classifiers [17, 21].

1.1.2 Fault Tolerant Control

Different fault-tolerant architectures exist for quadcopters, hexacopters, and octocopters [18, 20]. Here, we focus on a fault-tolerant architecture for hexacopter in single actuator failure [11, 19, 23]. Fault tolerance can be achieved by modifying the hardware [14, 24] or changing the controller architecture [12, 29]. Work done in [24] and [14] allows for higher torque in all directions, since it uses a tilted rotor configuration. This added torque and the tilting ability of the mechanism allow it to tolerate motor faults. Such a mechanism is difficult to manufacture and does not provide a general solution that can

be easily implemented. Some existing works propose a different motor placement configuration, in order to achieve a higher level of fault tolerance, while sacrificing overall controllability in non-faulty/ general scenarios [18, 25, 38]. This approach is not ideal since fault scenarios are a low probability occurrence. Since the control allocation problem is an optimization problem, there are other alternatives in existing literature [5], such as mixer logic, linear programming and the simplex algorithm. Work done in [12] shows the architecture of an attitude control approach with classical control allocation on a hexacopter. To improve on this method, we have used the Weighted Control Allocation (WCA) method and conducted a study on the performance of the UAV using exhaustive flight trials.

Figure 1.3: A fault tolerant quadrotor built by Researchers at the University of Zurich and the Delft University of Technology. Source: https://robohub.org/how-to-keep-drones-flying-when-a-motor-fails/

In existing UAV fault-tolerant methodologies, several gaps are evident that limit their effectiveness in managing complete actuator loss. Specifically, the utilization of a machine learning-based approach like the Rotation Forest Classifier in the Fault Detection and Isolation (FDI) domain remains largely unexplored. Furthermore, while dynamic Control Allocation strategies have been employed in the Fault Tolerant Control (FTC) domain, their applications for UAVs with complete actuator loss are scarce. This research seeks to bridge these gaps by pioneering the implementation of the Rotation Forest Classifier for FDI and Dynamic Control Reallocation (DCR) for FTC specifically for the scenario of complete actuator loss. This innovative combination of machine learning and dynamic control reallocation strategies, tailored for UAVs under severe fault conditions, serves as a key contribution of this work and is anticipated to significantly enhance the state-of-the-art in UAV fault tolerance.

1.2 Contributions

The main contributions of this thesis are:

1. Use of Rotation Forest (RF), an ensemble classification method, can effectively approximate non-orthogonal boundary curves in decision trees. Using a two-stage algorithm, this method can identify faults and classify or pinpoint the failing motor.

2. A large simulation dataset is generated, half of which is used for training the two-stage algorithm. The other half of this dataset is used for validation. In addition, outdoor tests are conducted to verify the system in a test flight where the fault was injected through software.

3. Proposed method (RF) is analysed and compared to existing methods such as Linear Regression, Gaussian Naive Bayes, AdaBoost, and Random Forest.

4. Use of a Weighted Control Allocation (WCA) matrix implemented as a part of dynamic control reallocation for fault tolerant control of hexacopter UAV under single motor failure. This method can successfully hold the UAV's position or land safely in the event of a single motor failure, and some cases of two motor failures.

5. Through extensive flight trials, it has been shown that the proposed method can ensure flight stability in the event of any one of the six motors failing.

6. Proposed solution (WCA) also works in the event of two motor failures. Two motor failure solutions for hexacopters have not been demonstrated in real test flights [36], in the existing literature, to our knowledge.

7. A detailed performance analysis is performed for the proposed fault-tolerant control, quantified in terms of tracking error, percentage overshoot, control effort, settling time, CPU usage, endurance, and effect on closed-loop sampling time. To our knowledge, this study has not been carried out for any of the existing fault-tolerant control architectures.

8. An open-source dataset consisting of single-motor failure simulations is published for training and validation[1]. Another dataset containing real flight trials with post-fault stabilization[2] for faults in different motors is also published.

1.3 Preliminaries

This section details UAV dynamics, fault conditions, fault cases we are considering and how the UAV dynamics change with fault occurrence.

[1]Can be downloaded at https://doi.org/10.5281/zenodo.7739411
[2]Can be downloaded at https://doi.org/10.5281/zenodo.7642240

1.3.1 Hexacopter Dynamics

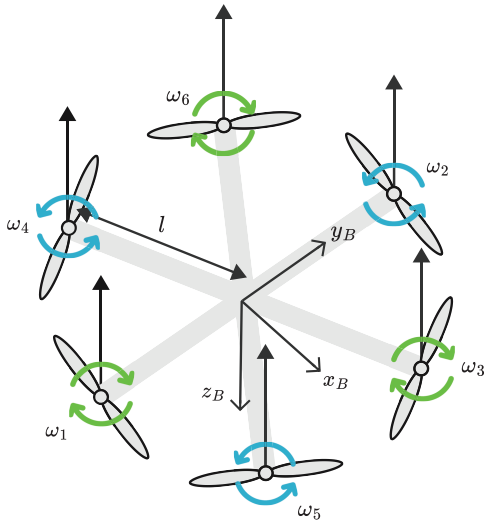

Figure 1.4: Figure shows the Hexacopter in the body-fixed frame, which is represented by the coordinates x_B, y_B, and z_B.

Fig. (1.4) shows a hexacopter with 6 motors with ω_i denoting the individual motor speed for the i^{th} motor. We use a general non-linear hexacopter model derived from its kinematics and dynamics [7]. The state variables are the Euler angles roll-pitch-yaw ($\phi - \theta - \psi$), body axis angular velocities ($\omega_x - \omega_y - \omega_z$), vertical velocity $\dot{z}$ and vertical acceleration $\ddot{z}$.

The moment balance equation is given below.

$$\begin{pmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{pmatrix} = \begin{pmatrix} k_1 \omega_y \omega_z \\ k_3 \omega_x \omega_z \\ k_5 \omega_x \omega_y \end{pmatrix} + \begin{pmatrix} k_2 \tau_\phi \\ k_4 \tau_\theta \\ k_6 \tau_\psi \end{pmatrix} \tag{1.1}$$

Where $k_1 = \frac{Jyy - Jzz}{Jxx}$, $k_2 = \frac{1}{Jxx}$, $k_3 = \frac{Jzz - Jyy}{Jyy}$, $k_4 = \frac{1}{Jyy}$, $k_5 = \frac{Jxx - Jyy}{Jzz}$ and $k_6 = \frac{1}{Jzz}$.

Jxx, Jyy, and Jzz are the moment of inertia along the the axis x_B, y_B and z_B, respectively. The input torques along the body x_B, y_B and z_B axes are denoted by τ_ϕ, τ_θ, and τ_ψ, respectively.

The relation between the Euler angles and ($\omega_x - \omega_y - \omega_z$) is given below.

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & \sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\sin\theta} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{1.2}$$

The altitude dynamics is given in Eq (1.3).

$$\ddot{z} = (\cos\phi \cos\theta) \times \frac{T_f}{m} - g \tag{1.3}$$

Where z is the altitude of the UAV, $\dot{z}$ and $\ddot{z}$ are vertical velocity and acceleration, respectively. T_f is the total thrust generated by the motors, m is the mass of the hexacopter, and g is the acceleration due to gravity.

The relationship between the angular velocity of the i^{th} motor ω_i, to corresponding thrust $T_i\,(N)$ and the torque $\tau_i\,(Nm)$ is modeled as given below.

$$T_i = k_f {\omega_i}^2 \tag{1.4}$$

$$\tau_i = k_m {\omega_i}^2 \tag{1.5}$$

Where T_i, τ_i are generated for motor i at a distance $l\,(m)$ from the center of gravity (as shown in Fig. 1), and $k_f\,(N/rps^2)$, $k_m\,(Nm/rps^2)$ are the propeller thrust and torque coefficients (constants) that depend on the geometry of the propeller and the flight velocity.

The relationship between control inputs M and motor angular velocity ω_a is given below.

$$M = G\,\omega_a \tag{1.6}$$

Where

$$G = \begin{bmatrix} -lk_f & lk_f & \frac{1}{2}lk_f & -\frac{1}{2}lk_f & -\frac{1}{2}lk_f & \frac{1}{2}lk_f \\ 0 & 0 & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f \\ -k_m & k_m & -k_m & k_m & k_m & -k_m \\ k_f & k_f & k_f & k_f & k_f & k_f \end{bmatrix} \tag{1.7}$$

And

$$M = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T_f \end{bmatrix} \tag{1.8}$$

$$\omega_a = \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \\ \omega_5^2 \\ \omega_6^2 \end{bmatrix} \tag{1.9}$$

Eq. (1.6) then becomes.

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T_f \end{bmatrix} = \begin{bmatrix} -lk_f & lk_f & \frac{1}{2}lk_f & -\frac{1}{2}lk_f & -\frac{1}{2}lk_f & \frac{1}{2}lk_f \\ 0 & 0 & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f \\ -k_m & k_m & -k_m & k_m & k_m & -k_m \\ k_f & k_f & k_f & k_f & k_f & k_f \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \\ \omega_5^2 \\ \omega_6^2 \end{bmatrix} \tag{1.10}$$

Here, G acts as the transformation matrix between the motor velocities and the vector of total thrust and torques. Each row of G represents the contribution of the motors to roll, pitch, yaw, and thrust, respectively. Similarly, each column represents the contributions of that i^{th} motor.

1.3.2 Fault Conditions

In this research, we are specifically investigating fault cases related to complete loss of actuation in UAVs. This kind of fault scenario, although severe, is not uncommon and can be induced by a variety of factors, including mechanical failure, electrical problems, or external damage. Our focus is on understanding and addressing the implications of this total actuator loss, from its immediate impact on the UAV's control system to the subsequent effects on the vehicle's flight stability and safety.

Let desired angular velocities of the hexacopter be ω_{xd}, ω_{yd}, and ω_{zd}, and the corresponding desired input torques be $\tau_{\phi d}$, $\tau_{\theta d}$ and $\tau_{\psi d}$ generated by the PID controller. In normal operation, the desired input torques can be achieved by solving the equations relating individual motor thrust and input torque, given by Eq. (1.10), as a function of the angular velocities ω_i. In motor failure, the input torques achieved deviate from the desired input torques as the angular velocity ω_i of the failed motor drops to zero. The input torques achieved can be expressed as a function of the remaining angular velocities and the known properties of the hexacopter by substituting ω_i with zero in Eq. (1.4).

In case of complete motor failure in motor 3, Eq. (1.10) would then become

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T_f \end{bmatrix} = \begin{bmatrix} -lk_f & lk_f & \frac{1}{2}lk_f & -\frac{1}{2}lk_f & -\frac{1}{2}lk_f & \frac{1}{2}lk_f \\ 0 & 0 & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f \\ -k_m & k_m & -k_m & k_m & k_m & -k_m \\ k_f & k_f & k_f & k_f & k_f & k_f \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ 0 \\ \omega_4^2 \\ \omega_5^2 \\ \omega_6^2 \end{bmatrix} \tag{1.11}$$

This fault in motor 3 will result in $\omega_3 = 0$, which will cause that motor's contribution to 3$^{\text{rd}}$ column of matrix G to be 0. In this case, the equations relating the input and desired torques are given by Eq. (1.12), and can be used to analyze the impact of the fault where ω_{3d} is the desired non-zero angular velocity of motor 3.

$$\begin{bmatrix} \tau_{\phi d} \\ \tau_{\theta d} \\ \tau_{\psi d} \end{bmatrix} = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} + \begin{bmatrix} \frac{1}{2}lk_f\omega_{3d}^2 \\ -\frac{\sqrt{3}}{2}lk_f\omega_{3d}^2 \\ -k_m\omega_{3d}^2 \end{bmatrix} \tag{1.12}$$

Eq.(1.12) implies that a motor failure in motor 3 will cause an increase in the applied rolling torque τ_ϕ, resulting in an instantaneous negative change in the roll rate (p) with a magnitude inversely proportional to the moment of inertia, J_{xx}. Similarly, there will be instantaneous negative changes in both the pitch rate (ω_y) and the yaw rate (ω_z). These changes in rotation rates can be analyzed using the known properties of the hexacopter and the remaining angular velocities.

Table (1.1) summarizes the changes in angular velocity that occur in response to the failure of each motor in the hexacopter. "+" and "-" indicate a positive and negative change in angular velocity, respectively. By analyzing these instantaneous changes in angular velocity, we can detect motor faults and develop a fault detection method.

Table 1.1: Instantaneous change in Torque ($\tau_\phi, \tau_\theta, \tau_\psi$) for faults in different motors for the UAV configuration in Fig. 1

Fault in	$\Delta\tau_\phi$	$\Delta\tau_\theta$	$\Delta\tau_\psi$
Motor 1	+	0	-
Motor 2	-	0	+
Motor 3	-	-	-
Motor 4	+	+	+
Motor 5	+	-	+
Motor 6	-	+	-

Here,

$$\Delta\tau_\phi = \tau_{\phi d} - \tau_\phi \tag{1.13}$$

$$\Delta\tau_\theta = \tau_{\theta d} - \tau_\theta \tag{1.14}$$

$$\Delta\tau_\psi = \tau_{\psi d} - \tau_\psi \tag{1.15}$$

The onset of a complete actuator loss triggers substantial changes in the dynamics of a UAV. Actuators play a pivotal role in maintaining and controlling the flight of a UAV; they regulate essential operations like changes in altitude, direction, and speed. When a complete actuator loss occurs, it results in an instantaneous zeroing of the control input for the affected actuator. This abrupt change creates a discontinuity in the UAV's system dynamics, leading to potential instability and erratic behavior. Depending on the specific actuator that fails, the UAV might exhibit sudden shifts in its trajectory, alterations in attitude or velocity, or changes in its equilibrium points. These altered dynamics can make the UAV challenging to control and possibly lead to uncontrolled movement or even a crash if not properly managed. Therefore, understanding how the dynamics change with the occurrence of such a severe fault is crucial for developing effective fault-tolerant control strategies.

1.3.3 Fault Classification

Ensemble methods consisting of accurate and diverse base classifiers are an active research field in machine learning and pattern recognition. Such classifiers can have high accuracy and can avoid coincidence errors [37]. In this method, if a base classifier wrongly classifies a sample, it will be corrected by others, since the combined classifier output is more accurate than any given individual base classifier. Using the ensemble approach, Rotation Forest can reduce the risk of overfitting and improve generalization performance. Moreover, it can handle many features and select the most relevant ones for the task, improving interpretability and computational efficiency. As a fault detection and isolation method in hexacopter UAVs, it shows promising results and highlights the potential of this algorithm in other similar applications.

The classifier is trained with simulation data, where faults are injected through software and the timestamp is logged. Post-flight, this log entry is used to add the class markers in our dataset. In training, Principal Component Analysis (PCA) is performed on a randomly chosen subset of features, and these new uncorrelated principal components form the basis for a new dataset. This process is repeated, generating a 'forest' of classifiers, each trained on a differently rotated version of the original data.

When a new input, such as UAV sensor data, is fed into the system, each classifier in the 'forest' makes a decision. These decisions are then aggregated, often by voting, to form the final output of the Rotation Forest. This output can indicate whether a fault, specifically a complete actuator loss, has occurred, making it a useful for fault detection and isolation (FDI).

The strength of this classifier lies in its robustness to overfitting and its ability to handle high-dimensional data, making it particularly suitable for complex systems like UAVs where a multitude of sensor inputs needs to be monitored and analyzed continuously for fault detection.

1.4 Outline

- In the Second chapter, Fault Detection and Isolation (FDI) on a Hexacopter UAV using a Two-stage classification method is discussed.
- In the Third chapter, Performance evaluation of Dynamic Control Reallocation on a Hexacopter for single and two rotor failure in outdoor test flights is carried out.
- In the Fourth chapter, the two modules (FDI and FTC) are combined and simulation results are presented and analysed.
- In the Fifth chapter, the thesis is concluded and outcomes, future work is discussed.

Chapter 2

Fault Detection and Isolation (FDI) using Rotation Forest Classifier

Identifying whether a motor failure has occurred presents a significant challenge in the Fault Detection and Isolation (FDI) process. Motor failures, particularly a complete actuator loss, can instantaneously alter the UAV's flight dynamics, leading to rapid, potentially catastrophic changes in flight behavior. However, these changes can sometimes be subtle or masked by other simultaneous flight dynamics or environmental factors, making their detection nontrivial. Moreover, different types of fault can result in similar system responses, making it challenging to isolate a motor failure from other potential issues. Rapid and accurate detection and isolation of motor failures are crucial, as they trigger the necessary control reallocation strategies to mitigate the impacts of the fault. Overcoming the complexities associated with accurately diagnosing motor failure in real time is a central aspect of this research, aimed at improving the reliability and safety of UAV operations.

The following problems are crucial for designing a FDI system:

- Detecting a motor failure
  - False negatives can occur - where the FDI module fails to detect a fault even after its occurrence.
- Identifying which motor has failed
- Delay since occurrence of fault and result from FDI system
- Accuracy of the FDI result.
  - False positives can cause the Fault Tolerant Control (FTC) system to inadvertently balance the incorrect motors.

2.1 Overview of Fault Detection & Isolation

Major factors for choosing our FDI algorithm are the ability to choose the most relevant features, to handle time series data, to avoid overfitting, and to provide a result within a duration permitted by

the Fault Tolerant Control algorithm. The input data is dependent on the real-time sensor input, and features depend on the UAV dynamics. Decision Trees are useful for breaking down complex data into multiple manageable parts. We have a large set of features that can vary depending on which motor we want to classify. Furthermore, decision trees are known to perform well with time-series data classification [1], making them a good choice for the classification task. In this chapter, we analyze various classification algorithms in order to solve for i) real-time (online) response within 500ms, ii) accuracy of the classifier, and iii) ability to handle a large set of features. Here, ensemble methods are particularly interesting, as they can be used to increase the confidence value significantly and avoid false negatives. We implemented classifier algorithms: Logistic Regression, Gaussian Naive Bayes, Support Vector Machine (SVM), AdaBoost, Random Forest, and Rotation Forest. In our analysis, we found that Rotation Forest worked best for our use-case and outperforms the others to the best of our knowledge.

2.2 Rotation Forest Algorithm for time-series classification

Due to the complex and coupled nature of our problem, we need a multi-level solution using an ensemble approach. Rotation Forest can reduce the risk of overfitting and improve generalization performance compared to generic Decision Trees. Moreover, it can handle many features and select the most relevant ones for the task, improving interpretability and computational efficiency. As a fault detection and isolation method in Hexacopter UAVs, it shows promising results and highlights the potential of this algorithm in other similar applications.

Rotation Forest algorithm as described in Algo. (1) uses L decision trees, where the feature set for training samples $\mathbf{X}$ with corresponding labels Y and a feature set F containing n features is randomly split into K subsets, each containing $U(=n/K)$ features, the class labels are denoted by α. For each decision tree, a subset of features is selected, and a bootstrap subset of objects is drawn to form a new training set. This training set undergoes a linear transformation to generate coefficients in a matrix, which is used to construct a sparse rotation matrix as shown in Eq. (2.1).

$$\mathbf{R_i} = \begin{bmatrix} a_{1j}^{(1)} & a_{2j}^{(1)} & \cdots & a_{Uj}^{(1)} \\ a_{1j}^{(2)} & a_{2j}^{(2)} & \cdots & a_{Uj}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1j}^{(N/2)} & a_{2j}^{(N/2)} & \cdots & a_{Uj}^{(N/2)} \end{bmatrix} \tag{2.1}$$

Here, $a_{ij}^{(k)}$ is the k^{th} coefficient in the linear transformation of the j^{th} feature subset of the training set for classifier D_i, and N is the total number of objects in the training set. The matrix $\mathbf{R_i} \in \mathbb{R}^{(N/2)\times U}$, where $N/2$ is the size of the bootstrap subset of objects drawn for the training set.

After constructing the sparse rotation matrix for each decision tree, the transformed data matrix $\mathbf{XR_i}$ is trained. Subsequently, the predictions of each decision tree on the test set are aggregated using the mode function, which results in the final prediction as shown in Eq. (2.2).

$$\hat{Y} = \text{mode}(D_1(\mathbf{XR_1}), D_2(\mathbf{XR_2}), \ldots, D_L(\mathbf{XR_L})) \tag{2.2}$$

Algorithm 1: The Rotation Forest algorithm

Input: k (the number of trees), n (the number of features), p (the sample proportion), D (input data)

Output: F (the set of classifiers)

1 Let $L =< L_1 ... L_k >$ be the C4.5 trees [32] in the forest

for $i = 1$ *to* k **do**

2 Randomly partition the original features into U subsets, each with n features ($U = n/K$), denoted $< K_1 ... K_U >$

Let D_i be the training set for tree i, initialised to the original data, $Di \longleftarrow D$.

for $j = 1$ *to* U **do**

3 Select a non-empty subset of classes and extract only cases with those class labels. Each class has 0.5 probability of inclusion.

Draw a proportion p of cases (without replacement) of those with the selected class value.

Perform a Principal Component Analysis (PCA) on the features in K_j of the whole training dataset.

Apply the PCA transform built on this subset to the features in K_j of the whole training dataset.

Replace the features K_j in D_i with the PCA features.

4 Build C4.5 Classifier F_i on the transformed data D_i. Here, each classifier contains decisions for L number of trees.

Here, the C4.5 algorithm [32] used is an innovative method in machine learning for generating decision trees from a dataset for classification tasks. C4.5 decision trees are less biased in comparison to other trees, and are able to efficiently handle both continuous and discrete data - making them a good choice for time-series data. $\hat{Y}$ is the predicted class labels, $D_i(\mathbf{XR_i})$ is the prediction of the i^{th} decision tree on the transformed data matrix $\mathbf{XR_i}$, and mode is the statistical mode function. The decision trees are trained on the transformed data and used to make predictions on the test set. The final prediction is based on the majority vote of the L decision trees, and the final prediction can be used to classify the appropriate actuator fault.

We present a two-stage classifier system that utilizes sensor data to detect and isolate faults on hexacopters in real time. Our primary objective is to improve the safety and reliability of hexacopters while minimizing the potential damage to the vehicle and the environment caused by missing fault conditions.

The proposed system is designed to be highly efficient for real-time operation, with a focus on minimizing the time delay ($\Delta t = t_2 - t_1$) between the occurrence of the fault (t_1) and the detection of the fault (t_2). The system can thus enable control reconfiguration of the hexacopter before it becomes unstable, ensuring vehicle safety.

It also aims to achieve a high true positive detection rate (correctly identifying the presence of a fault) ensuring that any issues with the hexacopter are detected and addressed promptly. A low false negative detection rate is achieved to reduce the possibility of missing any fault conditions.

The proposed system utilizes the Rotation Forest algorithm to achieve these objectives, designed to capture complex relationships between features and class labels. The confidence value $C_v(\alpha|\mathbf{X})$ is used to adjust the contribution of each decision tree in the algorithm, with weights assigned based on the accuracy of the decision tree on the training data.

$$C_v(\alpha|\mathbf{X}) = \frac{\sum_{i=1}^{L} w_i D_i(\mathbf{XR_i}, \alpha)}{\sum_{i=1}^{L} w_i} \tag{2.3}$$

Here, $D_i(\mathbf{XR_i}, \alpha)$ is the predicted probability of class label α for the i^{th} decision tree, and $\mathbf{X}$ is the test sample transformed by $\mathbf{R_i}$. The weight w_i is assigned based on the accuracy of the decision tree on the training data, with higher accuracy trees receiving higher weights.

In conclusion, the proposed two-stage classifier system represents a significant step forward in detecting and isolating faults on hexacopters.

A data-driven method is used to detect and isolate the failure of a single rotor for a hexacopter UAV during a flight. Considering the Eq. (1.10) describing the rotational dynamics of the hexacopter and Eq. (1.12) showing the change in input torques, the following functional relations are obtained for the k^{th} discrete time sample.

$$\Delta M(k) = f(\omega_{x,y,z}(k), \omega_{x,y,z}(k-1), \omega_d(k)) \tag{2.4}$$

Where $\Delta M(k) = [\Delta M_\phi(k), \Delta M_\theta(k), \Delta M_\psi(k)]^T$ is a vector representing the changes in the input torques, and f and $\omega_{x,y,z}$ are vectors containing the functions $f_1(.)$, $f_2(.)$ and $f_3(.)$ and the angular velocities

$\omega_x, \omega_y, \omega_z$, respectively. The vector $\omega_d(k) = [\omega_{xd}(k), \omega_{yd}(k), \omega_{zd}(k)]^T$ represents the desired angular velocities.

2.3 Two-stage classification based on Rotation Forest algorithm

The architecture of the proposed data-driven method is given in Fig. (2.1).

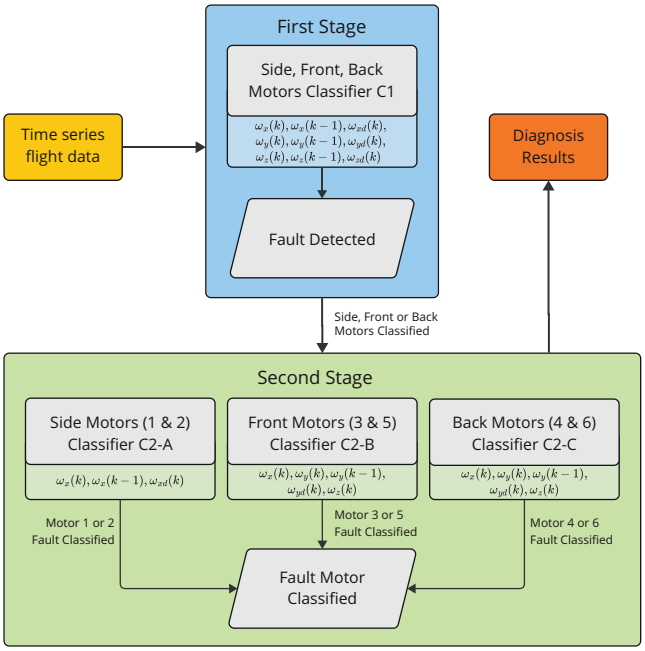

Figure 2.1: Flowchart of the two-stage fault classifier

The proposed method contains two-stage fault detection and isolation on a hexacopter UAV, which offers several advantages. Firstly, it enables us to detect and group a fault based on its type and then classify which motor is affected. This approach simplifies the problem and results in higher accuracy. In the group classifier stage, sensor data is analyzed to identify any anomalies or deviations from normal behaviour, indicating the presence of a fault. Once a fault is detected, the motor classification stage identifies the failed motor.

2.3.1 Fault Detection - First Stage

In the first stage of the proposed method, the fault is detected and grouped into one of the three categories - front, back, or side motor, to efficiently and effectively distinguish between the faults occurring in the motor. To achieve this, all the variables used in Eq. (2.4) and their past values are used to classify the fault.

Once the fault is classified in the first stage, we reduce the number of variables used in the second stage. Specifically, we only use the variables related to the first stage variable classification C1. This reduces the classification process's computational complexity and improves the algorithm's efficiency. This matrix $\in \mathbb{R}^{k\times 9}$, where k is the current sample.

$$\text{Classifier C1} \rightarrow \begin{bmatrix} \omega_x(1)\dots\omega_x(k) \\ \omega_x(0)\dots\omega_x(k-1) \\ \omega_{xd}(1)\dots\omega_{xd}(k) \\ \omega_y(1)\dots\omega_y(k) \\ \omega_y(0)\dots\omega_y(k-1) \\ \omega_{yd}(1)\dots\omega_{yd}(k) \\ \omega_z(1)\dots\omega_z(k) \\ \omega_{z(0)}\dots\omega_z(k-1) \\ \omega_{zd}(1)\dots\omega_{zd}(k) \end{bmatrix} \tag{2.5}$$

The results of this stage of classification (detection) are passed to the next stage to isolate and diagnose the fault.

2.3.2 Fault Classification - Second Stage

In the second stage, the objective is to further refine fault detection and isolate the malfunctioning motor. For this purpose, we use a subset of variables specifically chosen to distinguish between the different motors using Eq. (1.10) and Table (1.1).

In the case of classification between motors 1 and 2 (C2-A), the matrix includes past values of ω_x and ω_{xd}, which are variables related to the angular velocity of the hexacopter on the x axis. These variables are relevant to distinguish between motors 1 and 2 because a fault in one of these motors would affect the angular velocity of the hexacopter on the x axis. This matrix $\in \mathbb{R}^{k\times 3}$.

$$\text{Classifier C2-A}$$

$$\downarrow$$

$$\begin{bmatrix} \omega_x(1) & \omega_x(0) & \omega_{xd}(1) \\ \vdots & \vdots & \vdots \\ \omega_x(k) & \omega_x(k-1) & \omega_{xd}(k) \end{bmatrix} \tag{2.6}$$

For the classification between motors 3 and 5 (C2-B) and between motors 4 and 6 (C2-C), additional variables such as ω_y and ω_z are included. These variables are related to the angular velocity of the hexacopter on the y-axis and the z-axis, respectively, and help distinguish between motors (3, 5) and (4, 6). The matrix used for C2-B and C2-C $\in \mathbb{R}^{k\times 5}$.

$$\text{Classifier C2-B, C2-C}$$

$$\downarrow$$

$$\begin{bmatrix} \omega_x(1) & \omega_y(1) & \omega_y(0) & \omega_{xd}(1) & \omega_z(1) \\ \vdots & \vdots & \vdots & \vdots & \\ \omega_x(k) & \omega_y(k) & \omega_y(k-1) & \omega_{xd}(k) & \omega_z(k) \end{bmatrix} \tag{2.7}$$

Using a subset of variables specific to each motor, the fault detection and isolation method can effectively isolate the malfunctioning motor. This approach makes the method more accurate and reliable in detecting and isolating motor failures in hexacopters.

In general, the two-stage approach used in this method allows for a more robust and efficient fault detection and isolation process, especially in the context of a Hexacopter UAV, where quick and accurate identification of faults is crucial for the safety and success of the mission.

2.4 Training Datasets

The simulation datasets were generated by introducing faults through the software on a Hexacopter UAV in the Gazebo simulation environment. The UAV is set to hover for a few seconds, before the fault is introduced. The sampling rate is set to 50 Hz, and contains the sensor data ($\omega_x, \omega_y, \omega_z$) as well as the PID controller's desired values ($\omega_{xd}, \omega_{yd}, \omega_{zd}$).

For the training dataset,

$$\mathbf{X_T} = [\omega_x(.),\, \omega_{xd}(.),\, \omega_y(.),\, \omega_{yd}(.),\, \omega_z(.),\, \omega_{zd}(.)] \tag{2.8}$$

Where, 60 dataset files (10 per motor) are used that contain approximately 200 time series data points. These time series data points consist of the features selected above in Section III. Each dataset file is generated with a 90% signal-to-noise ratio. The classifier models generated in training are saved for later use in the validation/testing phase.

The validation dataset $\mathbf{X_V}$ also has 60 dataset files containing approximately 200 data points each, but is generated with a 80% signal-to-noise ratio instead. This adds slightly higher error values to the input dataset while trying to mimic real flight behavior.

Here, $\mathbf{X_T} + \mathbf{X_V} =$ total dataset.

For our case, the Rotation Forest classifier is restricted to 200 trees in training, and takes approximately 6 to 8 minutes to train the datasets defined above.

2.5 Validation & Results

This section provides the results for the proposed classification method in simulated datasets followed by experimental datasets.

2.5.1 Validation with simulation dataset

Validation is performed on the dataset X_V, which contains the log files from our Ardupilot simulation. Futher analysis demonstrates the proposed fault detection and classification approach and benchmarks

it against other existing algorithms. The proposed classifiers can accurately detect and classify motor system faults despite the addition of simulated disturbances and noise. These results are significant because they advance the development of reliable and robust methods for detecting and isolating faults in complex systems such as unmanned aerial vehicles (UAVs).

Figures (2.2) - (2.7) show the performance of classifiers C1 and C2(A,B,C) under simulated conditions, where a fault is introduced during a flight. Each figure displays the trajectory of the features (state variables) over time and the points at which the classifiers are activated. Note that stage one classifier C1 gives a group output denoting the locality of the motors, based on the functional relationship between the input and output variables. Second stage classifiers C2-A, C2-B and C2-C, however, classify the fault to the exact failing motor.

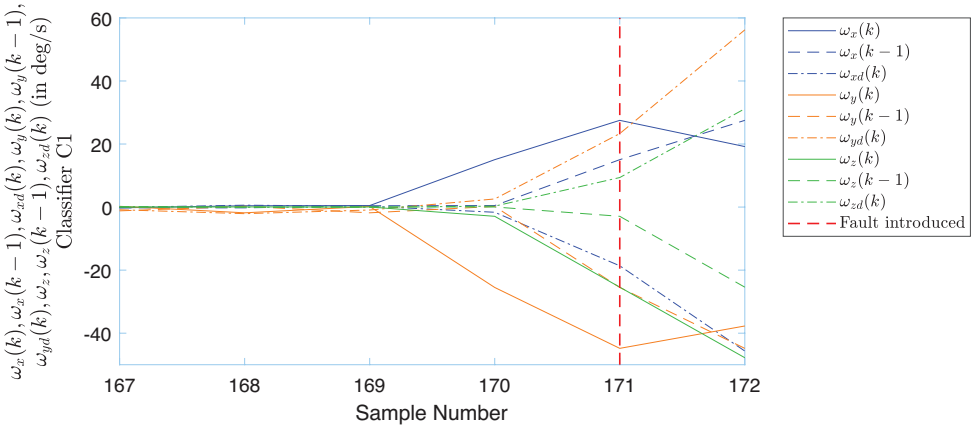

Figure 2.2: Trajectory of the features(state variables) for fault detection on Motor 1, showing classifier C1 running after a fault is introduced in simulation. The vertical red line shows the time at which fault is introduced. After fault occurrence, the classifier output (motor locality) is received in the next sample.

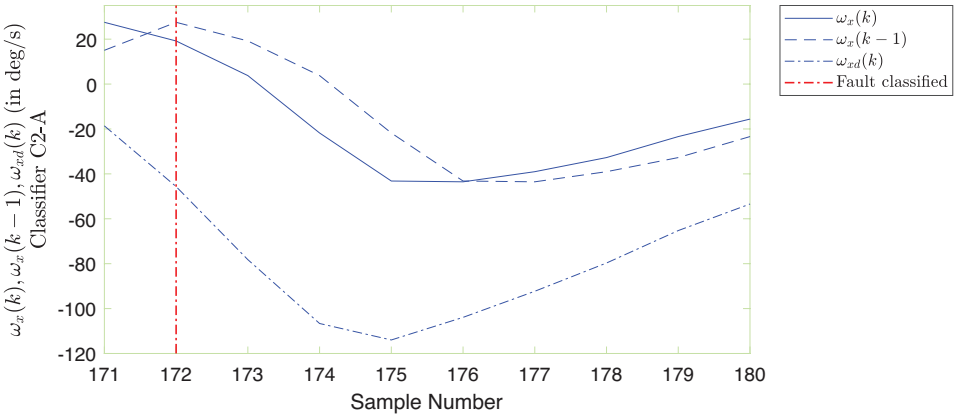

Figure 2.3: Trajectory of the features(state variables) for fault classification on Motor 1, showing classifier C2-A running after a fault is introduced in simulation. The vertical red line shows the time at which fault is introduced. After fault occurrence, the classifier output (motor number) is received in the next sample.

Fig. (2.2), (2.3) illustrate the trajectory of the features (state variables) used for fault detection and classification in Motors 1 in a validation study. Here, the roll axis values ω_x, ω_{xd} are affected the most, due to motor symmetry as shown in Fig. (1.4).

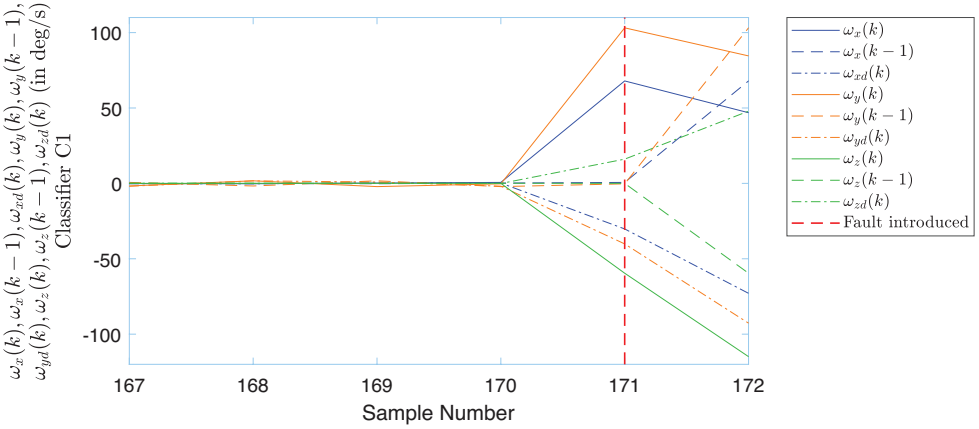

Figure 2.4: Trajectory of the features(state variables) for fault detection on Motor 3, showing classifier C1 running after a fault is introduced in simulation. The vertical red line shows the time at which fault is introduced. After fault occurrence, the classifier output (motor locality) is received in the next sample.

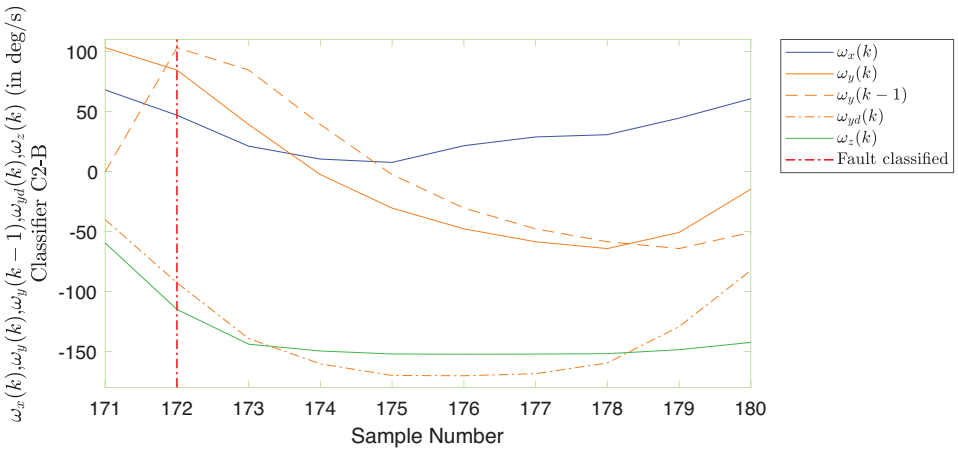

Figure 2.5: Trajectory of the features(state variables) for fault classification on Motor 3, showing classifier C2-B running after a fault is introduced in simulation. The vertical red line shows the time at which fault is introduced. After fault occurrence, the classifier output (motor number) is received in the next sample.

Fig. (2.4), (2.5) illustrate the trajectory of the features (state variables) used for fault detection and classification in Motors 3 in a validation study. Here, a net positive effect is seen on both the roll axis (ω_x, ω_{xd}), and pitch axis (ω_y, ω_{yd}) values, indicating a failure in the front motors (Motor 3 or Motor 5).

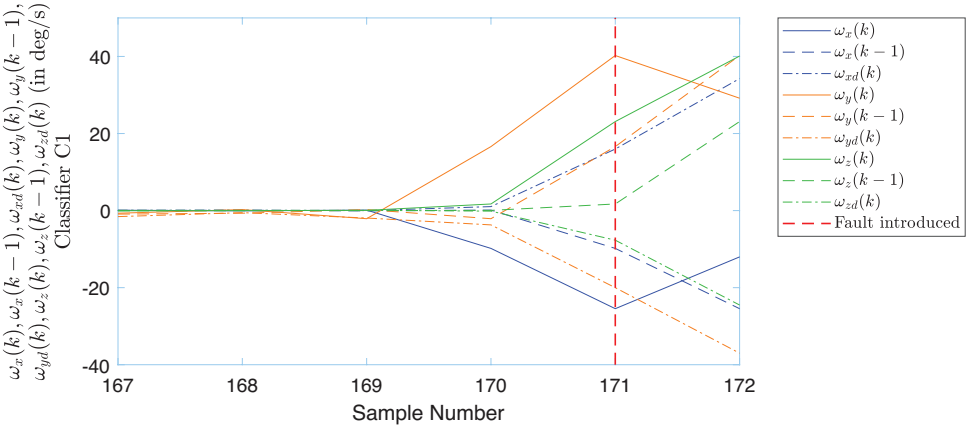

Figure 2.6: Trajectory of the features(state variables) for fault detection on Motor 4, showing classifier C1 running after a fault is introduced in simulation. The vertical red line shows the time at which fault is introduced. After fault occurrence, the classifier output (motor locality) is received in the next sample.

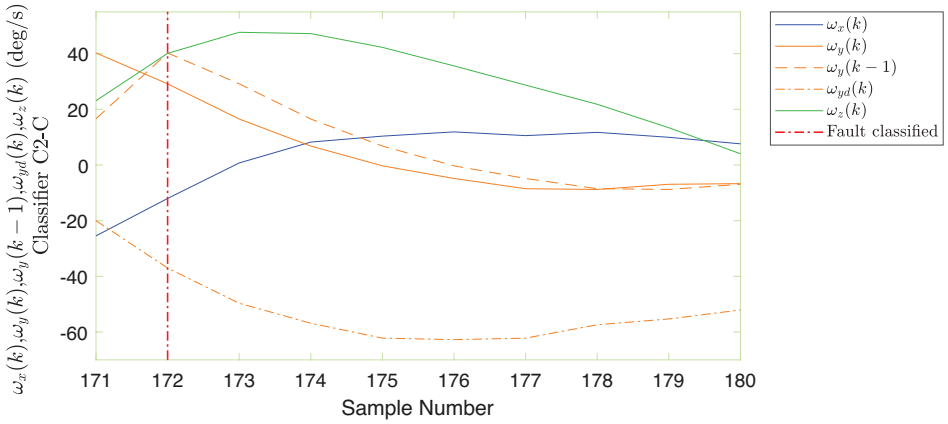

Figure 2.7: Trajectory of the features(state variables) for fault classification on Motor 4, showing classifier C2-C running after a fault is introduced in simulation. The vertical red line shows the time at which fault is introduced. After fault introduction, the classifier output (motor number) is received in the next sample.

Fig. (2.6), (2.7) illustrate the trajectory of the features (state variables) used for fault detection and classification in Motors 4 in a validation study. Here, the roll axis values (ω_x, ω_{xd}) have a net positive effect, while the pitch axis values (ω_y, ω_{yd}) have a net negative effect, indicating a failure in the back motors (Motor 4 or Motor 6).

2.5.2 Comparision with other Classifiers

We further process the same training (X_T) and validation (X_V) datasets with other classifiers, namely, Logistic Regression, Gaussian Naive Bayes, AdaBoost, Support Vector Machines (SVMs), and Random Forest.

Table 2.1: Fault Detection accuracy for different algorithms in simulations

Classifier	**Accuracy**
Logistic Regression	1.0
Gaussian Naive Bayes	0.96
AdaBoost	0.99
Support Vector Machine (SVM)	0.98
Random Forest	1.0
Rotation Forest	1.0

Table (2.1) shows a comparison between different classifier's ability to detect faults. Here, the classifier's ability to identify the failing motor is not considered. Instead, each of the classifiers is run across

the entire validation dataset X_V, and a binary output is taken where 0 represents no fault and 1 represents a fault. All the algorithms in comparison show a high accuracy for detection of faults.

Table 2.2: Fault Classification's confidence value (C_v) rate for different algorithms in Simulations

Motor Number	Logistic Regression	Gaussian Naive Bayes	AdaBoost	SVM	Random Forest	Rotation Forest
1	0.8	0.82	0.91	0.92	0.88	0.92
2	0.98	0.91	0.8	0.78	0.75	0.89
3	0.94	0.87	0.69	0.74	0.88	0.97
4	0.75	0.88	0.79	0.81	0.83	0.91
5	0.92	0.93	0.7	0.8	0.77	0.94
6	0.88	0.94	0.93	0.89	0.92	0.97
Avg	0.878	0.891	0.803	0.823	0.838	0.926

Table (2.2) shows the confidence value rates (C_v) for different algorithms compared to ours. Here, C_v is calculated by averaging the class probabilities output by the individual trees in the ensemble. The rates shown here refer to the classifier's ability to identify the failing motor. Logistic Regression [6, 39] had high accuracy in the simulation, but its performance was affected by outliers. Gaussian Naive Bayes [10] assumes independence among predictors, which is not suitable for the coupled nature of Hexacopter UAV dynamics. AdaBoost [34] is more prone to overfitting in noisy datasets, which is often the case for UAVs due to significant IMU data noise. Random Forest [8, 9] can provide unsatisfactory results for datasets with multiple outliers due to its grid-like decision structure. On the other hand, Rotation Forest can approximate non-orthogonal boundary curves in decision trees and is an improvement over Random Forest. Rotation Forest [28] showed better overall accuracy than other models, with an average true positive rate of 92.6% during simulation dataset validation. Here, the true positive rate (T_{pr}) is calculated as the ratio of true positives to the sum of true positives and false negatives ($T_{pr} = Tp/(Tp+Fn)$).

2.5.2.1 Decision Boundary Comparison

In this section, we visualize the decision boundaries in order to compare the different classifiers and analyze their performance.

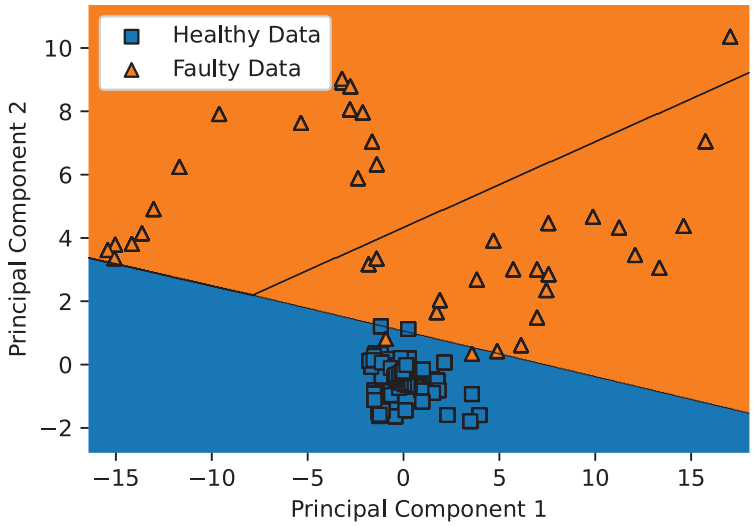

Figure 2.8: Decision boundaries for the Logistic Regression based classifier

Fig. (2.8) shows the decision boundaries with the Logistic Regression based classifier. For our case, Logistic Regression underfits and classifies data points that are in a cluster as the same class. This happens because of the high number of features, highly unbalanced datasets [21] and varying amount of noise in the sensor data.

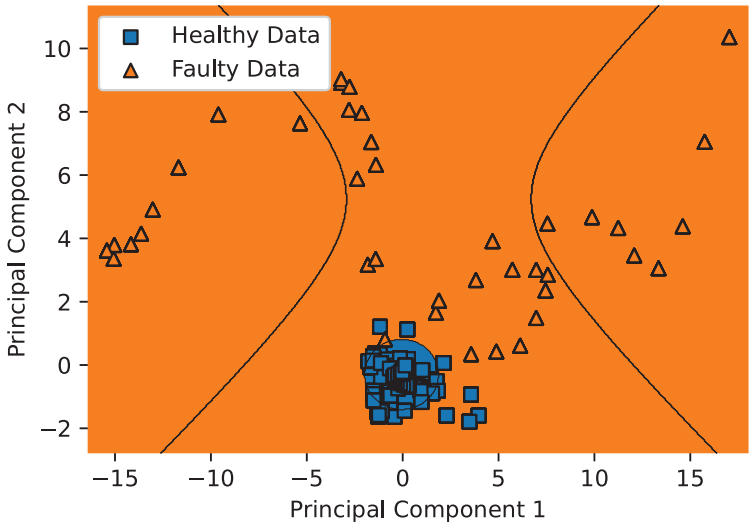

Figure 2.9: Decision boundaries for the Gaussian Naive Bayes based classifier

Fig. (2.9) shows the decision boundaries with the Gaussian Naive Bayes (GNB) based classifier. GNB can have curved boundaries due to the Gaussian distribution function it uses. Additionally, GNB assumes that features are independent of each other, and thus it misclassifies outliers [17].

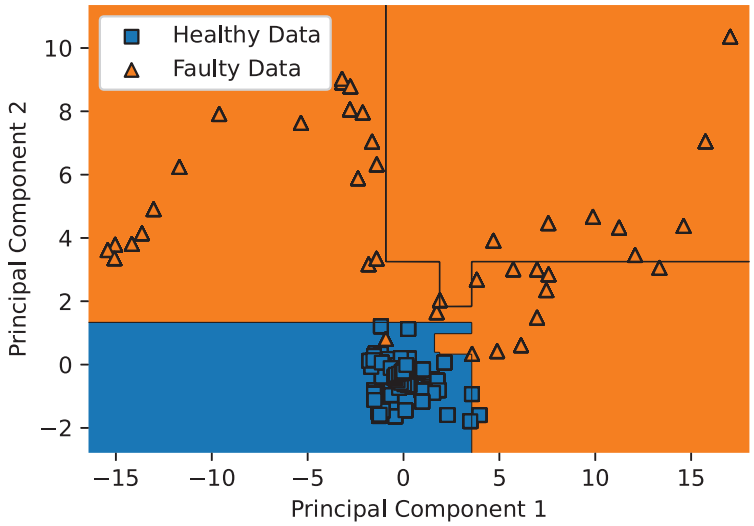

Figure 2.10: Decision boundaries for the AdaBoost based classifier

Fig. (2.10) shows the decision boundaries with the AdaBoost classifier. AdaBoost focuses on minimizing the error and hence can underfit on complex datasets, even though it provides very clean boundaries. In low-noise datasets, AdaBoost rarely overfits, but our input dataset is sensor-based and hence noisy. Furthermore, the performance is poor on unseen data [16]. i.e., new data points in validation begin to overfit.

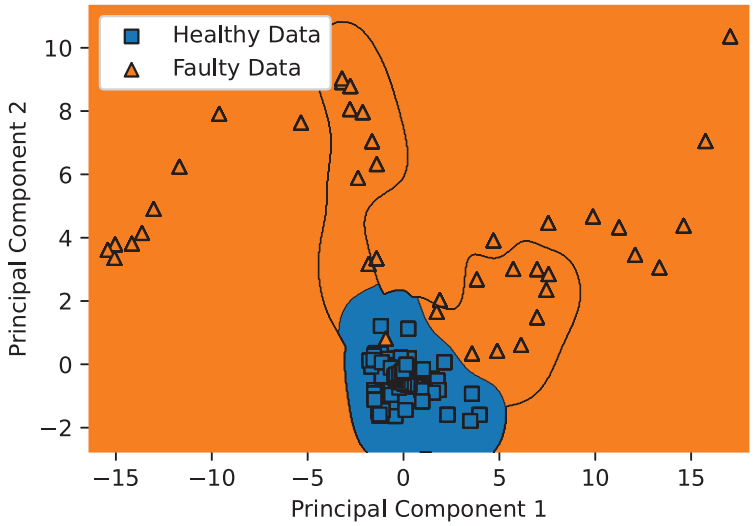

Figure 2.11: Decision boundaries for the Support Vector Machine (SVM) based classifier

Fig. (2.11) shows the decision boundaries with the Support Vector Machine (SVM) based classifier. This classifier performance reduces when the target classes overlap, and begins to underperform [3] as the number of data points keeps increasing.

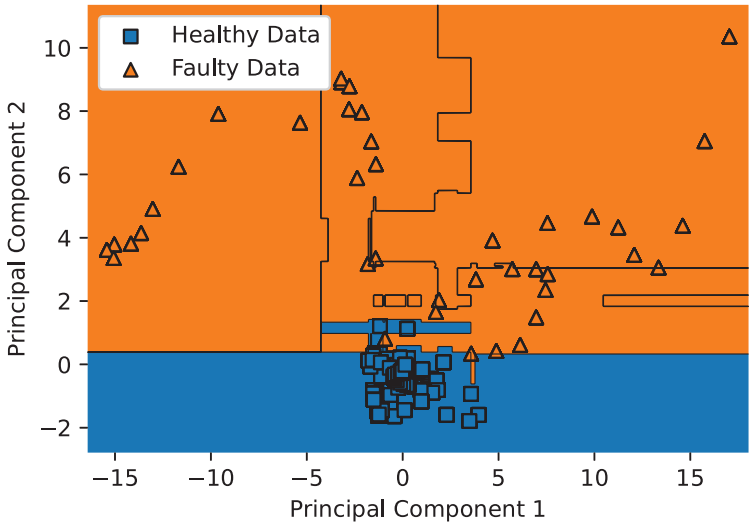

Figure 2.12: Decision boundaries for the Random Forest based classifier

Fig. (2.12) shows the decision boundaries with the Random Forest classifier. Random Forest works very well in terms of classifying the data, but as the number of trees grows, the algorithm begins to slow down [13] and hence proves inefficient for real-time (online) predictions.

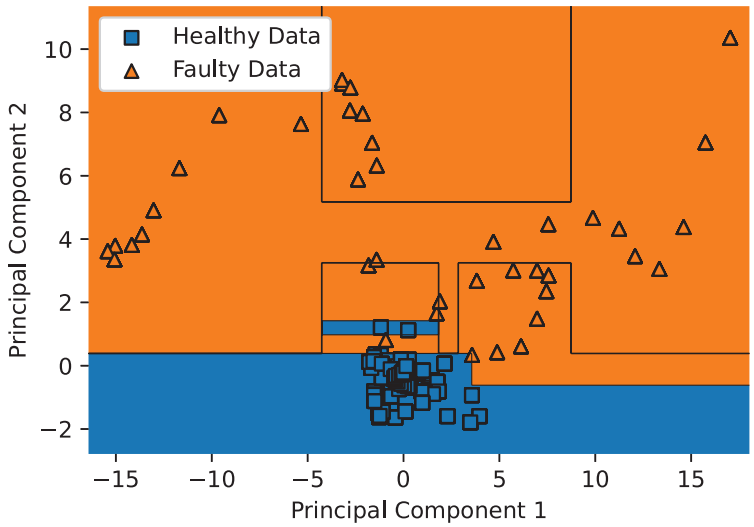

Figure 2.13: Decision boundaries for the Rotation Forest based classifier

Fig. (2.13) shows the decision boundaries with the Rotation Forest classifier. Rotation Forest boundaries are less complex in comparison to Random Forest, due to its non-orthogonal curve approximation. Even though Rotation Forest needs more computation during training in comparison to Random Forest [4], it's real-time (online) predictions are faster due to the reduction in decision tree depth.

2.5.3 Results of classification on experimental dataset

Figure 2.14: Our Hexacopter setup for outdoor flight testing consisting of the flight controller, external GPS and raspberry pi.

We used a hexacopter equipped with open-source ArduPilot firmware running on a CUAV v5+ Pixhawk board for testing/validating in real flights. To test the classifier, we ran the validation code on the post-flight data and found that the delay Δt is 60 ms. In real flights, the classifier responds to the fault within 2 to 5 samples, while it responds in the next sample for simulations. This gives us a delay of $2\Delta t$ to $5\Delta t$ after the fault before the controller can be notified of the occurrence of the fault.

Table 2.3: Our algorithm's True Positive and False Negative values for real test flights

Motor	True Positive	False Negative
1	0.93	0.07
3	0.97	0.03
6	0.98	0.02

Table (2.3) shows the test flight results for different groups chosen earlier for the two-stage approach. These values are averaged over two test flights for each case. Here, false negative refers to cases where a fault occurs but the classifier still labels it as normal state. This occurs when the classifier does not label the fault in the exact sample after fault injection since the input parameters do not change as quickly as the operating frequency.

2.6 Conclusion & Inference

We demonstrated a Fault Detection and Isolation (FDI) module, which uses an ensemble classifier, Rotation Forest. Our proposed classifier takes state inputs of roll rate (ω_x), pitch rate (ω_y) and yaw rate (ω_z) and predicts the occurrence of a fault. These input rates are a result of the change in torque

as discussed in Eq. (1.12) and in Table (1.1). The first stage of our classifier is used to detect a fault, while the second stage is used to isolate the fault. The classifier reports its first stage in 60 ms and can isolate the fault to the failing motor in a total of 120 to 250 ms (including the time taken for detection and input/output processing). The prediction values are published by the classifier at a constant rate (50 Hz) and can be subscribed from the Fault Tolerant Control (FTC) module, in order to perform post-fault stabilization.

Chapter 3

Fault Tolerant Control (FTC) using Dynamic Control Re-allocation

Following the detection and isolation of complete actuator loss, the next critical component in ensuring robust UAV operation is the implementation of a suitable Fault Tolerant Control (FTC) strategy. The objective of FTC is to maintain the stability and controllability of the UAV despite the occurrence of faults. The primary challenge lies in dynamically reallocating control among the remaining operational actuators in a manner that compensates for the effects of the actuator loss. This section delves into the intricacies of FTC for UAVs experiencing complete actuator loss, detailing the proposed control reallocation methodology, its mathematical analysis, and its effectiveness in maintaining system stability under such severe fault conditions.

The previous chapter details our Fault Detection and Isolation (FDI) module, which reacts to changes in body angular rates ($\omega_x, \omega_y, \omega_z$) and Torque and predicts the current state of the vehicle. This prediction can be used to correct the fault and stabilize the vehicle.

This chapter contains equilibrium analysis for determining the effects of a fault on the UAV dynamics, controllability test, the control allocation method used and it's attained control authority. Further, we discuss our FTC module and analyze its performance in terms of the change in motor angular velocities, tracking error, percentage overshoot, settling time, control effort, estimated vehicle endurance, CPU load, RAM usage, and closed-loop sampling delay.

3.1 Equilibrium analysis

Since most of the UAV's time is spent in a cruise or hover, which are both relatively close to the hover condition, we can analyze the UAV behavior at hover condition and study the effects of faults. As discussed earlier in Eq. (1.12), the occurrence of the fault can have some effect on the magnitude of torque inputs. The focus of our work is to successfully stabilize the hexacopter by controlling the angular momentum and altitude, based on the aforementioned torque changes. For this, we observe the equilibrium at hover and analyze the effects of fault occurrence on the UAV dynamics.

3.1.1 Equilibrium at hovering:

The equilibrium at the hovering state is given by following conditions in Eq (3.1 - 3.2).

$$\dot{\omega}_x = \dot{\omega}_y = \dot{\omega}_z = \dot{\phi} = \dot{\theta} = \dot{\psi} = \dot{z} = \ddot{z} = 0 \tag{3.1}$$

and

$$\phi = \theta = \omega_x = \omega_y = \omega_z = 0 \tag{3.2}$$

The variables ψ and z can be non-zero constants. The corresponding equilibrium input is denoted by $M = M_e = [\tau_{\phi e} = 0, \tau_{\theta e} = 0, \tau_{\psi e} = 0, T_{fe} = mg]$ obtained from Eq (1.1 - 1.3). From Fig. (1.4), the expression for $\tau_{\phi e}, \tau_{\theta e}, \tau_{\psi e}$, and T_{fe} are given in Eq (3.3 - 3.6).

$$\tau_{\phi e} = l[T_1 + T_4 + T_5 - (T_2 + T_3 + T_6)] = 0 \tag{3.3}$$

$$\tau_{\theta e} = l[T_4 + T_6 - (T_3 + T_5)] = 0 \tag{3.4}$$

$$\tau_{\psi e} = [\tau_2 + \tau_4 + \tau_5 - (\tau_1 + \tau_3 + \tau_6)] = 0 \tag{3.5}$$

$$T_{fe} = \sum_{i=1}^{6} T_i = mg \tag{3.6}$$

3.1.2 Deviation from equilibrium hover due to motor fault:

In case of i^{th} motor failure, $T_i = 0$, the control input M deviates from M_e as given below.

$$\tau_\phi = \tau_{\phi e} - lT_i \tag{3.7}$$

$$\tau_\theta = \tau_{\theta e} - lT_i \tag{3.8}$$

$$\tau_\psi = \tau_{\psi e} - \tau_i \tag{3.9}$$

$$T_f = T_{fe} - T_i \tag{3.10}$$

This deviation, which is a function of T_i and τ_i, can lead to flight instability as the $\omega_x, \omega_y, \omega_z, \theta, \phi$ values deviate from the equilibrium condition when the control input is given in Eq (3.11 - 3.14) is applied to Eq (1.1 - 1.3).

For example, in the case of failure in motor 1, the deviation will result in M_{f1} given below.

$$\tau_{\phi f1} = l * k_f[{\omega_2}^2 + \frac{1}{2}({\omega_3}^2 - {\omega_4}^2 - {\omega_5}^2 + {\omega_6}^2)] \tag{3.11}$$

$$\tau_{\theta f1} = l * k_f \frac{\sqrt{3}}{2}(-{\omega_3}^2 + {\omega_4}^2 - {\omega_5}^2 + {\omega_6}^2) \tag{3.12}$$

$$\tau_{\psi f1} = k_m({\omega_2}^2 - {\omega_3}^2 + {\omega_4}^2 + {\omega_5}^2 - {\omega_6}^2) \tag{3.13}$$

$$T_{f f1} = k_f({\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2 + {\omega_5}^2 + {\omega_6}^2) \tag{3.14}$$

By simplifying Eq. (3.11 - 3.14), we can see that the net resultant torque and thrust are effective on the body axis as given below.

$$\tau_{\phi f1} = l * k_f({\omega_2}^2) \tag{3.15}$$

$$\tau_{\theta f1} = 0 \tag{3.16}$$

$$\tau_{\psi f1} = k_m({\omega_2}^2) \tag{3.17}$$

$$T_{f f1} = k_f({\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2 + {\omega_5}^2 + {\omega_6}^2) \tag{3.18}$$

From Eq. (3.15 - 3.18) it can be observed that a deviation occurs in the roll, yaw and thrust, in case of fault in Motor 1. This deviation pushes the UAV out of equilibrium. The divergence is unbounded and can eventually lead to instability. In such a case, equilibrium can only be achieved if the diagonally opposite motor's angular velocity $\omega_2 = 0$ and the rest of the motors can provide enough thrust to balance the weight of the UAV. In real flights, turning off the opposing motor (in case of simultaneous failures in motors 1, 2) causes an imbalance in the direction of remaining propellors, resulting in a loss of yaw control. Similarly, for some other cases (simultaneous failure in motors 3, 5; motors 3, 6), the pitch and roll control respectively are lost significantly.

For the above example of failure in motor 1; the condition $\omega_2 = 0$ cannot be satisfied. In such a scenario, the original hovering equilibrium cannot be attained.

3.1.3 Controllability

We can study the behavior of our closed-loop control system in Eq. (1.1, 1.3), by linearizing at the equilibrium point under hover condition. The state equation for this linear first-order system is of the form given below.

$$\dot{s} = A\ s + B\ u \tag{3.19}$$

Where s is a state vector representing the attitude of the vehicle,

$$s = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ \dot{z} \end{bmatrix} \tag{3.20}$$

$\dot{s}$ is the derivative of the state vector,

A is a system matrix that is derived by linearizing our system around the hover equilibrium.

$$A = \frac{\partial \dot{s}}{\partial s}\bigg|_{s=s_h} = \begin{bmatrix} 0 & k1\ \omega_z & k1\ \omega_y & 0 \\ k3\ \omega_z & 0 & k3\ \omega_x & 0 \\ k5\ \omega_y & k6\ \omega_x & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.21}$$

Where s_h is the state vector under hover conditions as mentioned in Eq. (3.1-3.2).

B is the input matrix as given below.

$$B = J_c\, G \tag{3.22}$$

Where J_c is the matrix form derived by writing the inertia components (k_2, k_4, k_6) in Eq. (1.1) and G is as defined in Eq. (1.7).

$$J_c = \begin{bmatrix} k_2 & 0 & 0 & 0 \\ 0 & k_4 & 0 & 0 \\ 0 & 0 & k_6 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.23}$$

From Eq. (1.6), we have a relationship between torque, thrust inputs, and the motor angular velocities. Further, the control input u is given below.

$$u = \begin{bmatrix} {\omega_1}^2 \\ {\omega_2}^2 \\ {\omega_3}^2 \\ {\omega_4}^2 \\ {\omega_5}^2 \\ {\omega_6}^2 \end{bmatrix} \tag{3.24}$$

In a healthy condition, the state equation can be written as given in Eq. (1.1) and Eq. (3.19). After a fault occurs, the deviation is as mentioned in Eq. (3.11 - 3.14) changes the behavior due to the additional thrust and torque components.

We check controllability using the Controllability test, where the rank of the matrix C_i is checked to determine whether loss of control over a control input has occurred after motor failure in given i^{th} motor.

$$C_i = [B_i \; AB_i \; A^2B_i \ldots A^{\text{n-1}}B_i] \tag{3.25}$$

For motor 1 failure, we get B_1 and C_1 as given below. In this case, C_1 has full rank. Here, B_1 is given below.

$$B_1 = Jc\,G_1 \tag{3.26}$$

Where G_1 is given below.

$$G_1 = \begin{bmatrix} 0 & lk_f & \frac{1}{2}lk_f & -\frac{1}{2}lk_f & -\frac{1}{2}lk_f & \frac{1}{2}lk_f \\ 0 & 0 & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f \\ 0 & k_m & -k_m & k_m & k_m & -k_m \\ 0 & k_f & k_f & k_f & k_f & k_f \end{bmatrix} \tag{3.27}$$

and

$$C_1 = \begin{bmatrix} B_1 \; AB_1 \; A^2B_1 \; A^3B_1 \end{bmatrix} \tag{3.28}$$

Since C_1 maintains rank equal to number of states, it is a controllable system. Similarly, since all cases of single motor failure do not cause rank deficiency in the matrix C_i, the given system is controllable.

3.2 Control Allocation

Control allocation is a fundamental problem in the realm of Fault Tolerant Control (FTC) for UAVs [15], particularly in the scenario of complete actuator loss. Essentially, control allocation involves determining how to efficiently distribute control efforts among the available actuators to achieve the desired output of the system. This problem becomes notably complex in the event of a complete actuator loss, as the control efforts need to be redistributed amongst the remaining operational actuators. A primary challenge here lies in ensuring that the redistribution does not saturate the functioning actuators.

In order to reallocate the motor outputs after fault occurrence, we can modify the G matrix in Eq. (1.6).

Hence, the motor angular velocity ω_a is computed as given below.

$$\omega_a = G^{+}M \tag{3.29}$$

Where

$$G^{+} = G^{\text{T}}(GG^{\text{T}})^{\text{-1}} \tag{3.30}$$

is the pseudo-inverse of G matrix. Note that G is always non-singular.

In a normal fault-free scenario, Eq (1.6) has a unique solution for ω_a using the pseudo-inverse method given in Eq (3.30). Let us consider the case of motor failure where $\omega_i = 0$ for any of the i^{th} motors. Then the solution provided by Eq (3.29) does not satisfy the condition $\omega_i = 0$. Enforcing $\omega_i = 0$ and retaining the other terms of ω_a obtained using Eq (3.29) does not satisfy Eq (1.10).

This means that the desired torque and thrust generated by the PID controller cannot be achieved by the default control allocation. This difference in the desired torque and thrust and the one obtained through control allocation can lead to flight instability.

Let the new torque and thrust achieved through control allocation for i^{th} motor failure be denoted by M_i. The objective is to find a modified matrix M_{in} such that the stability of the UAV is maintained by making it as close to the desired value M. The flight condition considered here is the hovering [22] state.

The dynamic control reallocation proposed here reduces the difference between M and M_e to retain flight stability. After the i^{th} motor fails, the corresponding i^{th} column of G in Eq (1.7) is set to 0, and G_i denotes the corresponding matrix.

For example, a fault in motor 1 leads to G_1 as given below.

$$G_1 = \begin{bmatrix} 0 & lk_f & \frac{1}{2}lk_f & -\frac{1}{2}lk_f & -\frac{1}{2}lk_f & \frac{1}{2}lk_f \\ 0 & 0 & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f & -\frac{\sqrt{3}}{2}lk_f & \frac{\sqrt{3}}{2}lk_f \\ 0 & k_m & -k_m & k_m & k_m & -k_m \\ 0 & k_f & k_f & k_f & k_f & k_f \end{bmatrix} \tag{3.31}$$

The corresponding motor angular velocity ω_{ai} is obtained using the below-mentioned relation.

$$\omega_{ai} = {G_i}^{+} M \tag{3.32}$$

Note that the condition $\omega_i = 0$ is enforced when ω_{ai} is computed using Eq. (3.32).

Furthermore, to maintain the torque and thrust constraints, the matrix Gi is modified as $G_{Wi} = W_s G_i$ while implementing Eq. (3.32) on the autopilot hardware. Now, G_{Wi} is the Weighted Control Allocation (WCA) matrix.

3.3 Achievable control authority after control re-allocation

In Figs. (3.1 - 3.8), the upper and lower limits of a healthy case are compared to the reduced set of control achieved with our FTC module in the case of motor 1 failure. These Control Authority graphs show the range of Torque $(\tau_\phi, \tau_\theta, \tau_\psi)$ and Thrust output acting along the body axis, with given Roll, Pitch inputs. The upper and lower limits of control authority are plotted and saturation can be seen on all different control inputs due to changes in the system dynamics. In these figures, the roll and pitch inputs are given in a {-5, 5} range and the roll, pitch, yaw, and thrust outputs are mapped. It is observed that the control authority changes differently for each axis after the fault is corrected and the UAV is stabilized.

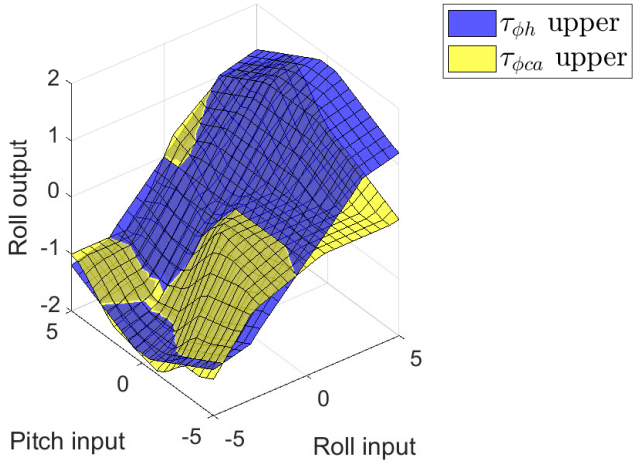

Figure 3.1: Upper limits of Roll output achieved with a given roll and pitch input. $\tau_{\phi h}$ represents healthy condition Roll, while $\tau_{\phi ca}$ represents Roll after fault occurs in motor 1 and control allocation is used.

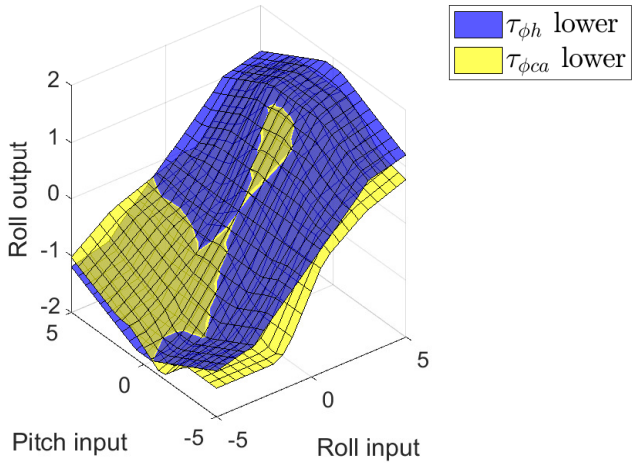

Figure 3.2: Lower limits of Roll output achieved with a given roll and pitch input. $\tau_{\phi h}$ represents healthy condition Roll, while $\tau_{\phi ca}$ represents Roll after fault occurs in motor 1 and control allocation is used.

In Fig. (3.1, 3.2), it can be seen that roll control authority is slightly lost, since the ability to control roll axis is reduced due to failure in motor 1, and the diagonally opposite motor has to compensate for the same amount of roll.

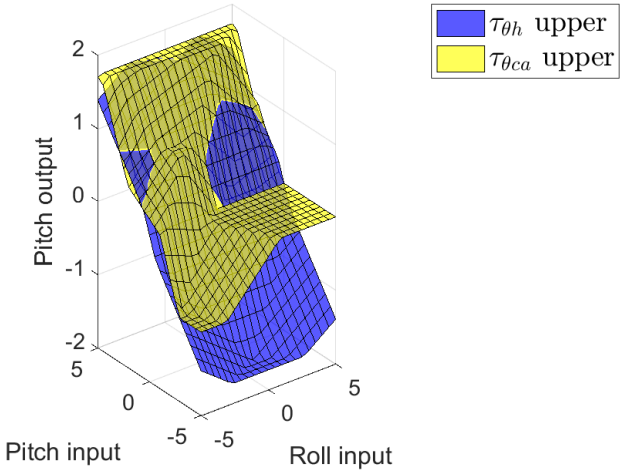

Figure 3.3: Upper limits of Pitch output achieved with a given roll and pitch input. $\tau_{\theta h}$ represents healthy condition Pitch, while $\tau_{\theta ca}$ represents Pitch after fault occurs in motor 1 and control allocation is used.

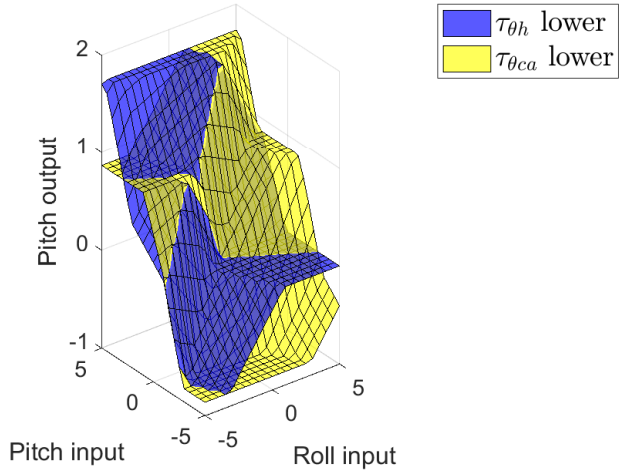

Figure 3.4: Lower limits of Pitch output achieved with a given roll and pitch input. $\tau_{\theta h}$ represents healthy condition Pitch, while $\tau_{\theta ca}$ represents Pitch after fault occurs in motor 1 and control allocation is used.

In Fig. (3.3, 3.4), it can be seen that pitch control authority is lost heavily when roll input is non-zero.

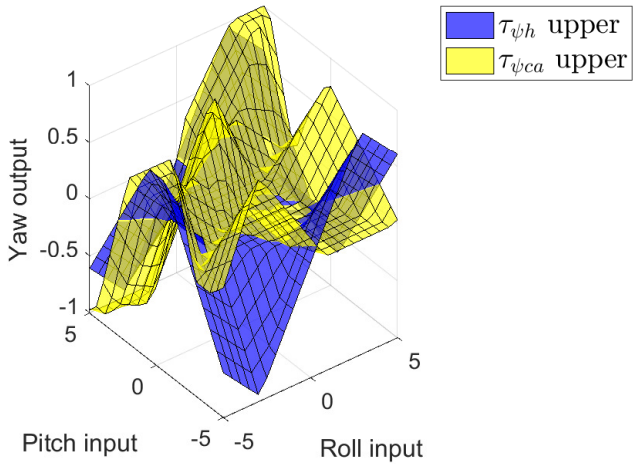

Figure 3.5: Upper limits of Yaw output achieved with a given roll and pitch input. $\tau_{\psi h}$ represents healthy condition Yaw, while $\tau_{\psi ca}$ represents Yaw after fault occurs in motor 1 and control allocation is used.

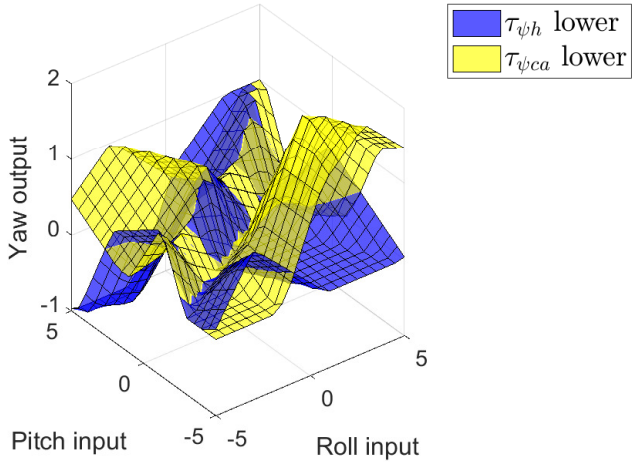

Figure 3.6: Lower limits of Yaw output achieved with a given roll and pitch input. $\tau_{\psi h}$ represents healthy condition Yaw, while $\tau_{\psi ca}$ represents Yaw after fault occurs in motor 1 and control allocation is used.

In Fig. (3.5, 3.6), it can be seen that the control authority of the yaw is heavily skewed on motor 1 failure. This occurs since τ_{ψ} is left unbalanced as discussed in Eq. (3.16).

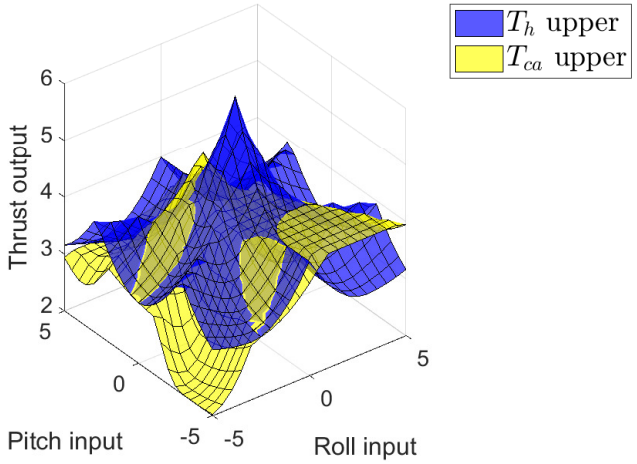

Figure 3.7: Upper limits of Thrust output achieved with a given roll and pitch input. T_h represents healthy condition Thrust, while T_{ca} represents Thrust after fault occurs in motor 1 and control allocation is used.

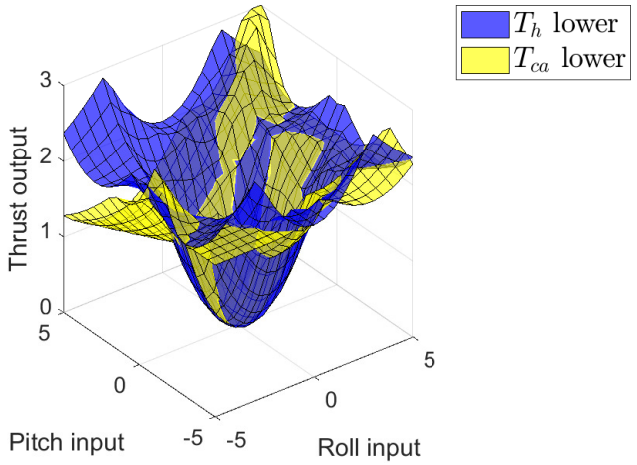

Figure 3.8: Lower limits of Thrust output achieved with a given roll and pitch input. T_h represents healthy condition Thrust, while T_{ca} represents Thrust after fault occurs in motor 1 and control allocation is used.

In Fig. (3.7, 3.8), it can be seen that thrust control authority is minimally affected. This verifies that the rest of the healthy motors can achieve enough thrust to stay in hover condition.

3.4 Implementation of Fault Tolerant Control on Experimental Hexa copter

This section gives details on the implementation of software, the hexacopter (hardware) used, and details of the flight tests.

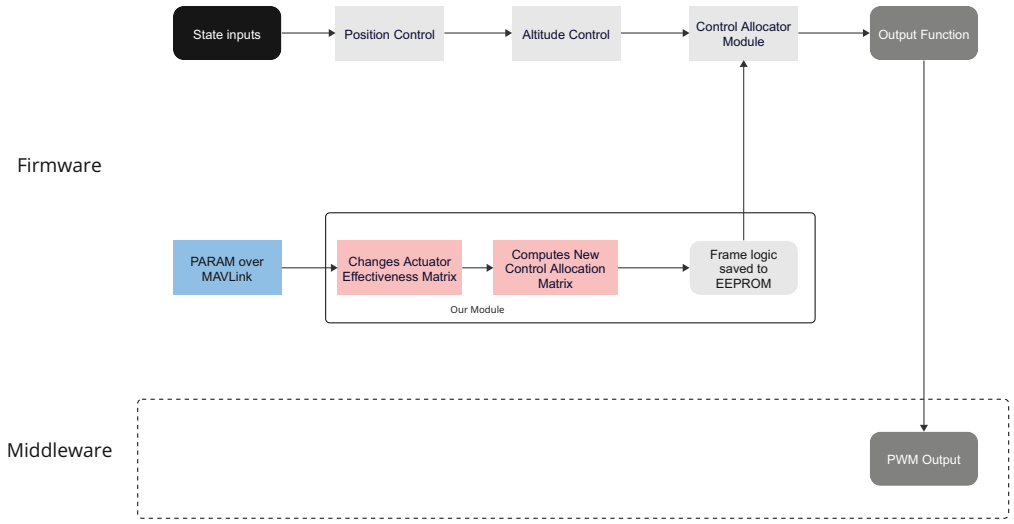

Figure 3.9: Module diagram showing how FTC module works on the autopilot firmware

3.4.1 Experimental Setup

A hexacopter is used with parameters given in Table (3.1) for flight trials. PX4 autopilot firmware v1.13 is modified to implement dynamic control reallocation as described in Fig. (3.9). Software introduces failures by setting parameters via a ground control station (GCS). The W_s matrix used for computing G_{Wi} is selected as a diagonal matrix given by $W_s = diag(0.3, 0.3, 0.15, 0.25)$. Here, the W_s matrix elements represent the maximum allowed control input normalized between 0 to 1, contributing to roll, pitch, yaw axis and to the total thrust.

Table 3.1: Hexacopter parameters

Parameter	Description	Value
Jxx	Moment of Inertia about body frame's x-axis	$6.89\text{x}10^{-3}$ kg.m^2
Jyy	Moment of Inertia about body frame's y-axis	$6.89\text{x}10^{-3}$ kg.m^2
Jzz	Moment of Inertia about body frame's z-axis	$3.44\text{x}10^{-2}$ kg.m^2
l	Arm length	0.33 m
Jr	Rotor inertia	$6\text{x}10^{-4}$ kg.m^2
m	Mass of hexacopter	0.95 kg
k_T	Aerodynamic force constant	$3.13\text{x}10^{-4}$ N.s^2
k_M	Aerodynamic moment constant	$7.5\text{x}10^{-6}$ Nm.s^2
Rmot	Motor circuit resistance	0.6 Ω
Kmot	Motor torque constant	5.2 mNm/A

Table 3.2: PID Tuning parameters

Parameter	Value
k_{Pz}	4.0
k_{Dz}	0.1
k_{Iz}	2.0
$k_{P\phi}$;$k_{P\theta}$	6.5
$k_{P\psi}$	2.8
$k_{D\phi}$;$k_{D\theta}$;$k_{D\psi}$	6.5
$k_{I\phi}$;$k_{I\theta}$;$k_{I\psi}$	2.8

The PID parameters used to generate the control input M are shown in Table (3.2) and are selected after tuning the closed-loop hexacopter model in the Gazebo simulator. Closed-loop sampling time, T_s = 10 ms.

3.4.2 Outdoor flight testing and dataset generation for analysis

(a) 2s before fault (b) Fault injected (c) 2s after fault

Figure 3.10: Demonstration of outdoor stabilization after a complete failure of Motor 1. (Yellow encircled propeller). Note that the encircled propeller is stationary in sub-figures (b) and (c).

Multiple tests have been conducted outdoors to validate the proposed fault-tolerant architecture on a real hexacopter. During flight trials, the hexacopter is launched to $1-3$ meters from the ground and set to altitude hold mode. A few seconds later, motor fault(s) is introduced, and the dynamic control reallocation is enabled. It is observed that the UAV can stabilize and perform a safe landing for all possible single motor failures[1], and some configurations for two motor failures[2]. It is noteworthy that the classifier (FDI) output is only useful for the 1st fault occurrence; hence the 2 motor failure cases are tested by immediately injecting 2 faults consecutively. Photograph of UAV flight for a single motor failure is shown in Fig (3.10). Here, the encircled motor is undergoing failure during flight. A dataset containing sensor and actuator data of outdoor flight results is published in the open-source domain to validate the same[3].

3.5 Results

We have conducted multiple flight trials on all six motors for the proposed dynamic control reallocation method to verify the Fault Tolerant Control (FTC) solution for the UAV platform as mentioned earlier. Regarding performance constraints, sensor sampling delay, CPU load, and RAM usage are verified. The onboard Arm Cortex-M7 processor(STM32F765, 216MHz, 32-Bit) can handle CPU loads and increased RAM consumption without reducing its operating frequency or causing disruption. We have conducted extensive experiments to evaluate various control and performance aspects of the UAV platform equipped with our fault-tolerant strategy, which is summarized below:

- Motor angular velocities before and after fault.
- Flight performance - tracking error, percentage overshoot, settling time.

[1]Can be viewed at https://youtube.com/watch?v=z1vDjloeLiw
[2]Can be viewed at https://youtube.com/watch?v=p_mJ04ftzE0
[3]Can be downloaded at https://doi.org/10.5281/zenodo.7642240

- Control effort and estimated endurance.
- System characteristics - changes in CPU load, RAM usage and closed-loop sampling time.

3.5.1 Motor angular velocities before and after fault:

Fig. (3.11) shows the motor angular velocity ($\omega_1 ... \omega_6$) scaled to 0 to 1. Note that fault is introduced around 14s to motor 1, i.e. $\omega_1 = 0$. Failure of motor 1 leads to a sudden significant drop in ω_2 (motor located symmetrically opposite to motor 1). In the rest of the flight, ω_2 remains lower than the other healthy motors, i.e., ω_3 to ω_6. This is mainly for compensating the unbalanced rolling moment caused due to the result of ω_1 going to zero. Similarly, Fig. (3.12) shows motor angular velocities for the case of two motor failure.

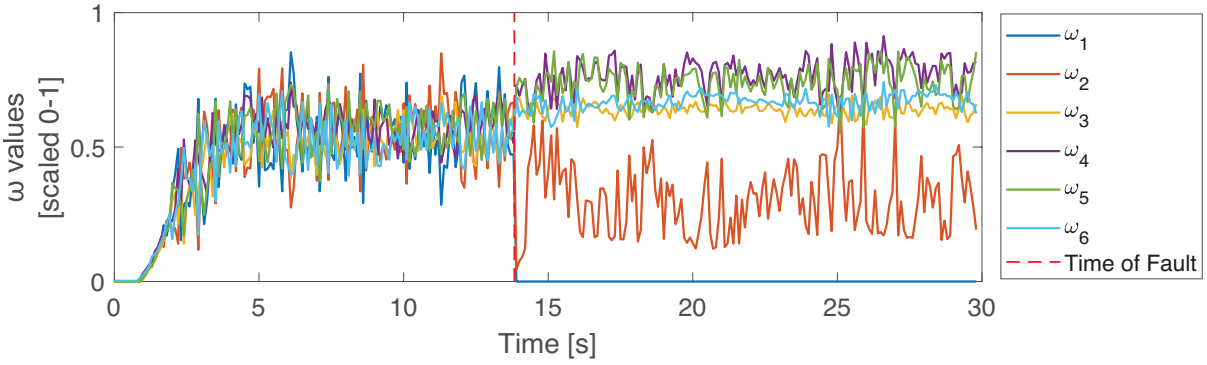

Figure 3.11: Motor outputs after introducing a hard fault (failure) on Motor 1 at 14s

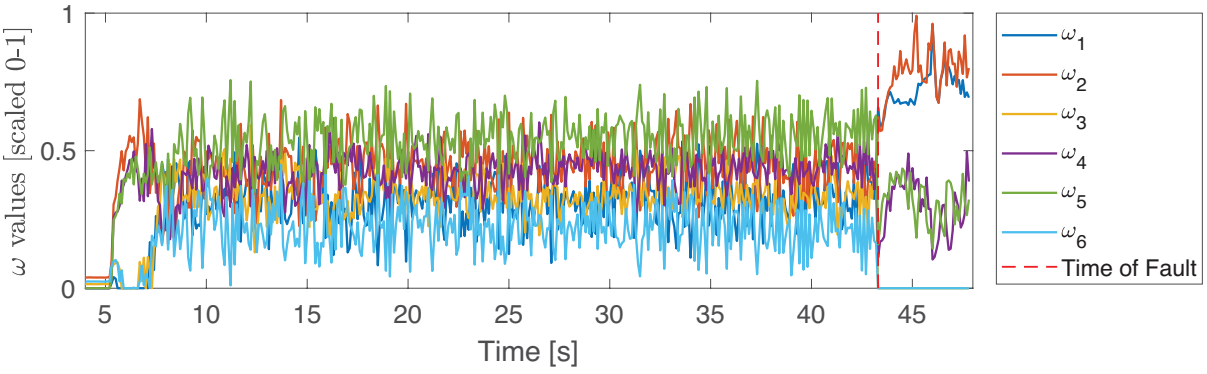

Figure 3.12: Motor outputs after introducing hard faults (failures) in Motor 3 and Motor 6 at 43s

3.5.2 Flight performance - tracking error, percentage overshoot, settling time:

Motor failure immediately affects flight performance, leading to asymmetry regarding the control inputs. Dynamic control reallocation reduces the effects caused by asymmetric control inputs. Fig.

(3.13, 3.14) compares UAV state variables 4 seconds before and after the occurrence of the fault in single motor scenario. Please note that the system response settles within the 4 second time period after the occurrence of the fault. The blue curve in Fig. (3.13, 3.14) denotes the trajectory of the state variables before fault. In contrast, the orange colour shows the trajectory of the state variables after the fault has occurred. Before the occurrence of a fault, the state variables are maintained around the equilibrium point as given in Eq (3.1, 3.2). After the fault occurs, the corresponding immediate change in the state variables, percentage overshoot, and the settling time of the deviation from the equilibrium conditions are given in Table (3.3).

The peak values among angular velocity and Euler angles are observed for ω_x and ϕ, respectively. The pitch attitude dynamics are least affected by the fault owing to the location of the motors M1 and M2 with respect to the pitch axis y_B. Overall, there is only a small difference in the magnitude of the state variables when comparing before and after the fault, especially after settling time. Yaw attitude dynamics settle at the slowest rate. Changes in altitude dynamics are negligible. We can see the peak values among angular rates in Fig. (3.13). An error in the orientation (roll, pitch) is seen - the magnitude is smaller in the pitch axis since motor M1 does not have any effect on pitch as seen in Fig. (1.4).

Table 3.3: Effect of motor fault on flight performance

Motor no	1	2	3	4	5	6	3, 6
$\Delta\phi$ **(rad)**	-0.4	-0.7	0.7	0.04	0.39	0.08	0.8
$\Delta\theta$ **(rad)**	0.25	0.11	-0.25	-0.52	0.17	-0.93	-0.84
$\Delta\psi$ **(ad)**	0.0	0.07	0.0	0.0	0.01	0.16	0.75
Δz **(m)**	0.16	0.11	0.06	0.03	0.04	0.02	0.35
Δp **(rad/s)**	0.6	0.51	0.27	-0.11	0.44	0.56	0.29
Δq **(rad/s)**	0.11	-0.02	-0.14	-0.17	0.0	0.11	0.1
Δr **(rad/s)**	0.0	-0.08	0.0	0.0	0.01	-0.02	0.6
% overshoot ϕ	298.62%	303.24%	103.86%	150.42%	103.1%	172.59%	263.35%
% overshoot θ	160.27%	121.49%	157.46%	16.54%	176.21%	11.16%	48.21%
% overshoot ψ	1.95%	3.23%	2.3%	0.24%	0.13%	2.86%	2.9%
% overshoot z	9.26%	6.45%	3.56%	2.07%	2.17%	1.12%	25.48%
Settling time (s)	6.8s	6.85s	2.75s	8.4s	3.83s	8.95s	-

Table (3.3) is generated by taking an average of two test flights for each motor failure. The immediate change of any state variable α is computed as shown below.

$$\Delta\alpha = \alpha(t_f + T_s) - \alpha(t_f) \tag{3.33}$$

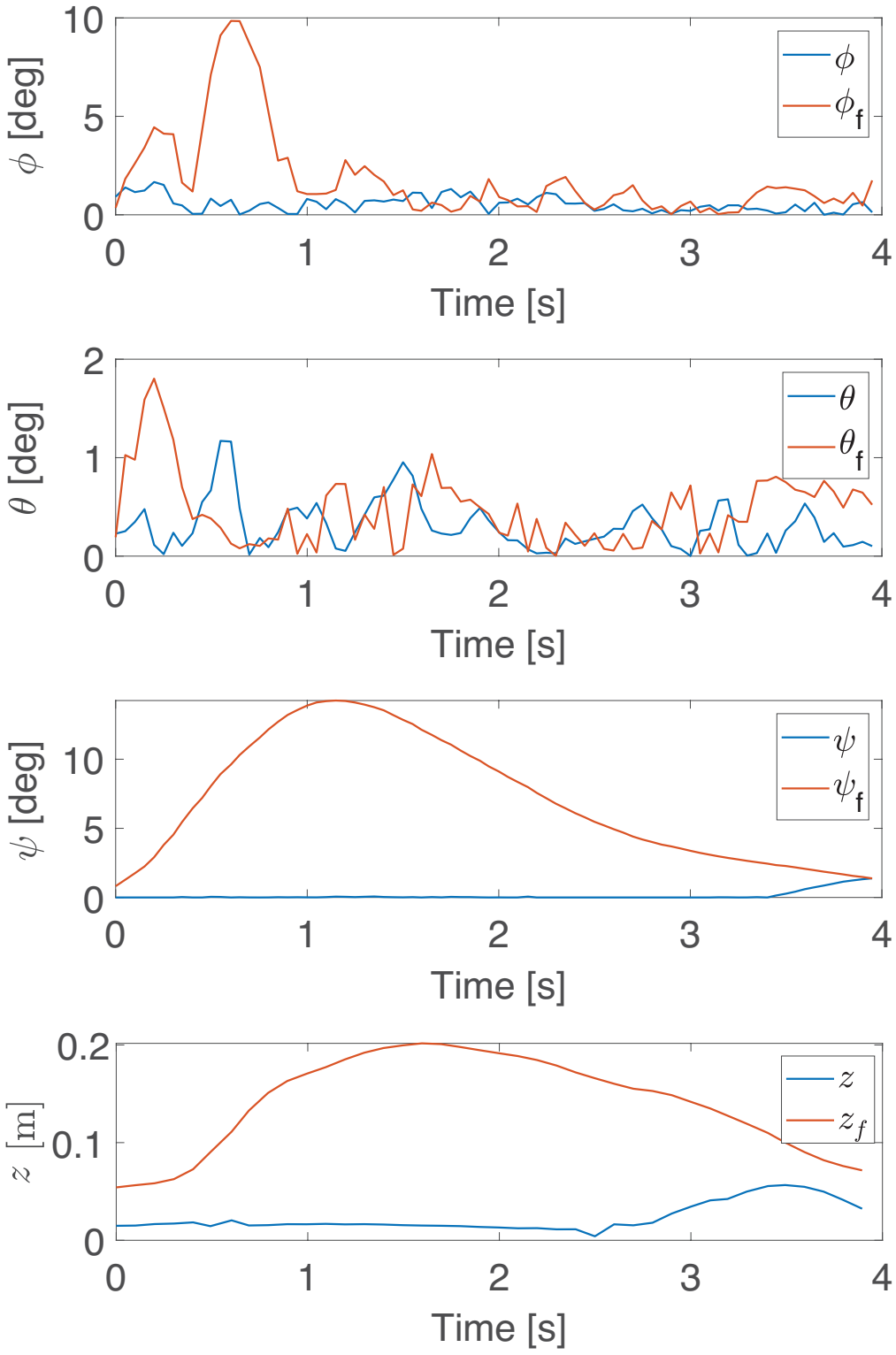

Figure 3.13: Trajectory of state variables ϕ, θ, ψ and z before and after the failure of Motor 1 for a duration of 4 seconds. Blue and orange curves denote the trajectory before and after failure, respectively.

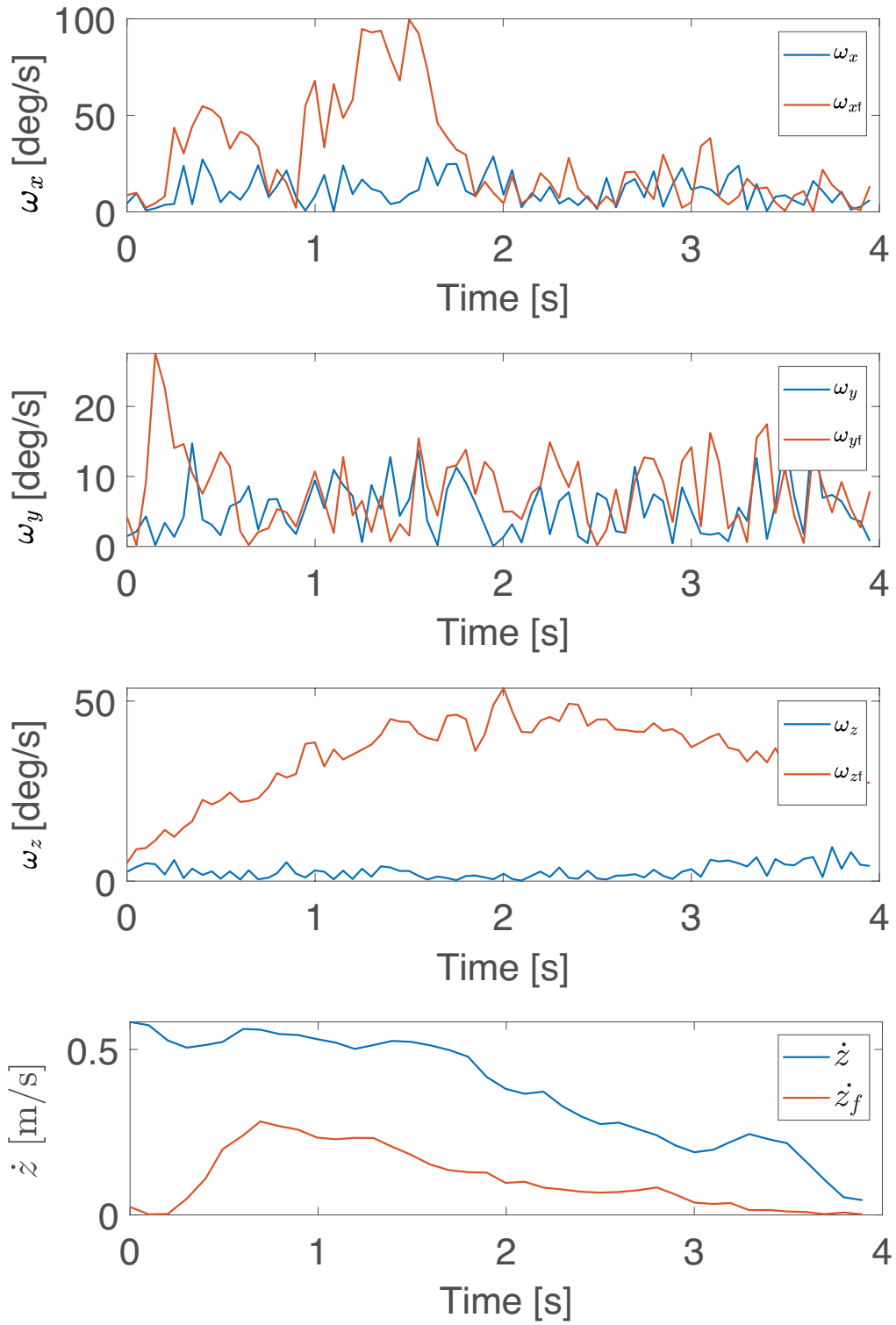

Figure 3.14: Trajectory of state variables $\omega_x, \omega_y, \omega_z$ and $\dot{z}$ before and after the failure of Motor 1 for a duration of 4 seconds. Blue and orange curves denote the trajectory before and after failure, respectively.

Where t_f is the time instant at which fault has occurred, and T_s is the closed loop sampling time or the time in which the PID control loop runs ($T_s = 10ms$).

The results of flight performance are given in Table (3.3) corresponding to the hexacopter shown in Fig. (1.4). The proposed algorithm stabilizes the UAV within a time range of 2.7 to 8.95 seconds, depending upon the location of the faulty motor with respect to the body axis. The maximum immediate change occurs in pitch angle due to the failure of motor M4 or M6. The percentage overshoot is the maximum for the failure of motor M1 or M2 on the roll axis. The failure of motors M1 or M2 results in high tracking errors and overshoot in response to the roll attitude dynamics. Similarly, failure of motors M3 or M5 results in higher tracking errors and overshoot on the pitch attitude dynamics. Still, the response reaches a steady state in a short duration compared to roll and yaw attitude dynamics. Failure in M4 or M6 results in a high overshoot on the roll attitude dynamics compared to the pitch axis. In case of two motor failure scenario (M3, M6), the an immediate drop in altitude occurs, since the thrust generated by remaining motors is not enough to compensate for the total weight of the vehicle.

3.5.3 Control effort and estimated endurance:

Motor failure directly affects the control effort used for stabilization. Here, the control effort (C_e) is computed below.

$$C_e = \sum_{i=1}^{6} {\omega_i}^2 \tag{3.34}$$

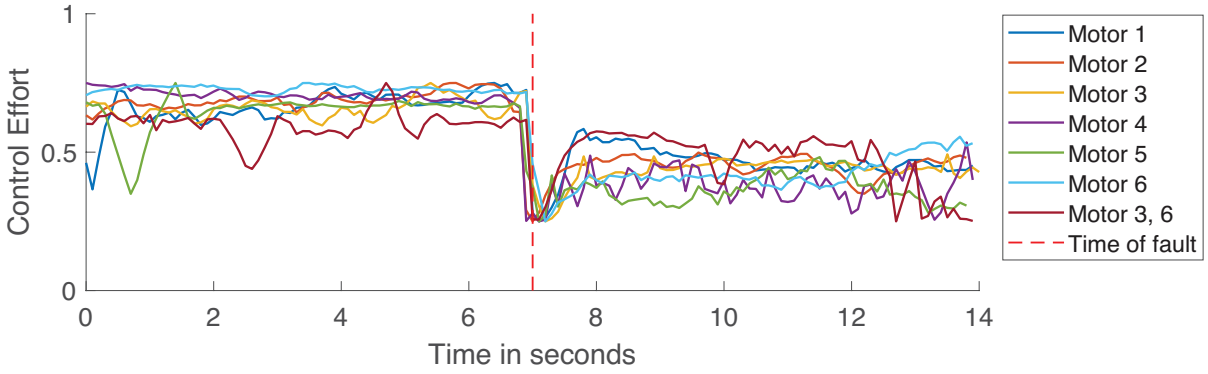

Figure 3.15: Control effort for all motors cases after fault occurs at 7s, in a 14s window.

Fig. (3.15) shows the control effort scaled between 0 to 1 for all the six single motor failure cases and one case for two motor failure. For all the cases, the control effort goes down momentarily after the fault occurs and then increases later. This is evident from Fig. (3), where $\omega_1 = 0$ and the magnitude of ω_2 reduces to compensate for unbalanced rolling torque. This results in the reduction of C_e, even if there is an increase in the magnitude of ω_3 to ω_6 after the fault occurs. This also leads to a reduction in flight performance in the post-fault scenario compared to the pre-fault scenario.

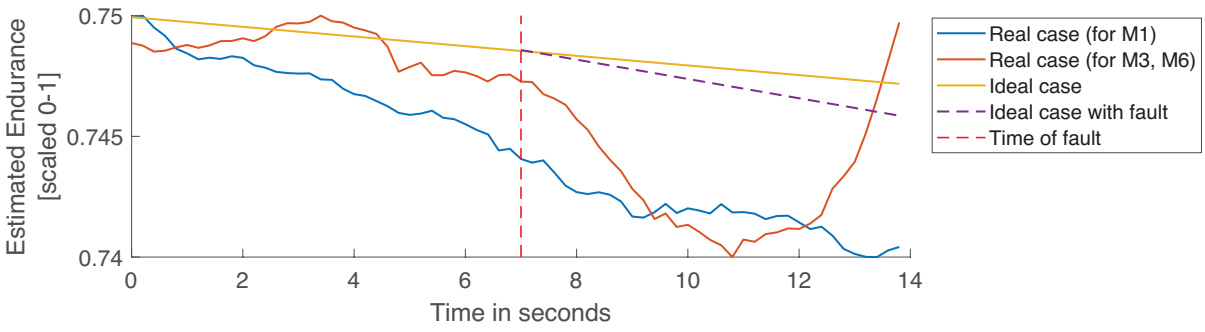

Figure 3.16: Estimated Endurance of the UAV when motor failures occur at 7s, in a 14s window.

Fig. (3.16) shows the estimated endurance of the UAV for a single flight test where failure was introduced for motor M1, in ideal case and in real case. It also shows estimated endurance in case of failure of 2 motors (M3 and M6). The endurance (E) in hours is calculated from Eq (3.35).

$$E = C \times \frac{V_B}{I_C \times V_C} \tag{3.35}$$

Where C represents the rated battery capacity (in mAh), V_B is the rated battery voltage (in V), V_C is the instantaneous battery voltage (in V), and I_C is the instantaneous current consumed (in mA). For the ideal case, we assume $V_B = V_C$. But for the real case, $V_C < V_B$ due to non-ideal battery characteristics. Here, $Ic \times Vc$ is the instantaneous power consumption (in Watt). Due to increased power consumption, the battery life decreases after the fault occurs. This is observed for both ideal and real cases. In Fig. (3.16), the endurance goes up after a few seconds in case of two motor failure scenario, since the UAV has already safely landed (within 4-5 seconds of fault).

3.5.4 System characteristics - changes in CPU load, RAM usage and closed-loop sampling time:

Performance constraints such as sensor sampling delay, CPU load and RAM usage are verified. The onboard Arm Cortex-M7 processor(STM32F765, 216MHz, 32-Bit) can handle CPU loads and increased RAM consumption without reducing its operating frequency or causing disruption. Motor failure is implemented by computing the control matrix as described in Eq (3.32). This matrix reconfiguration causes a momentary increase in the computational load of the CPU and causes an increase in the closed loop sampling time. No relevant effects on the RAM were observed.

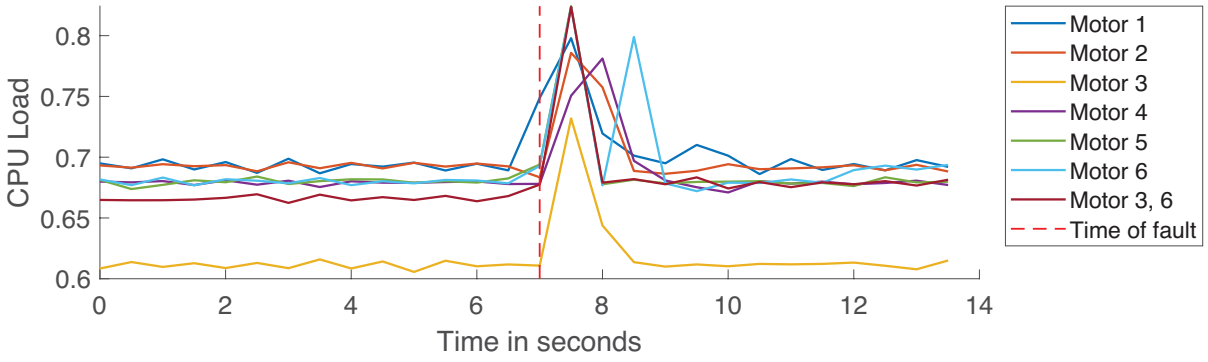

Figure 3.17: Comparison of CPU Loads in a 7-second window before and after a fault for all motor failure cases.

Fig. (3.17) shows the CPU's computational load increase across multiple flight tests for the i^{th} motor failure.

Table 3.4: Effect of reconfiguration algorithm on the performance of the CPU computational load and closed-loop sampling time

Motor no.	CPU load increase	Closed-loop sampling delay
1	10.7%	+8%
2	10.2%	+9%
3	12%	+5%
4	14.5%	+11%
5	8%	+3.9%
6	14.35%	+13%
3,6	16%	+11%

In Table 3.4, Fig. (3.17) CPU load is tracked within a few milliseconds of fault introduction. Due to the added matrix computations, an apparent increase is seen in the CPU load. Different motor reconfigurations (or failures in different motors) have minor differences in the amount of CPU load due to the positioning of the faulty motor. The amount of correction required for each axis is different for different fault cases, as explained in Section 5.2 using Table 3. An apparent lag in the closed-loop response (a few microseconds) is observed immediately after the fault introduction. The closed-loop response delay increases as the CPU load increases. The computational load increases momentarily by an average of 10% for 10 ms. The increase in load is caused by the increase in settling time because it takes more time to reach stability. The total RAM consumption does not change in any of the cases. CPU usage returns to normal once the matrix calculations are completed. The motor placement considerably affects the flight performance, but only slightly to CPU load.

3.6 Conclusion & Inference

In this chapter, we presented the Fault Tolerant Control (FTC) module that subscribes to fault prediction results from the Fault Detection and Isolation (FDI) module and is able to successfully stabilize the vehicle after single motor faults. We have also tested and analyzed cases of 2 motor failures, where the UAV has to be landed immediately. We also analyzed the system response after the fault is stabilized and compared it to that of the healthy case, without any motor faults.

Chapter 4

Integration of Fault Detection & Isolation (FDI) and Fault Tolerant Control (FTC) modules for Hexacopter UAV

This chapter presents a comprehensive evaluation of the proposed methodologies for Fault Detection and Isolation (FDI) and Fault Tolerant Control (FTC) in the event of complete actuator loss in Hexacopter UAVs. Here, we assess the performance of the Rotation Forest Classifier in accurately detecting and isolating the occurrence of faults and examine the effectiveness of the Dynamic Control Reallocation strategy in maintaining UAV stability post fault identification. By combining these two components, our aim is to illustrate a cohesive, end-to-end fault management solution. This section demonstrates the practical applicability of the methods developed through rigorous testing and validation scenarios, providing insights into their potential in enhancing UAV operational reliability and safety in real-world applications. It represents a critical juncture where theory meets application, solidifying the relevance and impact of our research.

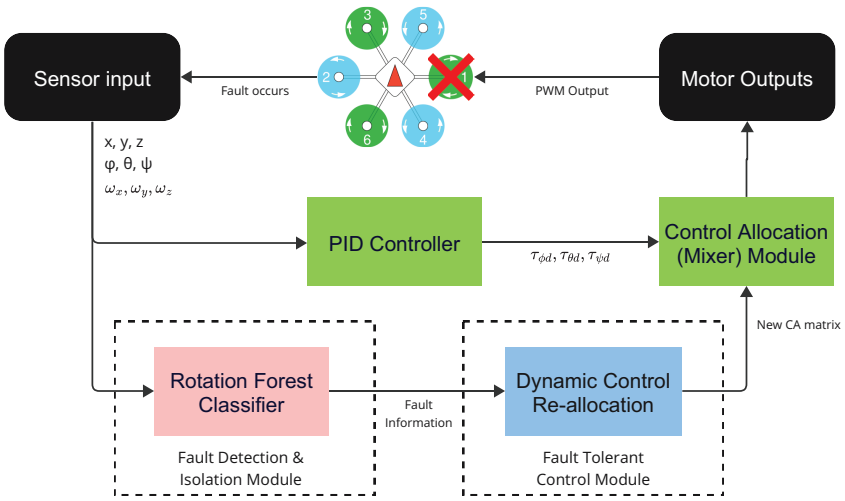

Figure 4.1: Proposed architecture of the integrated FDI and FTC modules. Here, the red cross denotes the motor under failure.

Fig. (4.1) shows the module diagram with details of our combined FDI and FTC implementation. Here, the FDI module runs in the background while the UAV performs its missions. On occurrence of the fault, the FDI module detects, classifies, and reports the fault. This FDI module output is taken in the FTC module, and correction is performed to stabilize the UAV.

4.1 Simulation Results for integrated FDI and FTC modules

Simulation is carried out in order to validate the proposed FDI and FTC modules together and analyze the system performance. The vehicle model used is a Hexacopter UAV as described in Fig. (1.4) and Tables (3.1, 3.2). We start the simulation in hover condition and then give a motor fault in Motor 1 at 2 seconds. The results are compared between our proposed FDI + FTC module and the case of immediate control reallocation. In the immediate case, the drone stabilizes as soon as a fault is introduced, whereas in the case of our module, the fault is introduced and then stabilized only after the FDI module classifies and isolates the fault to the failing motor. The simulation implementation is done in Python using the DOP853 ODE algorithm.

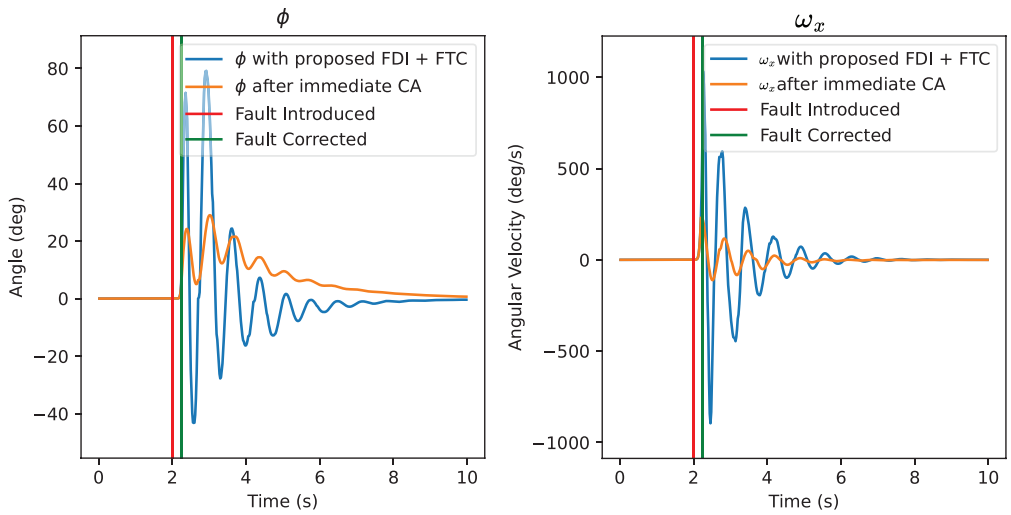

Figure 4.2: State variables ϕ and ω_x response after Motor 1 fault introduced at 2s, detected and classified at 2.2s and corrected (stabilized) at 2.25s (blue). Orange line shows response after fault introduced at 2s and immediately corrected (stabilized).

Fig. (4.2) shows that motor 1 failure has a huge effect on the roll axis, due to vehicle symmetry as shown in Fig. (1.4). This effect is minimal if the fault is immediately corrected with control allocation; however, real-world scenarios will have a delay for the detection and input/output delays.

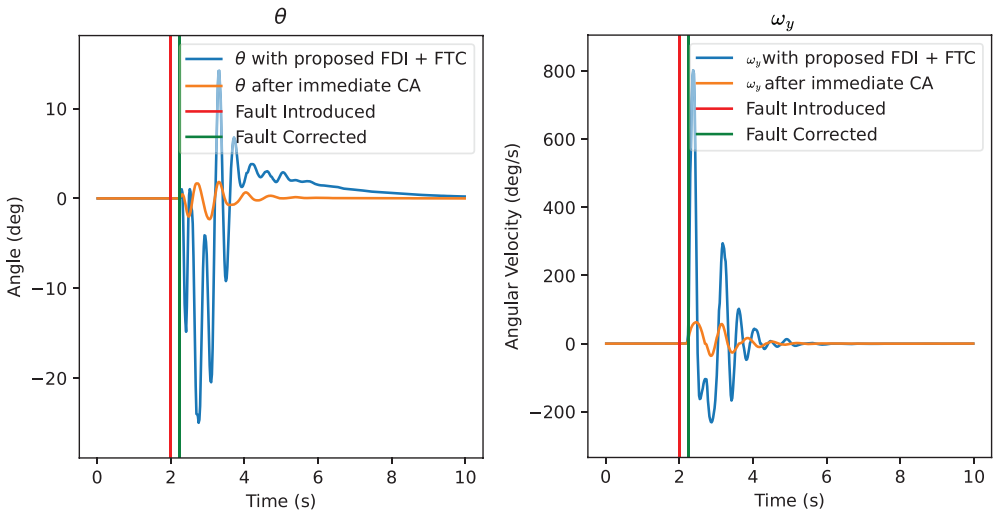

Figure 4.3: State variables θ and ω_y response after Motor 1 fault introduced at 2s, detected and classified at 2.2s and corrected (stabilized) at 2.25s (blue). Orange line shows response after fault introduced at 2s and immediately corrected (stabilized).

Fig. (4.3) shows that motor 1 failure has a negligible effect on the pitch axis. This can be verified intuitively in Fig. (1.4).

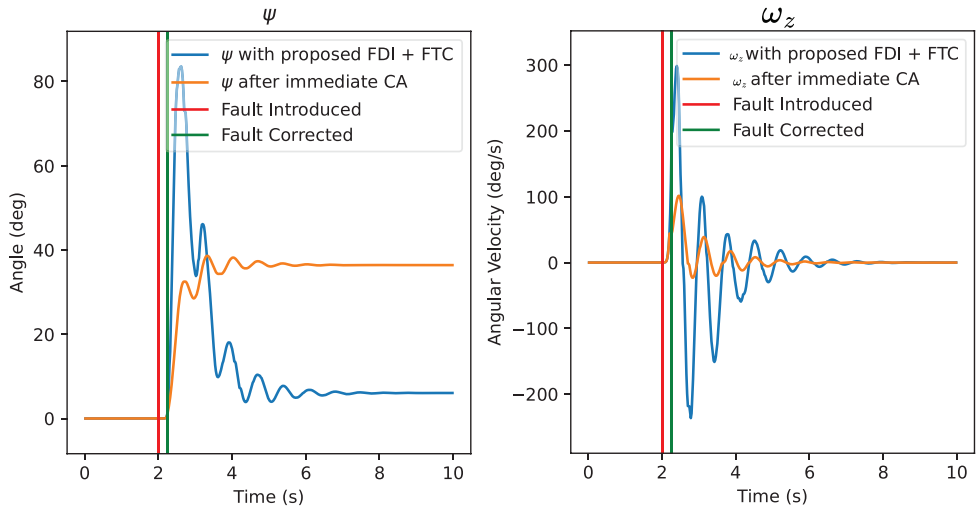

Figure 4.4: State variables ψ and ω_z response after Motor 1 fault introduced at 2s, detected and classified at 2.2s and corrected (stabilized) at 2.25s (blue). Orange line shows response after fault introduced at 2s and immediately corrected (stabilized).

Fig. (4.4) shows that motor 1 failure has a sudden shift in the yaw axis. This shift, however, does not cause any damage to the UAV, since it is only rotating around its yaw axis and the heading (direction)

changes. This effect is reduced in real scenarios where the PID controller is tuned to compensate for such a drift.

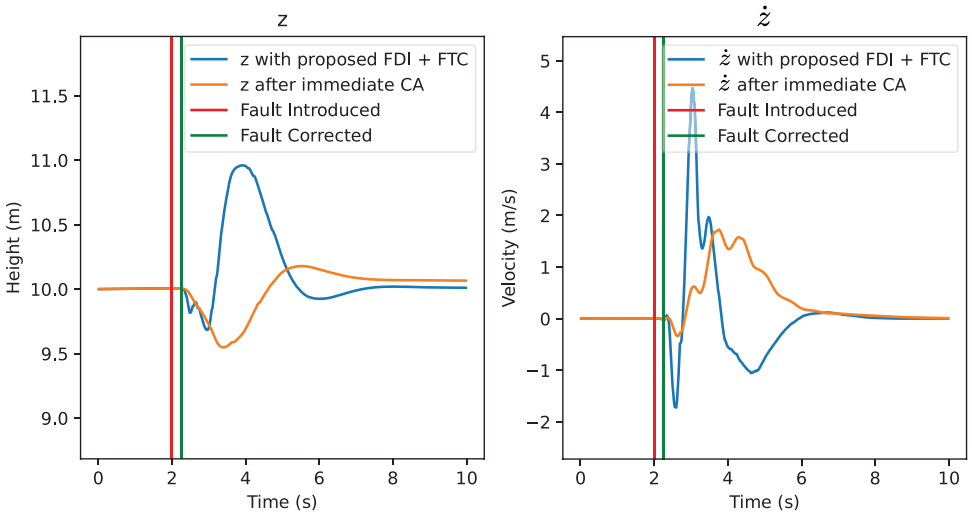

Figure 4.5: State variables z and $\dot{z}$ response after Motor 1 fault introduced at 2s, detected and classified at 2.2s and corrected (stabilized) at 2.25s (blue). Orange line shows response after fault introduced at 2s and immediately corrected (stabilized).

Fig. (4.5) shows that motor 1 failure has a varying effect on vehicle altitude (z). If stabilization is performed immediately, the change is close to 5% of the current altitude of the vehicle. This change increases to 10% for the proposed integrated FDI and FTC modules, as we have a delay of 250 ms.

Table 4.1: Difference in flight performance, compared between our FDI + FTC module and that in which control reallocation is done immediately after fault occurrence.

$\Delta\phi$ (deg)	70
$\Delta\theta$ (deg)	-14
$\Delta\psi$ (deg)	50.2
Δz (m)	1.3
$\Delta\omega_x$ (deg/s)	805
$\Delta\omega_y$ (deg/s)	744
$\Delta\omega_z$ (deg/s)	199
% overshoot ϕ	366.1%
% overshoot θ	-250%
% overshoot ψ	411.3%
% overshoot z	12.4%
Settling time (s)	3.5s

Table. (4.1) shows the performance comparison between our FDI + FTC module and that in which control reallocation is done immediately after the occurrence of the fault. All values listed in this table are the differences between the two methods of comparison. Note that the difference in rates and overshoot is higher since the simulation does not closely emulate the motor behavior due to slew rates and a difference in PID tracking performance. Here, the delta values are the immediate change in the next time sample, while the percentage overshoot represents the maximum difference in the two values before settling down.

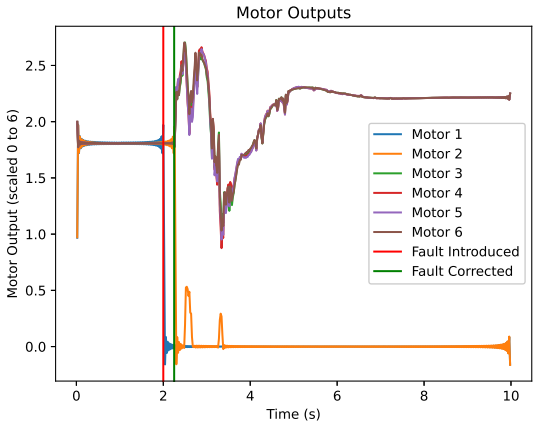

Figure 4.6: Motor outputs showing how stabilization works in simulation.

Fig. (4.6) shows motor output velocities. Fault is introduced at 2s and Motor 1 velocity ω_1 can be seen dropping to 0. After fault detection at 2.25 seconds, the diagonally opposite motor (Motor 2) is turned off due to control reallocation, and other motor velocities (ω_2 to ω_6) start to stabilize. Eventually, the total thrust output of the motors is increased to compensate for motor failure.

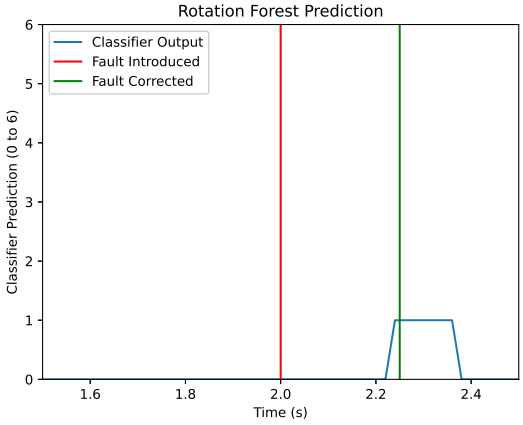

Figure 4.7: Proposed architecture of the integrated FDI and FTC modules.

Fig. (4.7) shows the predictions of the Rotation Forest classifier. Prediction value 0 refers to a healthy state without fault, while values 1 to 6 refer to Motor 1 to Motor 6 fault. Shortly after the introduction of the fault at 2s, the classifier output changes to 1, referring to a fault in Motor 1.

4.2 Conclusion & Inference

In this chapter, we presented the two modules for Fault Detection & Isolation (FDI) and Fault Tolerant Control (FTC) combined together and analyze their performance compared to the case where the FTC module is run directly after fault injection. With our proposed method, we demonstrate a real-world solution for detecting a fault and then correcting it by stabilizing the UAV. Factors such as input/output delay, classifier operational speed, and sampling frequency can cause a delay between the fault occurrence and its correction. By integrating the modules, we demonstrate the delay after fault occurrence and it's impact on the UAV dynamics. For example, the state variables start to oscillate as soon as the fault is introduced and can cause the UAV to crash if the delay is too high (above 400 ms in our simulations). It is observed that the FDI module is able to detect and classify the fault within 250 ms and hence allows the FTC module to stabilize the UAV before it rolls, tips over, or crashes. This is an acceptable delay in the response, since existing literature considers 500 ms as a pessimistic value for the delay [31]. In real-time, however, the response would be faster due to the PID controller tuning, motor slew rates, and higher sampling frequency.

4.2.1 Open-source results

Some of the work done related to this thesis has been published in the open-source domain:

- Code repository used to generation of simulation datasets.[1]
- Dataset repository containing simulation & real flight results datasets for multiple use-cases.[2]
- Dataset repository used for the FDI module(simulation and real data for training and validation).[3]
- Dataset generated after running the FTC module.[4]

[1] GitHub: https://github.com/Embedded-Fault-Tolerant-Control/rrc_fault_tolerant_control

[2] GitHub: https://github.com/Embedded-Fault-Tolerant-Control/rrc_ftc_datasets

[3] GitHub: https://github.com/Embedded-Fault-Tolerant-Control/RotationForest-FDI-Dataset

[4] GitHub: https://github.com/Embedded-Fault-Tolerant-Control/WCA-FTC-Dataset

Chapter 5

Conclusions

Existing fault management methods for UAVs face significant challenges in terms of real-time fault detection, isolation, and subsequent control reallocation. The high likelihood of UAV crashes following motor failures, especially in real flight scenarios, is a pressing issue that has yet to be satisfactorily addressed. Our research presents a comprehensive solution that not only mitigates these issues, but also demonstrates practical applicability and effectiveness.

Our proposed method integrates a Rotation Forest classifier for Fault Detection and Isolation (FDI) and a dynamic control reallocation strategy for Fault Tolerant Control (FTC). The classifier achieves high accuracy with a true positive rate of 92. 6%, and, remarkably, the fault is identified and classified within 60 to 180 ms of its occurrence. This quick and accurate response is crucial to trigger subsequent fault management actions.

In real and simulated environments, our method proved successful in managing single motor faults, significantly enhancing the resilience of UAV operations. For all cases of single motor failure, the vehicle is able to continue the mission as defined; i.e. waypoint navigation, landing at a point. In a limited number of cases, it also demonstrated effectiveness in managing two simultaneous motor failures. Specifically, in case of faults in motors 3, 6 or motors 4, 5; our proposed method can be used to safely land the vehicle. In other cases, simultaneous failure in two motors leads to unstable behaviour and results in a crash. However, comprehensive testing and optimization for all scenarios involving dual motor faults remain a subject for future research.

The handling of triple motor faults, an extreme scenario, presents another challenge that has yet to be fully addressed and calls for further analysis and algorithm development. The implications of our work are profound. Although we observed changes in UAV performance and a slight loss in tracking efficiency post fault occurrence, our method successfully achieves mid-flight stabilization, drastically reducing the chances of UAV crashes. Our work provides a solid foundation for future research aiming to enhance the robustness and safety of UAV operations, particularly in fault-prone or critical environments.

Future Work

- Analysis of 2 motor failure cases, its system response and performance. For some cases, real test flights have already been conducted.
- Analysis of 3 motor failure cases.
- Extension of our work to Quadcopters, where one of the axes will have to be sacrificed in order to avoid crashing.

Related Publications

- Aditya M., Mayank S., Munjaal B., Prudhvi T., Harikumar K, Deepak G. ”Fault Detection and Isolation on a Hexacopter UAV using a Two-stage classification method” **Accepted** in IEEE International Conference on Automation Science and Engineering (CASE) 2023.
- Aditya M., Mayank S., Prudhvi T., Harikumar K, Deepak G. ”Performance evaluation of Dynamic Control Reallocation on a Hexacopter for single and two rotor failure in outdoor test flights” **Submitted** in IECON 2023.

Bibliography

[1] E. Aasi, C. I. Vasile, M. Bahreinian, and C. Belta. Classification of time-series data using boosted decision trees. *CoRR*, abs/2110.00581, 2021.

[2] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions.

[3] M. Awad and R. Khanna. *Support Vector Machines for Classification*, pages 39–66. Apress, Berkeley, CA, 2015.

[4] A. Bagnall, M. Flynn, J. Large, J. Line, A. Bostrom, and G. Cawley. Is rotation forest the best classifier for problems with continuous features?

[5] M. Bodson. Evaluation of optimization methods for control allocation. *Journal of Guidance, Control, and Dynamics*, 25(4):703–711, 2002.

[6] D. Böhning. Multinomial logistic regression algorithm.

[7] S. Bouabdallah. Design and control of quadrotors with application to autonomous flying. page 155, 2007.

[8] L. Breiman. Arcing classifiers.

[9] L. Breiman. Random forests.

[10] T. F. Chan, G. H. Golub, and R. J. LeVeque. Updating formulae and a pairwise algorithm for computing sample variances.

[11] G. P. Falconì, J. Angelov, and F. Holzapfel. Adaptive fault–tolerant position control of a hexacopter subject to an unknown motor failure. *International Journal of Applied Mathematics and Computer Science*, 28(2):309–321, 2018.

[12] G. P. Falconí, J. Angelov, and F. Holzapfel. Hexacopter outdoor flight test results using adaptive control allocation subject to an unknown complete loss of one propeller. In *2016 3rd Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 373–380, 2016.

[13] J. A. Ferreira. Models under which random forests perform badly; consequences for applications, 2021.

[14] J. I. Giribet, R. S. Sanchez-Pena, and A. S. Ghersin. Analysis and design of a tilted rotor hexacopter for fault tolerance. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1555–1567, 2016.

[15] O. Härkegård. Dynamic control allocation using constrained quadratic programming. *Journal of Guidance, Control, and Dynamics*, 27(6):1028–1034, 2004.

[16] O. Hornyák and L. B. Iantovics. Adaboost algorithm could lead to weak results for data with certain characteristics. *Mathematics*, 11(8):1801, Apr 2023.

[17] Z. jun Bi, Y. quan Han, C. quan Huang, and M. Wang. Gaussian naive bayesian data classification model based on clustering algorithm. In *Proceedings of the 2019 International Conference on Modeling, Analysis, Simulation Technologies and Applications (MASTA 2019)*, pages 396–400. Atlantis Press, 2019/07.

[18] H. Mazeh, M. Saied, H. Shraim, and C. Francis. Fault-tolerant control of an hexarotor unmanned aerial vehicle applying outdoor tests and experiments. *IFAC-PapersOnLine*, 51(22):312–317, 2018. 12th IFAC Symposium on Robot Control SYROCO 2018.

[19] A.-R. Merheb, H. Noura, and F. Bateman. Active fault tolerant control of octorotor uav using dynamic control. 09 2014.

[20] A.-R. Merheb, H. Noura, and F. Bateman. Active fault tolerant control of quadrotor uav using sliding mode control. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 156–166, 2014.

[21] A. B. Musa. Comparative study on classification performance between support vector machine and logistic regression. *International Journal of Machine Learning and Cybernetics*, 4(1):13–24, Feb. 2013.

[22] D.-T. Nguyen, D. Saussié, and L. Saydy. Fault-tolerant control of a hexacopter uav based on self-scheduled control allocation. In *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 385–393, 2018.

[23] N. P. Nguyen, N. Xuan Mung, and S. K. Hong. Actuator fault detection and fault-tolerant control for hexacopter. *Sensors*, 19(21), 2019.

[24] C. D. Pose, J. I. Giribet, and A. S. Ghersin. Hexacopter fault tolerant actuator allocation analysis for optimal thrust. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 663–671, 2017.

[25] C. D. Pose, J. I. Giribet, and A. S. Ghersin. Hexacopter fault tolerant actuator allocation analysis for optimal thrust. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 663–671, 2017.

[26] R. Puchalski, A. Bondyra, W. Giernacki, and Y. Zhang. Actuator fault detection and isolation system for multirotor unmanned aerial vehicles.

[27] R. Puchalski and W. Giernacki. Uav fault detection methods, state-of-the-art. *Drones*, 6(11), 2022.

[28] J. Rodriguez, L. Kuncheva, and C. Alonso. Rotation forest: A new classifier ensemble method.

[29] M. Saied, B. Lussier, I. Fantoni, C. Francis, H. Shraim, and G. Sanahuja. Fault diagnosis and fault-tolerant control strategy for rotor failure in an octorotor. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5266–5271, 2015.

[30] M. Saied, B. Lussier, I. Fantoni, H. Shraim, and C. Francis. Fault Diagnosis and Fault-Tolerant Control of an Octorotor UAV using motors speeds measurements.

[31] M. Saied, B. Lussier, I. Fantoni, H. Shraim, and C. Francis. Active versus passive fault-tolerant control of a redundant multirotor UAV. *Aeronautical Journal -New Series-*, 124(1273):385–408, Nov. 2019.

[32] S. L. Salzberg. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16(3):235–240, Sep 1994.

[33] I. H. Sarker. Machine learning: Algorithms, real-world applications and research directions.

[34] Y. Shen, Y. Jiang, W. Liu, and Y. Liu. Multi-class adaboost elm.

[35] A. A. Soofi and A. Awan. Classification techniques in machine learning: Applications and issues. *Journal of Basic and Applied Sciences*, 13:459–465, 2017.

[36] F.-H. Wen, F.-Y. Hsiao, and J.-K. Shiau. Analysis and management of motor failures of hexacopter in hover. *Actuators*, 10(3), 2021.

[37] G. Wild, J. Murray, and G. Baxter. Exploring civil drone accidents and incidents to help prevent potential air disasters.

[38] Y. Yang, W. Wang, D. Iwakura, A. Namiki, and K. Nonami. Sliding mode control for hexacopter stabilization with motor failure. *Journal of Robotics and Mechatronics*, 28(6):936–948, 2016.

[39] H.-F. Yu, F.-L. Huang, and C.-J. Lin. Dual coordinate descent methods for logistic regression and maximum entropy models.