Leveraging Latent Temporal Features for Robust Fault Detection and Isolation in Hexarotor UAVs

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

Master of Science
in
Computer Science and Engineering
by Research

by

Shivaan Sehgal
2018111026
shivaan.sehgal@research.iiit.ac.in

International Institute of Information Technology
Hyderabad - 500 032, INDIA
May, 2024

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled **"Leveraging Latent Temporal Features for Robust Fault Detection and Isolation in Hexarotor UAVs '** by Shivaan Sehgal, has been carried out under my supervision and is not submitted elsewhere for a degree.

_____ _____

Date Adviser: Dr. Harikumar Kandath

To all my teachers, within and beyond the walls of the classroom.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. Harikumar Kandath, whose unwavering support and guidance have been instrumental in nurturing my passion for research in the interdisciplinary domain of hardware and software integration. Dr. Kandath's mentorship, insightful feedback, and encouragement have been invaluable throughout my academic journey. I am profoundly grateful for the opportunity to learn and grow under his guidance. I am also thankful to Dr. K. Madhava Krishna for his invaluable support and guidance during the formative stages of my research in robotics.

My deepest gratitude extends to Gunjan, Vikrant, KV, Mukund, and Nomaan. Their enduring support, expertise, wit, wisdom, and integrity were essential throughout my research. I would not have been able to achieve this without them. I am incredibly grateful to my wingmates: Vedant, Jay, Tanmay, Akash and Aakash, Vishal, Dhruv, Abhijit, Shikhar, Sanskar, Varun, and my part-time wingmates Mehul, Amit, Aman, Thathagath, Bharat, Ishan, and Rishab. They made my college journey an unforgettable experience filled with fun sleepless nights, great trips, and deep (sometimes directionless) discussions. I am thankful to all the seniors, batchmates, and juniors who were part of the Art Soc, Felicity, and Parliament. Working passionately for these endeavors was a source of immense joy and learning, and their support made it all worthwhile.

Finally, I am eternally grateful to my family for their unwavering support throughout my life. To my mother and father, thank you for your hard work and dedication. To my naani and nanaji, thank you for your courage and inspiration. To my mamaji, thank you for your guidance. And to my sister, thank you for always bringing a smile to my face.

Abstract

Unmanned Aerial Vehicles (UAVs), commonly known as drones, have revolutionized various sectors including surveillance, agriculture, and disaster management due to their versatility and maneuverability. Hexacopter UAVs, equipped with six rotors, offer enhanced stability and payload capacity compared to their quadcopter counterparts. However, their operational effectiveness is contingent upon reliable fault detection and isolation mechanisms. In the dynamic operational contexts of hexacopter UAVs, potential faults such as motor failures, sensor malfunctions, or communication disruptions can lead to catastrophic consequences including loss of control, collisions, or data loss. Detecting and isolating these faults accurately and swiftly is imperative to ensure safe and efficient UAV operations.

Our objective is to improve the reliability and accuracy of fault detection and isolation for a single motor failure in hexacopter UAVs. Commencing with a foundational exposition on hexacopter UAV dynamics, prevalent fault conditions, and classical machine learning classifiers, the research subsequently introduces LSTM networks and conducts a review of pertinent literature in fault detection and isolation, laying the groundwork for the proposed methodology.

The principal contribution of this thesis revolves around the formulation of a fault detection and isolation paradigm that combines LSTM networks for latent temporal feature extraction with ensemble classifiers, notably Random Forests, aimed at enhancing fault detection efficacy. By harnessing the temporal intricacies inherent in UAV data using LSTM networks, the proposed model exhibits robust performance under measurement noise in fault detection and isolation tasks. The evaluation of the proposed approach encompasses comprehensive analyses conducted on synthetic and real-world datasets, encompassing examinations of noise resilience, fault detection and isolation timing, and the deployment on resource-constrained platforms such as the Raspberry Pi.

Comparative assessments are conducted with benchmark models from both classical and deep learning domains, wherein our proposed approach demonstrates superior performance with an accuracy of 96.8% for stimulated datasets and 83.2% for real-world datasets. Furthermore, noise analysis highlights the resilience of our proposed method across varying noise intensities, underscoring its adaptability and robustness. The analysis conducted on synthetic data serves as a crucial validation step, instilling confidence in the model's efficacy for real-world applications where data collection is inherently challenging.

Subsequent experiments investigate fault detection and inference times, yielding insights into the temporal efficiency of the proposed approach. On an onboard microcontroller like the Raspberry Pi, the average inference time for our model is measured at 6.512 milliseconds, with additional statistical

analyses providing further insights into performance metrics such as minimum and maximum algorithm run time and delay in prediction, alongside standard deviation.

In conclusion, the thesis summarizes key findings and contributions, emphasizing the efficacy of the proposed approach in enhancing fault detection and isolation in hexacopter UAVs. Furthermore, it outlines potential avenues for future research, thereby underscoring the practical applicability of advanced machine learning techniques within real-world UAV systems. In essence, this thesis presents a novel fault detection and isolation framework that integrates LSTM networks with ensemble classifiers, thereby advancing fault detection and isolation capabilities in hexacopter UAVs operating in real-world scenarios.

Contents

Chapter Page

1 Introduction . . . 1
1.1 Thesis Outline . . . 2
1.2 Thesis Contributions . . . 3
1.3 Preliminaries . . . 4
1.3.1 Dynamics of Hexacopter UAV . . . 4
1.3.2 Fault Conditions in Hexacopter . . . 6
1.3.3 Overview of Classical Machine Learning Classifiers . . . 10
1.3.3.1 Decision Trees . . . 10
1.3.3.2 Random Forest . . . 13
1.3.4 Introduction to Long Short-Term Memory (LSTM) . . . 13
1.4 Related Work . . . 16
1.4.1 Fault Detection and Isolation . . . 16
1.4.2 Classical Approaches . . . 16
1.4.3 Deep Learning based FDI . . . 16
1.4.4 LSTM-based Approaches . . . 17

2 Fault Detection and Isolation in Hexacopter UAVs using Ensemble Classifier-Enhanced LSTM 19
2.1 Problem Formulation . . . 19
2.2 Methodology . . . 20
2.2.1 State-Forecasting LSTM . . . 21
2.2.2 Random-Forest Ensemble Classifier . . . 21
2.2.3 Addressing Class-imbalance . . . 23
2.3 Implementation and Experimental Setup . . . 24
2.3.1 Metrics Used . . . 24
2.3.1.1 Correctness . . . 24
2.3.1.2 Latency . . . 25
2.3.2 Baseline Models . . . 26
2.3.3 Implementation Details . . . 27
2.3.4 Dataset . . . 28

3 Results and Analysis . . . 33
3.1 Results on Simulated Dataset . . . 33
3.1.1 Overall Performance . . . 34
3.1.2 Motor-Wise Performance . . . 35
3.1.3 Noise Tolerance Analysis . . . 36

3.1.4 Detection Time 37
3.2 Results on Real-World Dataset 37
3.2.1 Outdoor Dataset Collection 37
3.2.2 Experiment Setup 39
3.2.3 Model Performance Comparison 39
3.2.4 Motor-wise Results 41
3.2.5 Detection Time 43
3.3 Performance on Raspberry Pi 44
3.3.1 Experiment Setup 44
3.3.2 Overall Performance on Raspberry Pi 46
3.3.3 Statistical Analysis on Inference Time 47

4 Conclusions 50

Bibliography 52

List of Figures

Figure Page

1.1 (a) Healthy motor, (b) Propeller failure (c) Eccentric failure (d) Bearing failure 1

1.2 **Fault Scenario Overview** 2

1.3 **FBD of a Hexcopter UAV**: The hexacopter is depicted with six rotors, the arms of the hexacopter with length l. The body-fixed coordinate frame is represented by the axes X, Y, and Z. The rotational directions of the rotors are indicated, with the green and orange color. 5

1.4 **Effect of Motor 1 Failure on Torque on UAV Body**. When Motor 1 is turned off, as described in Table 1.2, the Roll and Yaw responses become positive due to the resultant positive torque, while the Pitch remains mostly unchanged due to zero torque. Here, Fig. (a), (b), and (c) describe the UAV's angular rate in roll, yaw, and pitch directions along the body axis. The orange line represents the actual values, while the blue line represents the desired values. The red dotted line indicates the timestep of the inception of the fault in the given motor. 8

1.5 **Effect of Motor 3 Failure on Torque on UAV Body**. As described in Table 1.2, the Roll and Pitch responses become negative due to the resultant negative torque, while the Yaw increases due to the positive torque. Here, Fig. (a), (b), and (c) describe the UAV's angular rate in roll, yaw, and pitch directions along the body axis. The orange line represents the actual values, while the blue line represents the desired values. The red dotted line indicates the timestep of the inception of the fault in the given motor. . 9

1.6 **Decision tree Overview:** This diagram illustrates the structure of a decision tree with examples of both binary and multi-way branching. The root node at the top represents the starting point of the decision-making process. From the root node, the tree can branch out into decision nodes, which are the points where the data is split. 10

1.7 **Unrolled RNN:** This diagram depicts an unrolled recurrent neural network (RNN), where each node A represents a neural network layer at a different time step. The input at each time step x_t is processed by the node, resulting in an output h_t that is passed on to the same network at the next time step, illustrating the network's ability to maintain state over time. 14

1.8 **Architecture of a LSTM Cell:** An LSTM cell at time step t showing the flow of information through various gates. The forget gate f_t determines which parts of the cell state C_{t-1} are to be discarded. The input gate i_t and the candidate cell state $\tilde{C}_t$ decide which values are to be added to the cell state. The cell state C_t is updated by combining the past cell state and new candidate values. The output gate o_t controls which parts of the cell state make it to the output h_t. 15

2.1 **Seq2Seq State-forecasting Model Φ:** The figures illustrates the sequence-2-sequence (seq2seq) prediction of our state-forecasting model. The input to our model is a the state vector $\mathcal{X}_t = [X_T\ X_{T-1}\ \ldots\ X_{T-P}]$. The LSTM initial state (c_i, h_i) is updated at each prediction, and at the last step gets updated to (c_{P+1}, h_{P+1}) after the LSTM outputs $\hat{X}_{T+1}$. 21

2.2 **Ensemble-Classifier LSTM Pipeline**: Hexacopter system onboard consists of on-board microcontroller and a flight controller chip. The sensors onboard collect the time-series flight data $\mathbf{X}$. We propose an Ensemble classifier which is trained over the temporal features ϕ derived from the LSTM state and current state. During inference, our proposed Ensemble Classifier-enhanced LSTM takes the data $\mathbf{X}$, and predicts the states for the next timestep $\hat{X}_{T+1}$ in a sequence-2-sequence (seq2seq [1]) fashion. The hidden state updated till the end h_{P+1} is passed onto the Random Forest [2] to obtain the fault detection and localization $\hat{Y}_T$. If a fault is detected, based on the motor localized, the onboard microcontroller can give corrective signals to the flight controler chip and reconfigure to resume stable flight. 27

2.3 **Training Simulated Data:** For Motor 1 (Fig. (a)) and Motor 2 (Fig. (b)) during simulated flight 1, each subfigure describes the roll (R), yaw (Y), and pitch (P) angular rates with respect to the body frame. 29

2.4 **Training Simulated Data:** For Motor 3 (Fig. (a)) and Motor 4 (Fig. (b)) during simulated flight 1, each subfigure describes the roll (R), yaw (Y), and pitch (P) angular rates with respect to the body frame. 30

2.5 **Training Simulated Data:** For Motor 5 (Fig. (a)) and Motor 6 (Fig. (b)) during simulated flight 1, each subfigure describes the roll (R), yaw (Y), and pitch (P) angular rates with respect to the body frame.. 31

2.6 **Test Data with Noise:** For Motor 1, (a), (b), and (c) represent the varying levels of noise at 10%, 50%, 90%, respectively. Within each fig., i, ii, and iii are the roll(R), yaw(Y), and pitch(P) angular rates. 32

3.1 **Accuracy with Noise Level:** The x-axis represents the range of noise levels, spanning from 0% to 180%, in 20% increments. The y-axis shows the accuracy of each approach. The proposed LSTM-based fault localization approach outperforms traditional methods. 36

3.2 **Trajectory Features and Fault Detection Motorwise:** The above graphs showcase trajectory features with time steps (sequence numbers) along with fault introduction and detection times. The x-axis indicates the sequence number (n), with each unit corresponding to a time-step at a rate of 36 frames per second, thus showcasing a truncated sequence of the full operation. The y-axis displays the angular velocity values in radians per second (rad/s). Red vertical lines represent the sequence step at which a fault is introduced, and green vertical lines denote the step at which the fault is detected by our model. Our detection system identifies the occurrence of a fault within 2-5 frames of its inception. 38

3.3 Hexacopter UAV ZD850 used for data collection. 39

3.4 The figure presents real-world data illustrating UAV Hexcopter motion, detailing the actual and desired values for roll (R), yaw (Y), and pitch (P). As evident, real data often exhibits significantly more noise and uncertainty in behavior compared to simulated datasets.Figures (a), (b), and (c) denote the data for Motors 1, 2, and 3, respectively, with subfigures (i), (ii), and (iii) corresponding to Roll, Pitch, and Yaw, respectively. . . 40

3.5 **Raspberry Pi**: This figure depicts a Raspberry Pi Model B, a low-cost, single-board computer employed in this research as a micro controller. The image highlights several key components: a microSD card slot for expandable storage, HDMI and USB ports for connecting peripherals, GPIO pins for interfacing with electronic components, and an LED power indicator. 45
3.6 This heatmap visualizes the prediction time of our model. 48

List of Tables

Table Page

1.1 Rotation Matrices along x, y and z axis. 4
1.2 Changes in the torque on the body due to motor failure. 7

2.1 Hexacopter Parameter Description and Value 28

3.1 **Benchmark Comparison**: The table denotes results aggregated over all types of faults on motors and no fault in the dataset mentioned above. Our method outperforms the baselines in all metrics. 34
3.2 **Motor-Wise Accuracy**: The table presents a comprehensive comparison of models' performance for fault detection and isolation across various motors, including scenarios where there are 'No Faults.' The models evaluated include both classical statistical methods (Logistic Regression, SVM, Random Forest, Rotation Forest) and deep learning approaches (ANN, LSTM), with our proposed approach highlighted for comparison. 35
3.3 Hexacopter parameters 41
3.4 Comparison of Accelerometer and Gyroscope Specifications present in flight controller **px4 cauv5 nano**[3] which has 3 sensors ICM-20602,ICM-20689, and BMI055. 42
3.5 **Outdoor Benchmark Comparison**: This table presents aggregated results across various fault types in outdoor motor conditions, as well as scenarios without faults, using the dataset specified. 42
3.6 **Outdoor Motor-Wise Accuracy**: This table compares fault detection and isolation accuracy across motors for various models, including classical methods (Logistic Regression, SVM, Random Forest, Rotation Forest) and DL approaches (ANN, LSTM), highlighting our proposed approach. 43
3.7 Median Detection Steps and Bounds for Each Motor. The timesteps are measured at a sampling rate of 50Hz. 44
3.8 Comparison of Raspberry Pi 3 Model B and 16" MacBook Pro with M1 Pro 46
3.9 **Inference Time (in seconds)**: Comparative Analysis of Model Performance on Differing Hardware Configurations summed for 1000 Data Points. The table presents the inference time for various machine learning models executed on Raspberry Pi 3 and Apple M1 Pro hardware platforms. 47
3.10 Comparison of Model Performance in Terms of Execution Time over 1000 Data Points. 47

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) have witnessed an exponential growth in applications across various industries, including surveillance [4], agriculture [5], aerial photography [6], and delivery services [7]. These versatile machines are propelled by multiple motors, each of which plays a crucial role in ensuring the UAV's stability and maneuverability. However, just like any mechanical system, these motors(including its associated propeller) are susceptible to wear and tear, manufacturing defects, and operational anomalies, which can lead to potential failures[8]. Fig. 1.1 describe some of the scenarios of UAV failure. The potential detecting and diagnosing these faults in a timely and accurate manner is imperative to ensure the safety and reliability of UAV operations.

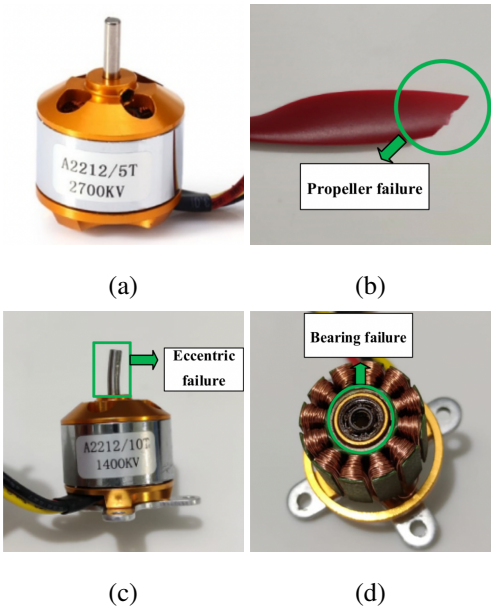


Figure 1.1: (a) Healthy motor, (b) Propeller failure (c) Eccentric failure (d) Bearing failure

Image Source : [8]

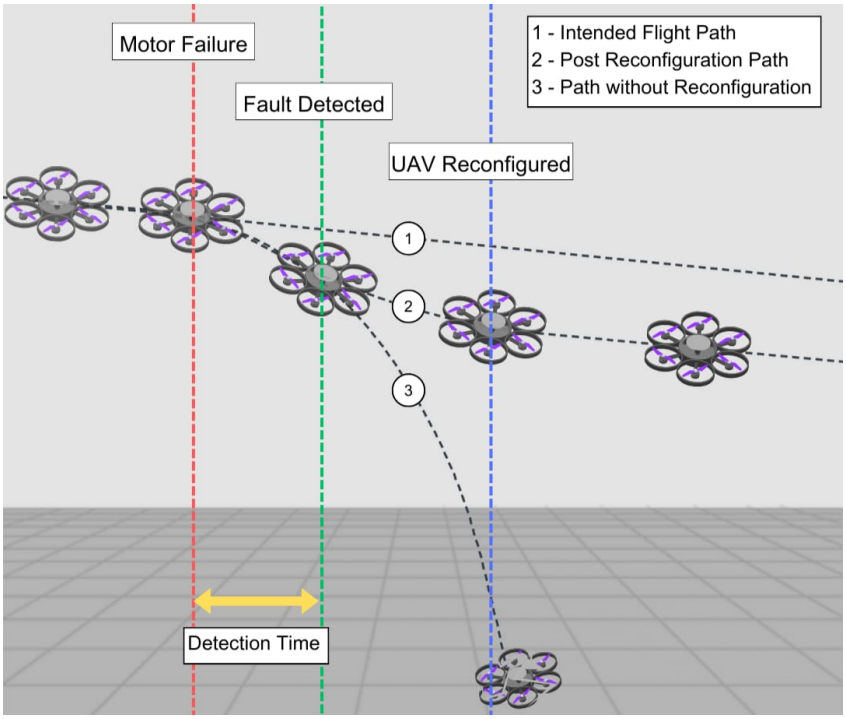


Figure 1.2: **Fault Scenario Overview**

The Figure 1.3 illustrates the sequence of events following a single motor failure in a UAV, upto its reconfiguration. When the motor fails inflight (red), the UAV begins to deviate from its intended path (1). The system onboard UAV detects and localizes the fault (green). The timespan between these two events is termed the "detection time". Once the problematic motor is identified, the UAV is reconfigured (blue) so it can operate with the remaining functional motors. The primary objective of our work is to accurately detect and pinpoint the faulty motor within a time to ensure the UAV can be safely maneuvered and recovered, and that the UAV can resumes stable flight and prevent crash-causing paths (2) that can happen due to delay or error in detection and localization of the fault. The UAV strays off course when a motor fails (red line) until the fault is detected and localized by the green line, representing the detection time. The UAV is then reconfigured to function with the remaining motors, stabilizing its flight by the blue line. Our paper's goal is to accurately locate faulty motor, enabling safe UAV recovery.

1.1 Thesis Outline

This thesis presents a comprehensive study on Fault Detection and Isolation (FDI) in Hexacopter Unmanned Aerial Vehicles (UAVs) using a novel approach that combines ensemble classifiers with Long Short-Term Memory (LSTM) networks. The focus is on enhancing the reliability and accuracy of

fault detection and isolation in the complex dynamics of hexacopter UAVs under various fault conditions and operational environments.

Chapter 1 provides foundational knowledge required to understand the complexities involved in the dynamics of hexacopter UAVs, outlines common fault conditions, and reviews classical machine learning classifiers including Decision Trees and Random Forests. This chapter also introduces LSTM networks, setting the stage for their application in fault detection and isolation. Additionally, it surveys related work in the field, categorizing existing approaches into classical approaches, deep learning-based fault detection, and specifically, LSTM-based approaches.

Fault Detection and Isolation in Hexacopter UAVs using Ensemble Classifier-Enhanced LSTM (Chapter 2) delves into the core contribution of this thesis. It begins with the problem formulation, followed by a detailed description of the proposed methodology which integrates a state-forecasting LSTM with a Random-Forest ensemble classifier to enhance FDI performance. This chapter also addresses the challenge of class imbalance in training data and describes the implementation and experimental setup, including metrics used for evaluation, baseline models for comparison, implementation details, and dataset characteristics. Chapter 3 presents the findings of this study. It is divided into sections that discuss the performance of the proposed method on both synthetic and real-world datasets, providing insights into overall and motor-wise performance, noise tolerance, and detection times. Additionally, the feasibility of deploying this solution on low-power devices like the Raspberry Pi is evaluated, highlighting the practical implications of this research.

Finally we summarizes the key findings of the research, emphasizing the effectiveness of the ensemble classifier-enhanced LSTM approach in improving fault detection and isolation in hexacopter UAVs. It also outlines potential areas for future research, suggesting ways to extend and refine this work to further advance the field of UAV fault detection and isolation. Throughout the thesis, the emphasis is on the practical application of sophisticated machine learning techniques to real-world problems, demonstrating the potential of combining classical classifiers with advanced neural networks to significantly improve the reliability and efficiency of FDI systems in UAVs.

1.2 Thesis Contributions

In this work, we make several significant contributions to the field of fault detection and localization in hexacopters. Our contributions in this thesis are as follows:

- We introduce an innovative approach that combines pre-trained Long Short-Term Memory (LSTM) networks with ensemble learning, achieving an impressive accuracy of 96.78%. This hybrid method leverages the strengths of deep learning to effectively identify and localize faults.
- To address the challenge of imbalanced data, which is common in real-world scenarios, we incorporate Weighted Class Loss into our model. This enhances the model's ability to generalize from data that closely mimic real-world conditions.

- We have developed a synthetic dataset that simulates a broad spectrum of motor behaviors, including both normal operations and faulty conditions. This dataset not only serves as a robust platform for training and evaluating our model but also stands as a valuable resource for future research in UAV FDI.

- Our experiments extend to analyzing the model's noise tolerance, showcasing its resilience in handling data with a high signal-to-noise ratio.

- We demonstrate the practical applicability of our approach by conducting experiments with real-world data and evaluating the execution time of our model on a Raspberry Pi, highlighting its potential for real-time applications in UAV systems.

Further in this chapter, we present an overview of fundamentals of hexacopters, alongside an explanation of the basic functioning of LSTM networks and Random Forest algorithms, which are integral to our proposed FDI model. The latter part of the chapter is dedicated for the existing literature in the domain of fault detection and isolation for hexacopters, emphasizing both classical and machine learning methodologies, and also Long Short-Term Memory (LSTM) networks for handling time series data.

1.3 Preliminaries

1.3.1 Dynamics of Hexacopter UAV

This section will deal with the methods used to build the mathematical model for the hexacopter. The angular orientation of the aircraft is described by the three angles (Roll, Pitch, and Yaw) that govern the flight of the hexacopter, these are called the Euler Angles. The Euler angles Yaw (ψ), Pitch (θ), and Roll (ϕ) define the angular position of the Body frame with respect to the Inertial frame. The schematic of the hexacopter is presented in Fig. 1.3.

Let us take linear position vector and rotational position (Euler angle) vector in the inertial frame by means of $\mathbf{X} = [x\, y\, z]^T$ and $\boldsymbol{\eta} = [\phi\, \theta\, \psi]^T$, respectively. Therefore, the linear velocities and angular velocities in the inertial frame will be $\dot{\mathbf{X}} = [\dot{x}\, \dot{y}\, \dot{z}]^T$ and $\dot{\boldsymbol{\eta}} = [\dot{\phi}\, \dot{\theta}\, \dot{\psi}]^T$ [9]. Now, we define the transformation from the body frame to the inertial frame using the rotation matrix $\mathbf{R}$ which is orthogonal. $\mathbf{R}$ can be defined by the combined rotation along x, y, and z-axis as :

$$\mathbf{R_x} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \quad \mathbf{R_y} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad \mathbf{R_z} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Table 1.1: Rotation Matrices along x, y and z axis.

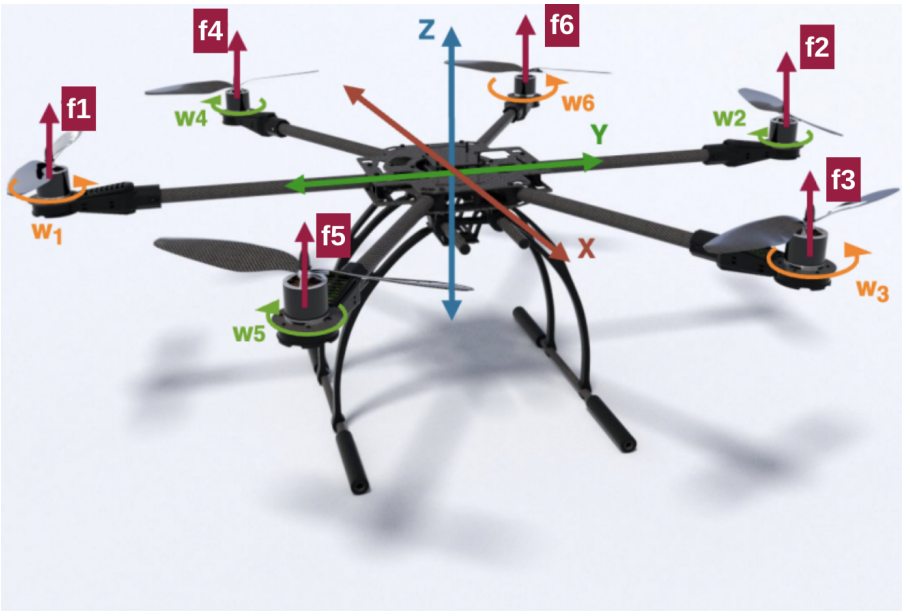


Figure 1.3: **FBD of a Hexcopter UAV**: The hexacopter is depicted with six rotors, the arms of the hexacopter with length l. The body-fixed coordinate frame is represented by the axes X, Y, and Z. The rotational directions of the rotors are indicated, with the green and orange color.

The motion of a rigid body can be decomposed into the translational and rotational components. Therefore, in order to describe the dynamics of the hexacopter, assumed to be a rigid body, the Newton-Euler equations, that govern linear and angular motion, are taken into account. First of all, the force acting on the hexacopter is provided by Eq. 1.1 where $\boldsymbol{\nu}$ is the angular velocity of UAV and $\mathbf{v}_B$ is the translation velocity. Rotation metric from world to bodyframe is denoted by $\mathbf{R}^T$.

$$\mathbf{F} = \frac{d(m\mathbf{v}_B)}{dt} + \boldsymbol{\nu} \times (m\mathbf{v}_B) \tag{1.1}$$

Every rotor i has an angular velocity ω_i, which generates a force $f_i = [0\,0\,k\omega_i^2]$ being k the lift constant, thus the total thrust T_B is given by $T_B = [0\,0\,T]^T$ with

$$T = \sum_{i=1}^{6} f_i = k \sum_{i=1}^{6} \omega_i \tag{1.2}$$

Total thrust together with gravitational force represents the total force acting on the hexacopter,

$$\mathbf{F} = \mathbf{R}^T \mathbf{F}_g + T_B. \tag{1.3}$$

As a consequence, the translation component of the motion referred to the body frame is

$$m\dot{\mathbf{v}}_B + \boldsymbol{\nu} \times (m\mathbf{v}_B) = \mathbf{R}^T \mathbf{F}_g + T_B. \tag{1.4}$$

Let I be the inertia matrix. The hexacopter has a symmetric structure with respect to the X_B-axis, Y_B-axis, and Z_B-axis, thus the inertia matrix is the diagonal one $I = \text{diag}(I_{xx}, I_{yy}, I_{zz})$. As the total external moment M concerns, the rate of change of the angular momentum $H = I\boldsymbol{\nu}$ is considered and the moment acting on the hexacopter is provided by

$$\mathbf{M} = \frac{d(I\boldsymbol{\nu})}{dt} + \boldsymbol{\nu} \times (I\boldsymbol{\nu}). \tag{1.5}$$

Moreover, angular velocity and acceleration of the rotor create a torque

$$\tau_{Mi} = b\omega_i^2 \tag{1.6}$$

around the rotor axis, where b is the drag constant and I_M is the inertia moment of the rotor i. From the geometrical structure of the hexacopter and from the components of f_i and τ_{Mi} over the body frame, it is possible to get the information on roll, pitch, and yaw moment, namely

$$\begin{bmatrix} \tau_r \\ \tau_p \\ \tau_y \end{bmatrix} = \begin{bmatrix} kl\left(-\omega_1^2 - \frac{\omega_5^2}{4} + \frac{\omega_3^2}{4} + \omega_2^2 + \frac{\omega_6^2}{4} - \frac{\omega_4^2}{4}\right) \\ -\frac{3}{4}kl\left(\omega_5^2 + \omega_3^2 - \omega_6^2 - \omega_4^2\right) \\ b\left(-\omega_1^2 + \omega_5^2 - \omega_3^2 + \omega_2^2 - \omega_6^2 + \omega_4^2\right) \end{bmatrix} \tag{1.7}$$

Here l is the distance between the rotor and the center of gravity of the hexacopter and $\dot{\omega}_i$ denotes the derivative of $\omega_i(t)$ with respect to time, i.e., $\frac{d\omega_i(t)}{dt}$. Afterwards, the equation that governs the rotational dynamic can be summarized as

$$I\dot{\boldsymbol{\nu}} + \boldsymbol{\nu} \times (I\boldsymbol{\nu}) + \Gamma = \boldsymbol{\tau}_B, \tag{1.8}$$

in which Γ represents the gyroscopic forces and $\boldsymbol{\tau}_B$ the external torque.

1.3.2 Fault Conditions in Hexacopter

In the event of motor failure within a hexacopter UAV system, where the velocity of a particular motor diminishes to zero, a deviation occurs between the actual and desired torque values. Our research focus on such cases where one of the motor stops functioning i.e. $\omega_i = 0$ for i in [1..6]. In this section we analyse effect of such motor failure on the dynamics of Hexcopter UAV.

Consider a scenario where motor 1 experiences a fault, resulting in its angular velocity (ω_1) being reduced to zero. Consequently, the torque experienced by the UAV undergoes a transformation, as depicted in Equation **??**, differing from the baseline torque representation outlined in Equation **??**.

$$\begin{bmatrix} \tau_r \\ \tau_p \\ \tau_y \end{bmatrix} = \begin{bmatrix} kl\left(0 - \frac{\omega_5^2}{4} + \frac{\omega_3^2}{4} + \omega_2^2 + \frac{\omega_6^2}{4} - \frac{\omega_4^2}{4}\right) \\ -\frac{3}{4}kl\left(\omega_5^2 + \omega_3^2 - \omega_6^2 - \omega_4^2\right) \\ b\left(0 + \omega_5^2 - \omega_3^2 + \omega_2^2 - \omega_6^2 + \omega_4^2\right) \end{bmatrix} \tag{1.9}$$

Here, τ_r, τ_p, and τ_y represent the roll, pitch, and yaw torques, respectively. The term klf_1 symbolizes the loss of force contribution from motor 1 in the roll direction, while τ_1 denotes the corresponding

torque loss in the yaw direction. Upon motor 1 failure, it becomes evident that there is an increase in both roll and yaw torques, while the pitch remains unaffected. This observation finds validation through real flight experiments as depicted in Figure 1.4.

$$\tau_r = \tau_{rd} + lf_1, \tag{1.10}$$

$$\tau_p = \tau_{pd}, \tag{1.11}$$

$$\tau_y = \tau_{yd} + \tau_1, \tag{1.12}$$

When motor 3 experiences a failure, resulting in its cessation of operation, the alteration in body torque can be described by Equation 1.13. This equation presents the difference in body torque before and after the motor failure, outlined in terms of roll (τ_r), pitch (τ_p), and yaw (τ_y) torques is also illustrated in 1.5 for real flight data. Here, the negative signs in the torque components indicate a reduction in torque magnitude due to the failure of motor 3. Specifically, the roll torque (τ_r) experiences a decrease as motor 3 contributes negatively to this torque component.

$$\begin{bmatrix} \tau_r \\ \tau_p \\ \tau_y \end{bmatrix}_{new} - \begin{bmatrix} \tau_r \\ \tau_p \\ \tau_y \end{bmatrix}_{old} = \begin{bmatrix} -kl\left(\frac{\omega_3^2}{4}\right) \\ -\frac{3}{4}kl(\omega_3^2) \\ b(\omega_3^2) \end{bmatrix} \tag{1.13}$$

To comprehensively understand the impact of motor failure on the hexacopter's body torque, Table 1.2 is provided. This table summarizes the changes in body torque resulting from the failure of each motor. Each row corresponds to a specific type of torque (roll, pitch, or yaw), while each column represents a different motor within the hexacopter system. The entries in the table indicate whether the torque value increases (+), decreases (-), or remains unchanged (0) when a particular motor fails. The table only considers the relative changes in torque magnitude and does not provide absolute values.

	M_1	M_2	M_3	M_4	M_5	M_6
$\Delta\tau_r$	+	-	-	+	+	-
$\Delta\tau_p$	0	0	-	+	-	+
$\Delta\tau_y$	+	-	+	-	-	+

Table 1.2: Changes in the torque on the body due to motor failure.

Additionally, Figures 1.4 and 1.5 visually illustrate the relationship between motor failure and body torque for motors 1 and 3 in a real flight scenario. These visual representations further aid in verifying the theoretical conclusions regarding the dynamic behavior of the hexacopter system under such circumstances.

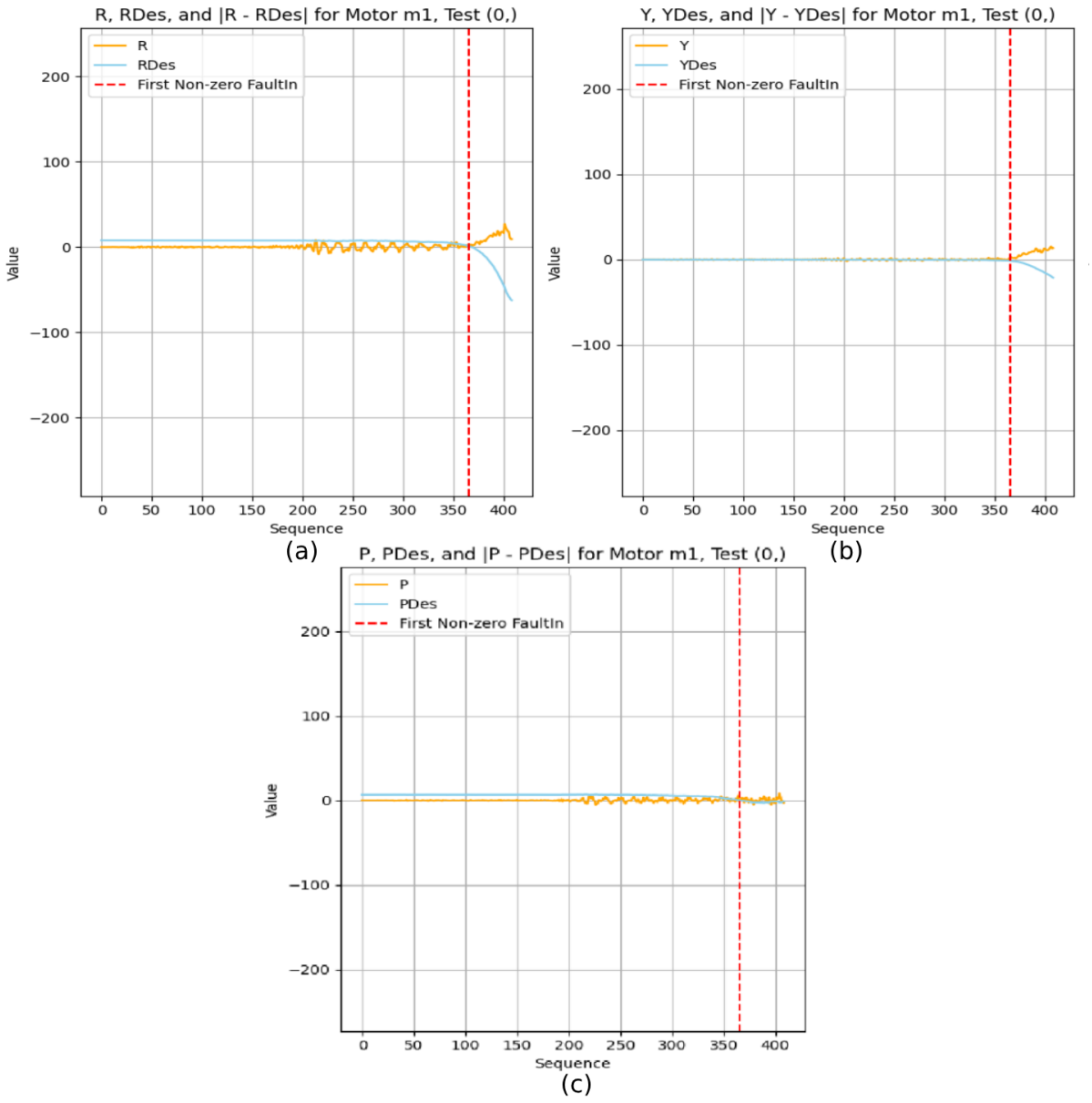


Figure 1.4: **Effect of Motor 1 Failure on Torque on UAV Body**. When Motor 1 is turned off, as described in Table 1.2, the Roll and Yaw responses become positive due to the resultant positive torque, while the Pitch remains mostly unchanged due to zero torque. Here, Fig. (a), (b), and (c) describe the UAV's angular rate in roll, yaw, and pitch directions along the body axis. The orange line represents the actual values, while the blue line represents the desired values. The red dotted line indicates the timestep of the inception of the fault in the given motor.

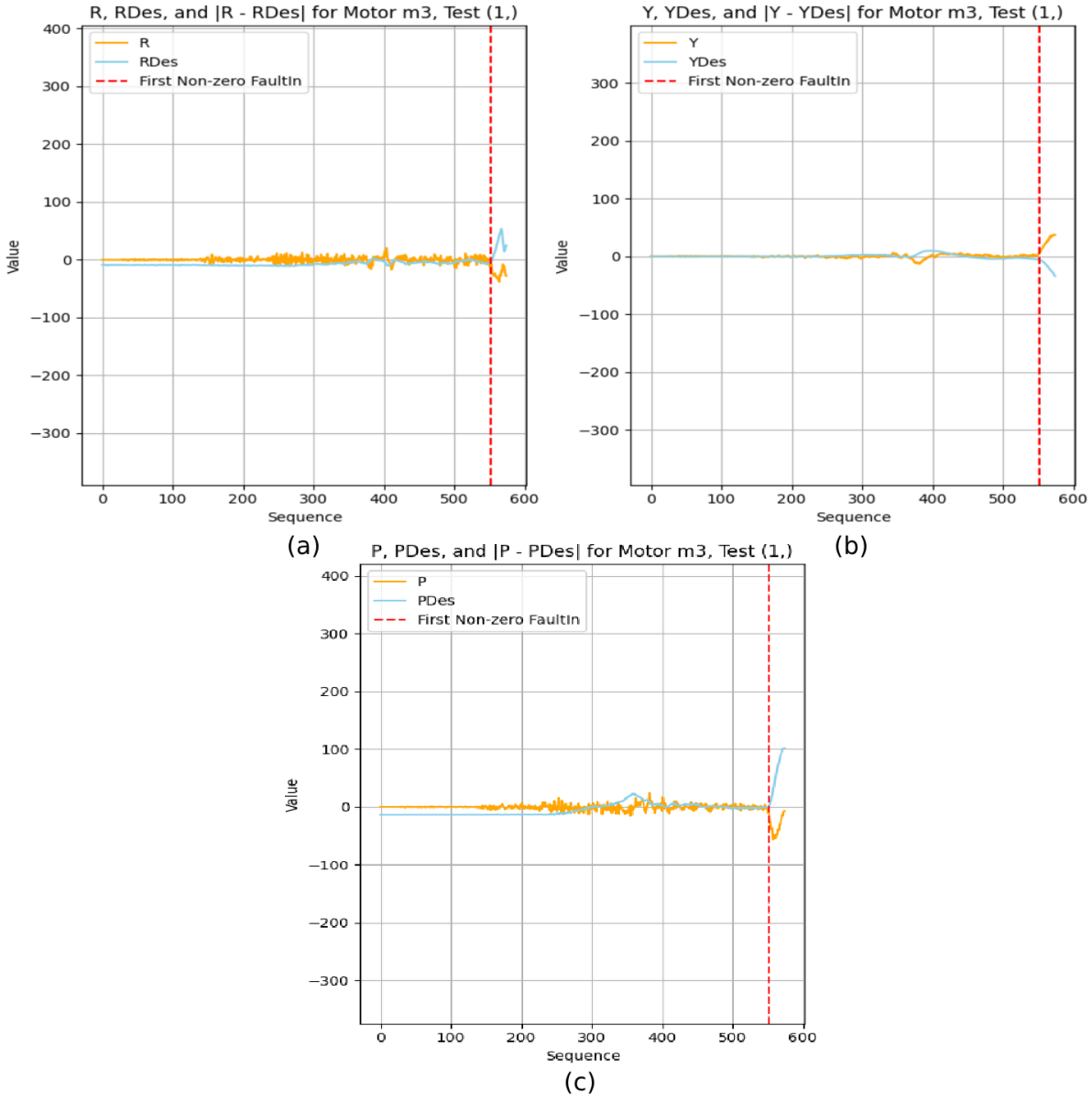


Figure 1.5: **Effect of Motor 3 Failure on Torque on UAV Body**. As described in Table 1.2, the Roll and Pitch responses become negative due to the resultant negative torque, while the Yaw increases due to the positive torque. Here, Fig. (a), (b), and (c) describe the UAV's angular rate in roll, yaw, and pitch directions along the body axis. The orange line represents the actual values, while the blue line represents the desired values. The red dotted line indicates the timestep of the inception of the fault in the given motor.

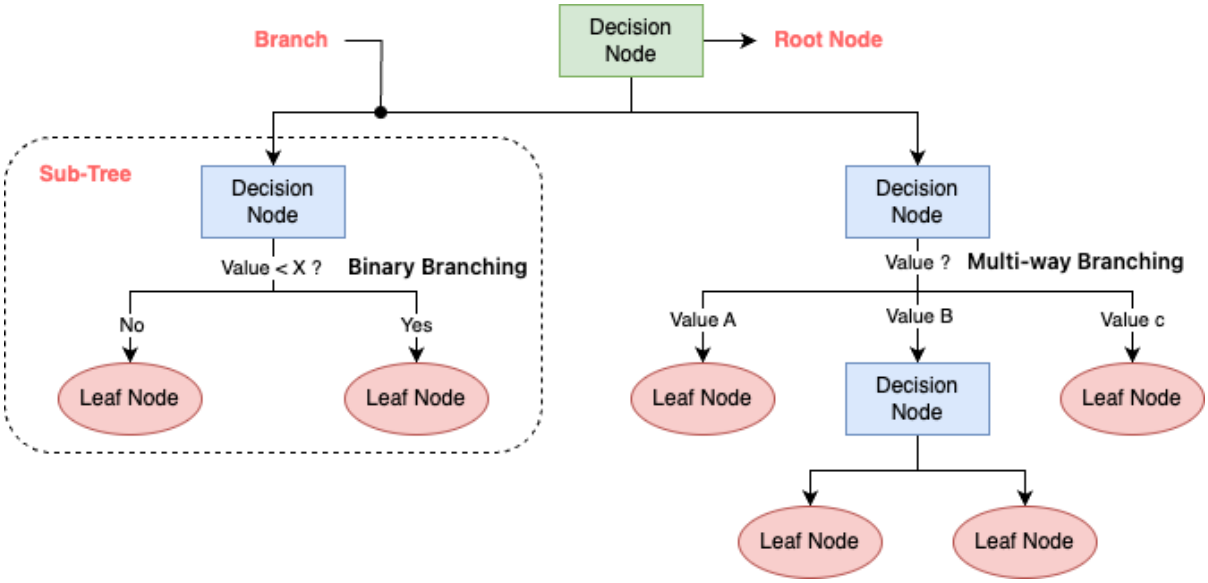


Figure 1.6: **Decision tree Overview:** This diagram illustrates the structure of a decision tree with examples of both binary and multi-way branching. The root node at the top represents the starting point of the decision-making process. From the root node, the tree can branch out into decision nodes, which are the points where the data is split.

1.3.3 Overview of Classical Machine Learning Classifiers

1.3.3.1 Decision Trees

A Decision Tree[10] is a simple yet effective model that divides the data into branches to form a tree structure. Each internal node of the tree represents a test on an attribute, and each leaf node represents a class label. The paths from root to leaf represent classification rules. It operates by breaking down a dataset into smaller and smaller subsets, while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches, each representing values for the attribute tested. Leaf nodes represent a classification or decision.

The core idea is to select the best attribute to split the dataset into smaller subsets based on certain conditions. The type of branching primarily depends on the nature of the attributes (features) used for making decisions. As show in 1.6 on the left, within the sub-tree enclosed by a dashed outline, we see an example of binary branching: a decision node splits the data into two paths based on a binary condition. Depending on whether the condition is true or false, the flow moves to the respective leaf node, where a classification or decision is made. When dealing with numerical values, the decision tree algorithm looks for a point to split the data into two groups. This is usually done by sorting the values and trying different split points, which are typically between two adjacent values in the sorted list. On

the right side, the tree demonstrates multi-way branching: a decision node splits the data into multiple paths based on categorical values.With categorical values, the branching is more straightforward as each unique value or category can form a branch. For instance, if the feature is "color" with categories like [Red, Blue, Green], the tree might split into three branches, one for each color.

Branching in a decision tree is performed through a process known as recursive partitioning. This process divides the data into subsets based on an attribute value. The goal is to partition the data in a way that increases the homogeneity regarding the target variable within those subsets.Homogeneity, in this context, means that the members of each group after the split should be as similar as possible in terms of the target variable. A perfectly homogeneous node is one where all instances belong to a single class. To achieve this, decision tree algorithms employ measures of purity to evaluate potential splits.

Gini Impurity is a metric used to gauge the purity of a node. The Gini impurity of a set is calculated as in Eq. 1.14. A Gini score of 0 indicates that the node is perfectly pure, with all instances belonging to a single class, while a higher Gini score indicates a higher level of impurity. **Entropy**, another measure, reflects the randomness or disorder within a node. The entropy of a dataset is defined as Eq. 1.15. The entropy is zero when all samples at a node are from a single class, signifying no disorder.

$$Gini(S) = 1 - \sum_{i=1}^{C} (p_i)^2 \tag{1.14}$$

$$Entropy(S) = -\sum_{i=1}^{C} p_i \log_2 p_i \tag{1.15}$$

where S is the dataset for a node, C is the number of classes, and p_i is the proportion of samples belonging to class i within the set S.

Information Gain is based on the concept of entropy and is used to determine which feature split will yield the most homogeneous branches. Information Gain is the change in entropy as a result of dividing a dataset according to a given attribute. It is calculated as the difference between the entropy of the parent node and the sum of the entropies of each child node, weighted by the proportion of instances at each child node:

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{1.16}$$

Here, S is the dataset of the parent node, A is the attribute by which the node is being split, $Values(A)$ are the unique values of attribute A, S_v is the subset of S for which attribute A has value v, and $|S_v|/|S|$ is the proportion of the number of instances in S_v to the number of instances in the parent set S.

The decision tree algorithm will evaluate each potential split based on these criteria, opting for the one that yields the greatest purity gain, that is, the most homogenous branches. This process is recursively continued for each branch until a stopping criterion is reached, ensuring that the tree does not overfit the data.

Algorithm 1 Decision Tree Algorithm

Require: Training dataset D, feature set F, target variable Y

function DECISIONTREE(D, F, Y)
    Create a node N
    if all samples in D belong to the same class C **then**
        Mark N as a leaf node with class C **return** N
    else if F is empty OR stopping criteria are met **then**
        Mark N as a leaf node with the most common class in D **return** N
    end if
    $f \leftarrow$ SELECTBESTFEATURE(D, F, Y)
    for each possible value v of feature f **do**
        Partition D into subsets D_v where feature f has value v
        if D_v is empty **then**
            Create a leaf node with the most common class in D
        else
            $N_v \leftarrow$ DECISIONTREE(D_v, $F \setminus \{f\}$, Y); Add branch to N with label v and subtree N_v
        end if
    end for
    return N
end function

Algorithm 2 Select Best Feature for Split

function SELECTBESTFEATURE(D, F, Y)
    $maxGain \leftarrow -\infty$; $f_{best} \leftarrow$ null
    for all feature $f \in F$ **do**
        $infoGain \leftarrow$ CALCULATEINFORMATIONGAIN(D, f, Y)
        if $infoGain > maxGain$ **then**
            $maxGain \leftarrow infoGain$; $f_{best} \leftarrow f$
        end if
    end for
    return f_{best}
end function

1.3.3.2 Random Forest

Random Forest[11] is an ensemble method that builds multiple decision trees and merges them together to obtain a more accurate and stable prediction. It effectively addresses the overfitting problem often faced by decision trees[12]. While a single decision tree makes its decision based on the paths from root to leaf, a Random Forest combines the output of multiple decision trees to make a final decision, thereby enhancing the overall predictive quality and robustness of the model.

In the context of ensemble methods in machine learning, particularly when discussing Random Forests, a fundamental question arises: given a single dataset, how can we produce a collection of decision trees that are each unique? To create diversity among the decision trees in a Random Forest, several techniques are employed. Bagging, or Bootstrap Aggregating, involves drawing random subsets of the data with replacement to train each tree, ensuring that each one has a different subset and thus a different perspective on the data.

Bagging (Bootstrap Aggregating)[13]: For a given dataset $X = \{x_1, x_2, ..., x_n\}$ with corresponding responses $Y = \{y_1, y_2, ..., y_n\}$, we generate B distinct bootstrap samples. Each of these samples is constructed by randomly selecting N observations with replacement from the original dataset, which results in new sample sets X_b and Y_b for each b in $1, ..., B$. Through bootstrap sampling, each decision tree f_b is trained on a slightly different data set. This variability is intentional and crucial as it introduces diversity among the trees in the forest, which in turn enhances the ensemble's overall performance by reducing the variance of the aggregated predictions.

Prediction: The output of the Random Forest is the class selected by most trees (mode) in the case of classification, or the average prediction (mean) of the individual trees in the case of regression. For classification problems, the Random Forest output $(\hat{f}(x))$ is the mode of the classes predicted by individual trees as:

$$\hat{f}(x) = \text{mode}\{f_1(x), \ldots, f_B(x)\} \tag{1.17}$$

The principle underlying Random Forests is the collaboration of "weak learners" (individual decision trees) to form a robust "strong learner" capable of mitigating noise and overfitting. Notably, Random Forests possess key hyperparameters such as the number of trees (B), the features considered for splitting at each leaf node (m), and the maximum tree depth. Fine-tuning these parameters contributes to achieving an optimized model. It's important to highlight that Random Forests often perform admirably with minimal tuning requirements, handling both categorical and numerical features while inherently performing feature selection[12]. This versatility has established Random Forests as a widely adopted and effective algorithm across various machine learning tasks.

1.3.4 Introduction to Long Short-Term Memory (LSTM)

Recurrent Neural Network (RNN) maintain a form of memory, allowing them to incorporate the context of previous inputs in their processing, Unlike standard networks, which process inputs in isolation. In a conventional neural network, the ability to consider sequential information is notably absent.

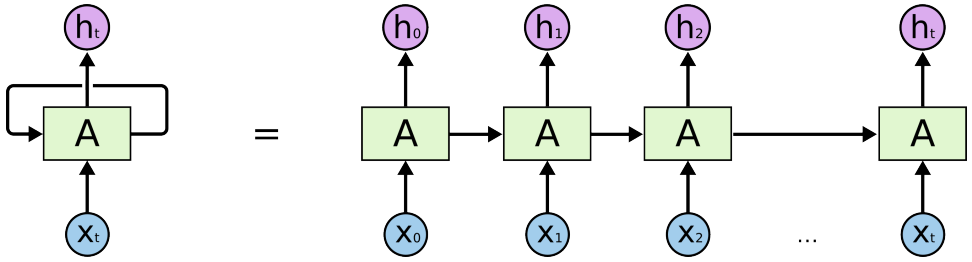


Figure 1.7: **Unrolled RNN:** This diagram depicts an unrolled recurrent neural network (RNN), where each node A represents a neural network layer at a different time step. The input at each time step x_t is processed by the node, resulting in an output h_t that is passed on to the same network at the next time step, illustrating the network's ability to maintain state over time.

RNNs, however, introduce loops within their architecture, enabling the persistence of information. This characteristic is illustrated in Fig. 1.7 when a segment of the network, denoted as A, processes an input x_t and produces an output h_t. The inclusion of a loop mechanism facilitates the transfer of information from one step to the next within the network.

RNNs are capable, in theory, of processing such long-term dependencies, and indeed, with carefully chosen parameters, they can solve simple problems that exhibit this characteristic. However, in practice, standard RNNs often fail to learn in the presence of long temporal gaps between relevant information and its point of use. This difficulty was highlighted in seminal works by Hochreiter[14] in 1991 and Bengio et al.[15] in 1994, which pointed to inherent issues in the network's architecture that make learning these dependencies challenging. The problem lies in the network's inability to maintain the influence of input information over long sequences, an issue commonly referred to as the vanishing gradient problem.

Long Short Term Memory(LSTM) networks are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997) [16]. An LSTM unit has the following components:

- **Cell State (C_t):** The essential innovation of LSTMs is the cell state, which acts like a conveyor belt, carrying relevant information throughout the process of sequential data handling with minimal alteration. Information is added or removed from the cell state via the gates, which are controlled by sigmoid layers that determine how much information should pass through.
- **Hidden State (h_t):** This is the output state of the LSTM, used for predictions and passed to the next time step.
- **Gates:** Gates are a way to optionally let information through. They are composed of a sigmoid neural net layer and a point-wise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. An LSTM has three of these gates:

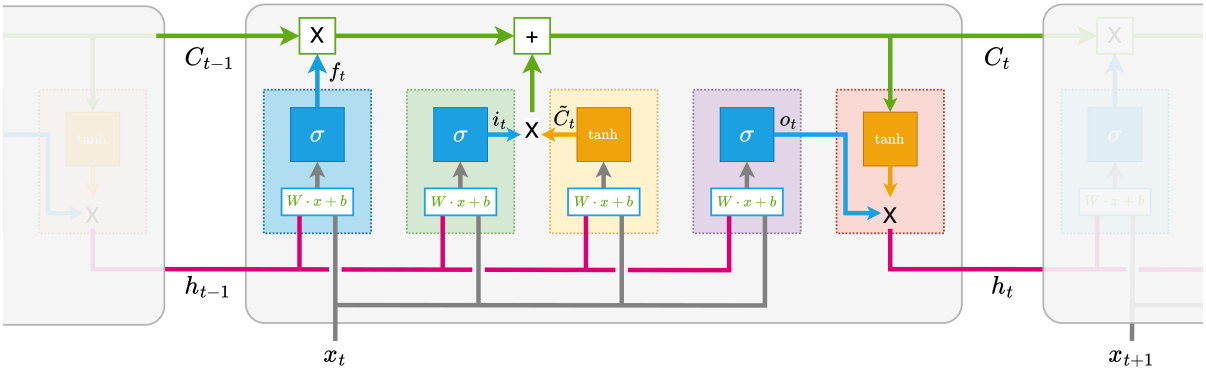


Figure 1.8: **Architecture of a LSTM Cell:** An LSTM cell at time step t showing the flow of information through various gates. The forget gate f_t determines which parts of the cell state C_{t-1} are to be discarded. The input gate i_t and the candidate cell state $\tilde{C}_t$ decide which values are to be added to the cell state. The cell state C_t is updated by combining the past cell state and new candidate values. The output gate o_t controls which parts of the cell state make it to the output h_t.

- Forget Gate (f_t): Decides what information to discard from the cell state.
- Input Gate (i_t): Updates the cell state with new information.
- Output Gate (o_t): Determines what the next hidden state should be.

The operations of an LSTM can be described by the equations below. Equation 1.18 describes the forget gate, equations 1.19 and 1.20 describe the input gate, equation 1.21 is for the cell state update, and equations 1.22 and 1.23 are for the output gate.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{1.18}$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{1.19}$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{1.20}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{1.21}$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{1.22}$$

$$h_t = o_t * \tanh(C_t) \tag{1.23}$$

In these equations σ represents the sigmoid function, $\tanh$ is the hyperbolic tangent function, W and b are the weights and biases for different gates, and $[h_{t-1}, x_t]$ denotes the concatenation of the previous hidden state and the current input. These equations collectively describe how an LSTM unit processes data at each time step, updating its cell state and hidden state based on the current input, the previous hidden state, and the learned parameters (weights and biases).

1.4 Related Work

1.4.1 Fault Detection and Isolation

Fault Detection and Isolation in control systems is an active area of research [17]. In recent years, several studies have addressed the critical task of identifying and mitigating faults in UAV systems [18]. FDI techniques in these systems are primarily divided into two categories: model-based and data-driven methods. Historically, model-based methods have been predominant, but data-driven approaches have been gaining significant traction and momentum in the field recently. This study reported an average accuracy of 97.8% in speed estimation and a fault detection and isolation accuracy ranging from 33.1% to 100%, averaging 82.75%. These tests demonstrated the method's sensitivity to even weak disturbances, with high accuracy despite low computational requirements.

1.4.2 Classical Approaches

In the classical approach to UAV fault detection and isolation, several papers have contributed insights into the mathematical modeling and actuator fault detection for UAV systems. [19], [20] claims to be effective for internal motor failures (no chnage to structure of UAV), however their performance quickly deteriorates if the UAV or the propeller incurs damage. The reason can be attributed to their assumption of rigid body dynamics.

In [21] they developed a random forest classifier trained on datasets generated on flight simulator, which utilizes the hexacopter's attitude and motor signals as inputs to detect and identify motor failures. The classifier demonstrated high accuracy, detecting and isolating faults in less than 100ms post-occurrence with fewer than one false positive per hour of flight. The authors of [22] design a 2-stage approach - fault detection followed by isolation, employing a Rotation Forest at each stage. Rotation Forest is an implementation of Random Forest tailored for sequential data.

In contrast to [22], our approach requires far lesser number of trees, and is able to output a accurate prediction faster. Classical methods discussed above often rely on handcrafted features and fixed thresholds making them not able to effectively capture intricate patterns in large datasets, limiting their adaptability and generalization across diverse scenarios. Our model uses features generated using LSTM enabling it to effectively capture intricate patterns in dataset.

1.4.3 Deep Learning based FDI

Deep-learning based approaches like [23, 24, 25] addresses the challenges of classical methods. J.J. Tong et. al. [23] presents a model for quadcopter propeller fault detection, the authors developed a hybrid data-generative model combining data-driven models and dynamic UAV models to simulate various fault scenarios and normal conditions. The model uses Long Short-Time Memory (LSTM) networks to estimate the performance drop in faulty propellers and a Convolutional Neural Network (CNN) for fault classification. The study focuses on identifying faulty propellers and their fault levels

using the propellers' RPM and flight data. The tests were conducted to validate the model, with an observed diagnosis accuracy over 80%.

Whereas [24] proposes the CNN model using the transfer learning. They proposes a method for diagnosing physical damage to quadrotor UAV propellers using only the audio data generated during flight. The method employs a Convolutional Neural Network (CNN) trained on time-frequency spectrograms of flight audio data. The model is capable of detecting and isolating propeller damage by analyzing the audio signal's amplitude and frequency characteristics. Additionally, the model utilizes transfer learning to adapt to UAVs with different characteristics, requiring only minimal new data for retraining. The approach was validated with an accuracy higher than 90%, demonstrating its effectiveness for propeller fault diagnosis in quadrotors. The approach in the paper primarily relies on audio data, which may be influenced by external noise and environmental factors, potentially affecting diagnostic accuracy. Additionally, it necessitates the installation of additional audio sensors on the UAV, adding to the complexity and potential weight of the onboard system. In [25], the dataset utilized for UAV fault detection and analysis is sourced exclusively from real-world UAV flight scenarios. This approach, while providing authentic data, inherently limits the diversity and range of fault scenarios that can be captured. Particularly, it's challenging to introduce and record data on fatal or severe faults in real-flight conditions, as this would pose significant risks to both the UAV and its surrounding environment.

1.4.4 LSTM-based Approaches

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network, have emerged as a powerful tool in FDI, particularly for their ability to effectively capture and analyze time-series data, extending to areas such as industrial machinery[26, 27], financial markets[28, 29], and healthcare systems[30, 31]. Their proficiency in handling time-series data makes them exceptionally well-suited for tasks that require the analysis of sequential information over extended periods.

In Unmanned Aerial Vehicles (UAVs), FDI involves processing complex flight sequence data, which often needs to be handled on-board. The inherent capability of LSTMs to learn from and make predictions based on sequential data is particularly beneficial in this context. [32] presents an innovative Fault Detection and Isolation framework for highly redundant Multirotor UAVs, specifically a hexadecarotor UAV with sixteen rotors. Utilizing LSTM networks they introduces FDI framework comprises a region classifier model for detecting and isolating faults and fault locator models for precise actuator location determination. While achieving high accuracy, the framework's dependence on a large, multi-stage model can increase the time required to identify and localize faults. Meanwhile, [33] emphasizes the utility of on-board LSTM techniques specifically for pinpointing the cause of faults. The research demonstrates over 90% accuracy in detecting faults and up to 85% accuracy in classifying various types of drone misoperations using both simulation and experimental data.

Inspired by the previous approaches, we utilize the weights of an LSTM-based model, which has been pre-trained for state-forecasting objectives and then fed to the LSTM embeddings to train a Random Forest ensemble algorithm to act as a classifier for detecting and localizing faults in hexacopter

UAVs. This approach capitalizes on the strengths of both deep learning and classical methods to provide more accurate and robust FDI.

Chapter 2

Fault Detection and Isolation in Hexacopter UAVs using Ensemble Classifier-Enhanced LSTM

Further, in this chapter we addresses this challenge by proposing a novel approach combining ensemble classifiers with Long Short-Term Memory (LSTM) networks tailored for hexacopter UAVs. The chapter begins by formulating the problem, delineating the necessity for accurate fault detection and isolation. Subsequently, the methodology is expounded, comprising a state-forecasting LSTM for trajectory prediction and a random-forest ensemble classifier for fault identification. Notably, techniques to handle class-imbalance issues are discussed. Implementation and experimental setups, including metrics for evaluation and baseline models, are detailed, along with dataset descriptions.

2.1 Problem Formulation

For a Hexacopter UAV system $\mathbf{M}$ consisting of N motors, each motor is represented by M_i. The state of the motors at timestep t is represented by Y_t, where $Y_t \in \{0, 1, \ldots, N\}$ such that:

$$\mathbf{M} = \{M_i\}, i \in \{1, 2, \ldots N\} \tag{2.1}$$

$$Y_t = 0 \iff \text{No motor is faulty} \tag{2.2}$$

$$Y_t = i \iff \text{Motor } M_i \text{ is faulty} \tag{2.3}$$

We assume that in one simulation, at most one motor may be faulty. For this UAV system, at each timestep t, where $t \in \{T-P, T-P+1, \ldots, T\}$, we assume the availability of a feature vector $X_t \in \mathbb{R}^6$ (Eq. 2.4) by concatenating the angular velocity around the x, y, and z-axis respectively w_{x_t}, w_{y_t}, w_{z_t}, along with the desired angular velocity around the x, y, and z-axis respectively w_{xd_t}, w_{yd_t}, w_{zd_t}.

$$X_t = [w_{x_t} \quad w_{y_t} \quad w_{z_t} \quad w_{xd_t} \quad w_{yd_t} \quad w_{zd_t}]^\top, \tag{2.4}$$

$$\mathbf{X} = [X_T \quad X_{T-1} \quad \ldots \quad X_{T-P}]. \tag{2.5}$$

Problem Statement: Given state vectors for the current timestep T and the previous P timesteps $\mathbf{X} \in \mathbb{R}^{P \times 6}$, the task of fault detection and isolation involves the prediction of the UAV state at the

current timestep T, i.e., $\hat{Y}_T$. A successful fault detection involves correctly predicting when the state is non-zero 2.6. A successful fault isolation, on the other hand, involves correctly identifying the motor number causing the fault 2.7.

$$\hat{Y}_t \neq 0 \iff Y_t \neq 0. \tag{2.6}$$

$$\hat{Y}_T = Y_T. \tag{2.7}$$

Given $F(X_i) = 1$, determine the M_i such that:

$$F(X_i) = Y_i = 1 \Rightarrow M_i. \tag{2.8}$$

Given $F(X_i) = 0$ for all $X_i \in X$, this implies that no motor is faulty:

$$\{F(X_i) = Y_i = 0, \forall X_i \in X\} \Rightarrow \varnothing. \tag{2.9}$$

The problem is to develop a model that can accurately classify the state of each motor (Y_i) and, when a fault is detected, identify which specific motor (M_i) is at fault based on the extracted features X_i. Additionally, the model should be able to determine when no motor is faulty.

2.2 Methodology

Temporal features are characteristics of data that capture information about the timing and ordering of events within a sequence. In the context of sequence data, like time series or signal data, temporal features reflect the dynamics that unfold over time. These features are crucial for understanding patterns that are dependent on the progression of time and are often leveraged in tasks that require the analysis of data across time intervals.

We use LSTM networks, as they are particularly well-suited for tasks involving sequential and temporal data [34] in the context of UAV data. These networks can capture complex dependencies in motor behavior, making them ideal for applications such as fault localization in hexacopter UAVs. We use them for modeling and predicting motor parameters over time. We first train a LSTM-based state-forecasting model, Φ which is capable of accurately predicting the next state, $\hat{X}_{t+1}$ from the previous state X_t, in a seq2seq [1] fashion. The recurrent architecture is able to encode and learn the dynamics of the UAV system. At test time, we use the LSTM-based model as an temporal-aware encoder and pass the input state vector, $\mathbf{X}$ to obtain the prediction of the next state, X_{t+1} and the LSTM state (c_{P+1}, h_{P+1}). We then train the Random Forest model conditioned on the hidden state, h_{P+1} obtained from LSTM along with the current state vector, X_t. We first describe the state-forecasting model, Φ followed by a discussion on the Random Forest ensemble.

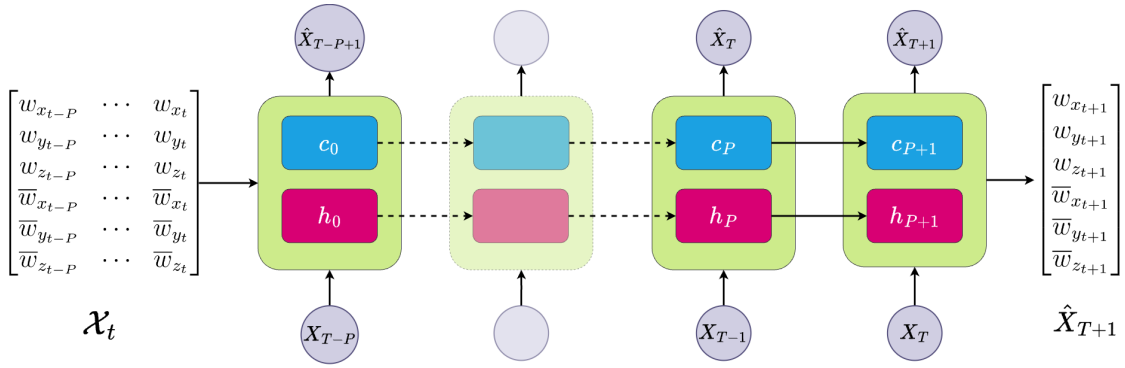


Figure 2.1: **Seq2Seq State-forecasting Model Φ:** The figures illustrates the sequence-2-sequence (seq2seq) prediction of our state-forecasting model. The input to our model is a the state vector $\mathcal{X}_t = [X_T\ X_{T-1}\ \dots\ X_{T-P}]$. The LSTM initial state (c_i, h_i) is updated at each prediction, and at the last step gets updated to (c_{P+1}, h_{P+1}) after the LSTM outputs $\hat{X}_{T+1}$.

2.2.1 State-Forecasting LSTM

To create temporal features, we train a state forecasting model **offline** for next state prediction problem in a sequence-2-sequence (seq2seq) fashion. The input horizon to our model is P steps, and the prediction horizon is $F = 1$. We use a single LSTM layer as illustrated in Fig 2.1. The hidden state size of the LSTM is chosen to be 256, and the projection size (the size of the final output) is set to be 6. The LSTM model here takes the current state vector X_t as input, and generates the output vector $\hat{X}_{t+1} \in \mathbb{R}^6$, updating its cell state and the hidden state (c_P, h_P) to the next state (c_{P+1}, h_{P+1}). Note that $c_k, h_k \in \mathbb{R}^{256}$. We train our forecasting model in a sequence-to-sequence (seq2seq [1]) fashion, with the Mean Squared Error (MSE) loss function as given in Fig 2.1. The algorithm to train is stated in Algorithm 3.

The hidden state of LSTM cell at timestep T, h_{P+1} is temporally aware of the past history. At test time, we have state vectors for the the previous P timesteps and the current T timestep, $\mathbf{X} = [X_T\ X_{T-1}\ \dots\ X_{T-P}]$, we pass the observed state vector $\mathbf{X}$ to the forecasting model Φ in a seq2seq fashion, and obtain the hidden state h_{T+1} at the end, from the LSTM encoder E. We concatenate it with state vector for the current timestep X_T to produce the input to our ensemble method: $\phi = [h_{T+1} \mid X_T], \phi \in \mathbb{R}^{262}$. We next describe the ensemble method used.

2.2.2 Random-Forest Ensemble Classifier

We employ Random Forest [2] as an ensemble learning approach to combine the raw as well as the extracted temporal features for better fault detection. We aim to leverage both the temporal features extracted from the hidden state of the LSTM, as well as the raw data from the data to train the Random Forest classifier. We use L constituent trees $(D_1, D_2, \dots D_L)$, each of which acts as "weak learners",

Algorithm 3 Feature Extractor LSTM Φ Pretraining

Input: Sequence of feature vectors $\{X^j_{T-P:T}\}, j \in [0, n)$

Output: LSTM Model Φ

function LSTMPRETRAINING($\{X^j_{T-P:T}\}$)
    $\theta_\Phi \leftarrow$ initialize($\mathbf{0}$) ▷ Initialize model weights
    $\eta \leftarrow 0.002$ ▷ Set learning rate
    for $j = 1 \rightarrow n$ **do**
        $(c_0, h_0) \leftarrow$ initialize($\mathbf{0}, \mathbf{0}$)
        pred, gt $\leftarrow$ initialize($\{\}, \{\}$)
        for $i = 0 \rightarrow P - 1$ **do**
            $\hat{X}_{T-P+i+1}, \{c_{i+1}, h_{i+1}\} \leftarrow \Phi(X^j_{T-P+i}, \{c_i, h_i\})$
            pred $\leftarrow$ pred $\cup \{\hat{X}_{T-P+i+1}\}$
            gt $\leftarrow$ gt $\cup \{X^j_{T-P+i+1}\}$
        end for
        loss $\leftarrow \|\text{pred} - \text{gt}\|^2$
        $\nabla\theta_\Phi \leftarrow$ gradient(loss, θ_Φ)
        $\theta_\Phi \leftarrow \theta_\Phi - \eta \cdot \nabla\theta_\Phi$ ▷ Gradient Descent Update using Backpropagation
    end for
end function

often capturing only a subset of features at once. The combined prediction of the trees however can prove to significantly outperform a monolithic decision tree. The training algorithm for the random forest is given in the algorithm 4. At test time, majority voting of the L decision trees decides the class as is given by Eqn. 2.10

$$\hat{Y}_t = \text{mode}\Big(D_1(\phi), D_2(\phi), \dots D_L(\phi)\Big) \tag{2.10}$$

Algorithm 4 Training Random Forest with class-imbalance

Require: Set of tuples $\{(X^j_{T-P:T}, Y_T)\}, j \in (0, n)$, Class Weights: W

Ensure: Random Forest Model $\mathcal{R}$

$\mathcal{R} \leftarrow \{D_1, \dots, D_{20}\}$ ▷ Initialize decision trees

$\theta_{\mathcal{R}} \leftarrow \theta_{\text{pretrained}}$ ▷ Load pretrained weights

$F \leftarrow \{\}$ ▷ Feature data list

for $j = 0 \rightarrow n$ **do**

  $\{c_0, h_0\} \leftarrow \text{initialize}(\mathbf{0}, \mathbf{0})$

  for $i = 0, \dots, P$ **do**

    $(\hat{X}_{T-P+i+1}, c_{i+1}, h_{i+1}) \leftarrow \Phi(X^j_{T-P+i}, c_i, h_i)$

  end for

  $F \leftarrow F \cup \{ [\, h_{P+1} \mid x_T \,] \}$

end for

Train $\mathcal{R}$ with F, Y_T, and W

2.2.3 Addressing Class-imbalance

Class-Imbalance: Sequence-based classification tasks, including those discussed in our paper, frequently encounter significant imbalances in class distributions within the dataset. This is similar to scenarios described in papers [22] and [21], where a simulated flight primarily consists of normal operation time, corresponding to a 'no fault' class, and only briefly exhibits a 'fault in motor' class. Such an imbalance poses a substantial challenge in training effective models, as the overwhelming prevalence of the 'no fault' class can bias the model against accurately identifying the rare 'fault' instances.

Modified Random Forests: To alleviate the effect of this skew on the model performance, we assign weights to the classes. For a decision tree D_i, if is p_j is the proportion of samples of class j at node n, the decision tree algorithm will try to maximize the decrease in impurity when making each split, by

using a quantity called as the Gini impurity. The Gini impurity for a node n is calculated as:

$$I_G(n) = 1 - \sum_{j=1}^{7} p_j^2 \tag{2.11}$$

We introduce class weights w_i for each class, and adjust the impurity calculation as:

$$I_G(n) = 1 - \sum_{j=1}^{7} (w_j \cdot p_j)^2 \tag{2.12}$$

The impurity is now a weighted sum, which biases the tree towards correctly classifying the classes with higher weights, because errors in these classes will now result in a higher impurity. When the decision tree algorithm looks for the best split, this new weighted impurity measure is what's minimized. This affects the thresholds chosen for splits and the selection of features used for those splits.

Choice of Class-Weight w_i: We assign weights proportional to the frequency of each class, giving higher importance to the rare fault classes. Eq. 2.13 is used for calculating w_c for particular class c, using $n_{samples}$, total number of samples in a dataset and n_{count} being the frequency of class, c, is as follows:

$$w_c = \frac{n_{samples}}{n_{count}(c)} \tag{2.13}$$

This approach ensures that rare fault classes receive higher weights, allowing the model to focus on their effective detection, even in cases where they occur infrequently. It contributes to a balanced and robust fault detection and localization model in the presence of class imbalance.

2.3 Implementation and Experimental Setup

2.3.1 Metrics Used

In this section, we discuss the evaluation metrics used to assess the performance of our fault detection and localization model for hexacopter UAVs. These metrics provide insights into the model's accuracy, precision, recall, F1-score, as well as its inference and detection times.

2.3.1.1 Correctness

We evaluate on accuracy, precision, recall and F1-score.

- **Normalized Accuracy:** Accuracy measures the overall correctness of our model's predictions. Weighted accuracy takes into account the class distribution by normalizing for classes that are underrepresented. This method of calculating accuracy ensures that each class contributes equally

to the overall accuracy metric, thus compensating for any class imbalance. Consider the confusion matrix C where the element $C_{i,j}$ would be the number of observations that were originally in classes i but predicted to be in class j. We calculate the accuracy as the average of diagonals of normalised confusion matrix. The true positives a class i are normalized by dividing by the number of instances in actual class i. Mathematically, the accuracy can written in Eq. 2.14.

$$\text{Accuracy} = \frac{1}{N}\sum_{i=0}^{N}\Big(\frac{C_{i,i}}{\sum_{j=0}^{N} C_{i,j}}\Big) \tag{2.14}$$

Here N=7 represents the 6 faulty motors and the no fault class.

- **Precision:** Precision evaluates the model's ability to make accurate positive predictions. It measures the ratio of true positives to the total number of positive predictions made by the model. A high precision is obtained by low number of false positives. In our case, False positive is when there is no fault but algorithm predicts one thus reconfiguration occurs. This not only reduces efficiency of the UAV, but if it occurs more than one time it leads to 2 motor failure which depending on the arrangement is uncontrollable. This has to be minimized but has lesser priority than below sections.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.15}$$

- **Recall:** Recall, also known as Sensitivity, measures the model's ability to correctly identify all positive instances. It is calculated as the ratio of true positives to the total number of actual positive instances. A high recall is obtained by low number of false negatives. In our case, a False negative is when there is a fault but algorithm doesn't detect it would lead to a crash or a huge delay which are undesirable.

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.16}$$

- **F1-Score**: The F1-Score is the harmonic mean of precision and recall. It provides a balanced measure of a model's performance, especially when class imbalance is present. If the wrong motor is classified as faulty then the reconfiguration would essentially make it into 2 motor failure which can be uncontrollable.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.17}$$

2.3.1.2 Latency

The model can incur delay in detecting a fault, either due to computation time or due to the in capabilities of itself. If the delay is large enough the UAV might take a unrecoverable state. Seeing as timely detection is a crucial aspect in such tasks, in addition to the above classification metrics, we also report and compare model performance for fault detection time ie. the amount of time, i.e. the number

of timesteps, between the timestep of the actual failure and the point where failure was first detected by the model.

- **Model Inference Time:** In the context of our system, refers to the duration it takes for our model to predict the results once it's provided with the input data. This metric provides insights into the complexity and computational efficiency of our model, as it measures the response time from input to output.

- **Fault Detection Time:** Detection time is a critical metric in our fault detection and localization system. It signifies the actual time elapsed from the occurrence of a fault, such as a malfunctioning motor, until the system successfully detects and identifies the fault. It's important to note that the Detection Time may vary and depend on the number of time frame data points it takes for the prediction to indicate a fault. This metric plays a crucial role in assessing the system's responsiveness and effectiveness in addressing real-time faults in our hexcopter UAV.

2.3.2 Baseline Models

In this section, we present the extensive experiments conducted to evaluate the effectiveness of our fault detection and localization system for hexcopter UAVs. Our evaluation encompasses a wide array of models, ranging from statistical approaches to deep learning networks.

- **Statistical Models**: In our analysis, we evaluate statistical models, including **Logistic Regression**[35], **Support Vector Machine (SVM)**[36], **Random Forest**[2], and **Rotational Forest**[37], on the dataset described above. These established models provide valuable insights into the dataset's fundamental characteristics and their simplicity and efficiency in fault detection. These act as a good baseline for comparison.

- **MLP Classifier**: The architecture of the Multi-Layer Perceptron (MLP) for fault localization consists of four linear layers of size 256, 128, 128, and 64 neurons, respectively. Each linear layer is followed by a batch Normalization[38] and the ReLU activation function [39]. The input to the MLP network is $\mathbf{X} \in \mathbb{R}^{P \times 6}$.

- **LSTM Classifier**: We train a Long Short-Term Memory (LSTM) model offline in a seq2seq fashion. We then freeze the weights of LSTM and train a Logistic Layer, which takes the hidden state h_{t+1}, applies a linear layer followed by a Softmax and outputs a probability vector of size 7 (number of classes). The LSTM is trained on MSE loss, and the logistic layer is trained on Cross-entropy loss. In this model, in contrast to our approach, we do not apply Random Forest, and instead pass the hidden state from LSTM directly to a linear layer.

2.3.3 Implementation Details

In our approach, the LSTM architecture, as described earlier, remains consistent. However, its purpose is distinct. Instead of predicting the faulty motor, the LSTM is pre-trained on the dataset using X_t, h_t as input and predicting the next feature vector h_{t+1} of size 256. This pretraining process enables the network to learn and encode temporal dependencies effectively. After the LSTM layer, we added a dense layer of size 6 predicting the next data vector, X'_{t+1}, from UAV. It is different from the dense layer added in the previous section. As mentioned, the loss used to pre-train the LSTM is MSE loss. Subsequently, the last layer of the LSTM is utilized to extract essential temporal features. These feature vectors are concatenated with a range of raw features, X_t i.e. combining the LSTM-learned temporal insights with the original data. As in described in Fig. 2.2 demonstrating the end-to-end pipeline, the concatinated features are passes to the ensembled classifier.

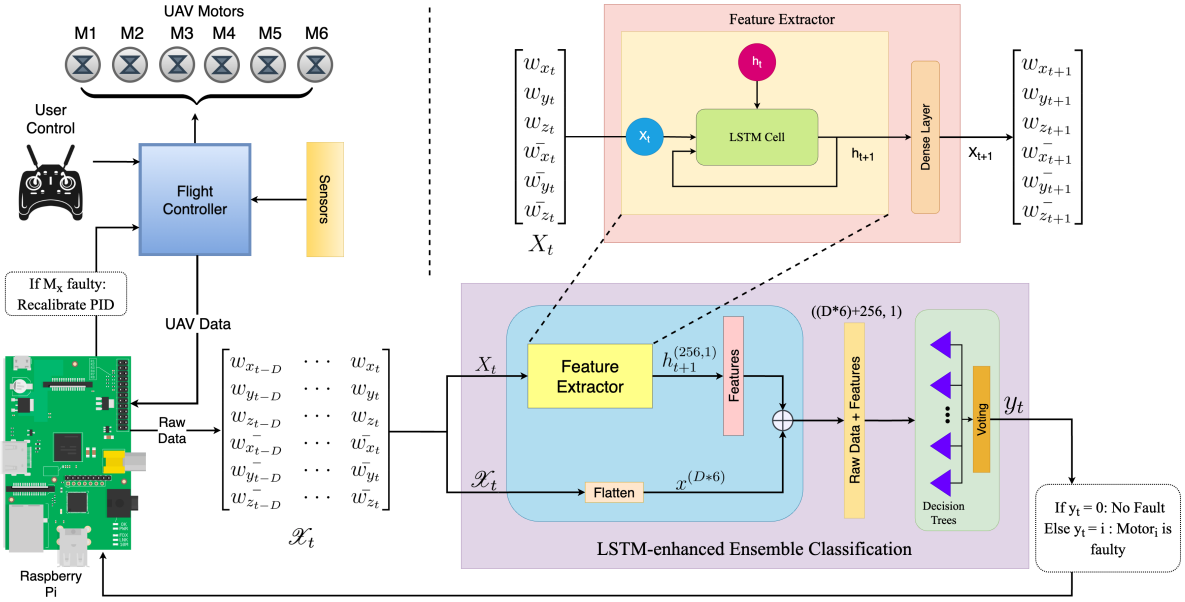


Figure 2.2: **Ensemble-Classifier LSTM Pipeline**: Hexacopter system onboard consists of on-board microcontroller and a flight controller chip. The sensors onboard collect the time-series flight data $\mathbf{X}$. We propose an Ensemble classifier which is trained over the temporal features ϕ derived from the LSTM state and current state. During inference, our proposed Ensemble Classifier-enhanced LSTM takes the data $\mathbf{X}$, and predicts the states for the next timestep $\hat{X}_{T+1}$ in a sequence-2-sequence (seq2seq [1]) fashion. The hidden state updated till the end h_{P+1} is passed onto the Random Forest [2] to obtain the fault detection and localization $\hat{Y}_T$. If a fault is detected, based on the motor localized, the onboard microcontroller can give corrective signals to the flight controler chip and reconfigure to resume stable flight.

Training Details: For experimentation purposes, we have used the P as 3, batch size as 8. We use the Adam optimiser [40] with the learning rate of 0.0002. For the task of classification, we have used cross-entropy loss and for pre-training the LSTM, the mean-square error (MSE) loss is used. For the Random Forest, we use $L = 20$ decision trees.

2.3.4 Dataset

Train data collection: We collect data in the Gazebo Simulation Framework [41]. We setup a custom PX4 firmware to introduce fault, and chose the vehicle as Typhoon H480 whose parameters are described in table 2.1. Our mission in simulation consists of UAV take off followed by hovering for 5 seconds, wherein we introduce a fault. We do 10 such missions for each motor failure with 90% signal-to-noise ratio. Note that for every mission, we only introduce fault in a single motor. Fig. 2.3,2.4,2.5 illustrates some of the cases of such data. These graphs provided represent the real-time monitoring data for a hexacopter's three motors during a test sequence.

Parameter	Description	Value
J_{xx}	Inertia about x-axis	6.89×10^{-3} kg.m^2
J_{yy}	Inertia about y-axis	6.89×10^{-3} kg.m^2
J_{zz}	Inertia about z-axis	3.44×10^{-2} kg.m^2
l	Arm length	0.33 m
J_r	Rotor inertia	6×10^{-4} kg.m^2
m	Hexacopter Mass	0.95 kg
k_T	Aerodynamic Force constant	3.13×10^{-4} N.s^2
k_M	Aerodynamic Moment constant	7.5×10^{-6} Nm.s^2
R_{mot}	Motor circuit resistance	$0.6\,\Omega$
K_{mot}	Motor torque constant	5.2 mNm/A

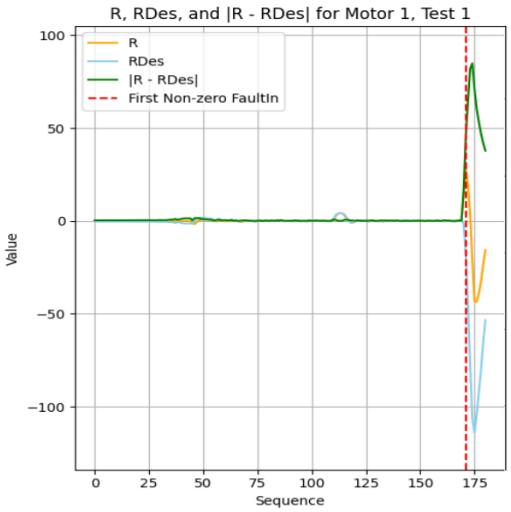
Table 2.1: Hexacopter Parameter Description and Value

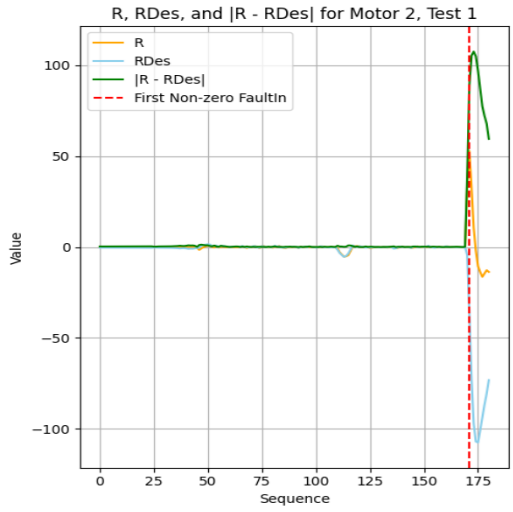
Test data synthesis: To create test data fig. 2.6, we further add Gaussian noise η to stimulate real world conditions [42] and increase the signal-to-noise ratio. Given train data S with distribution $N(\mu, \sigma)$, we sample a Gaussian and generate a noise signal η, such that -

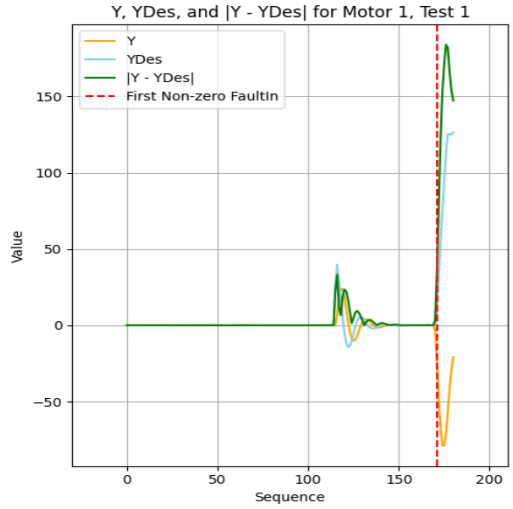
$$S' = S + \eta \tag{2.18}$$

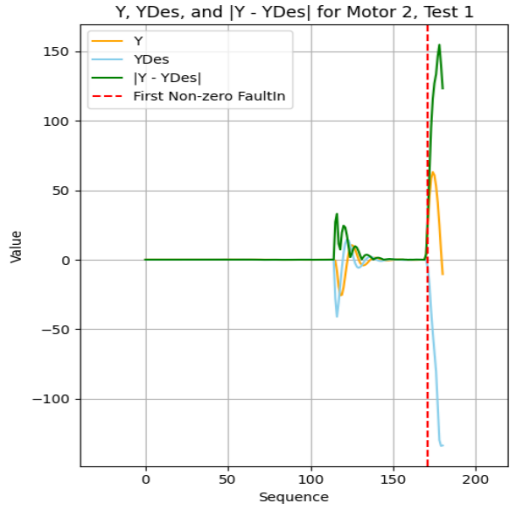
$$\eta \sim N\Big(0, \frac{\sigma}{k}\Big) \tag{2.19}$$

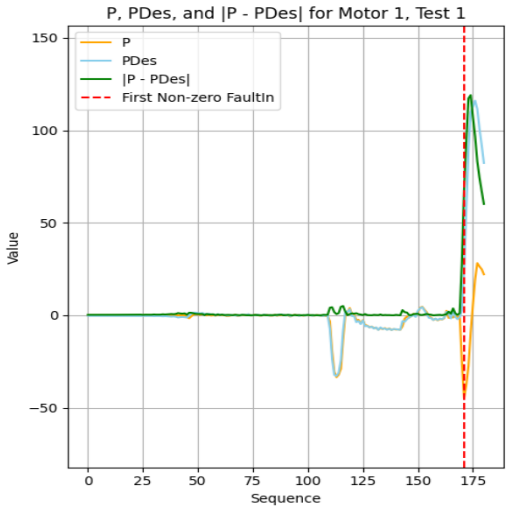
Here k is the Signal-to-Noise Ratio (SNR) of the resulting signal S'. We perform experiments with varying signal-to-noise ratios, and demonstrate that our method's tolerance to higher values of k.

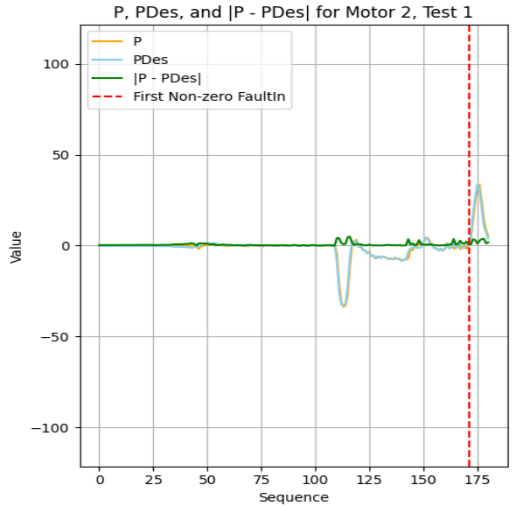








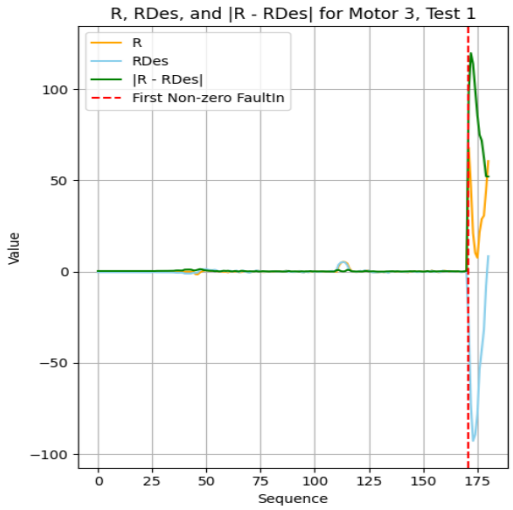


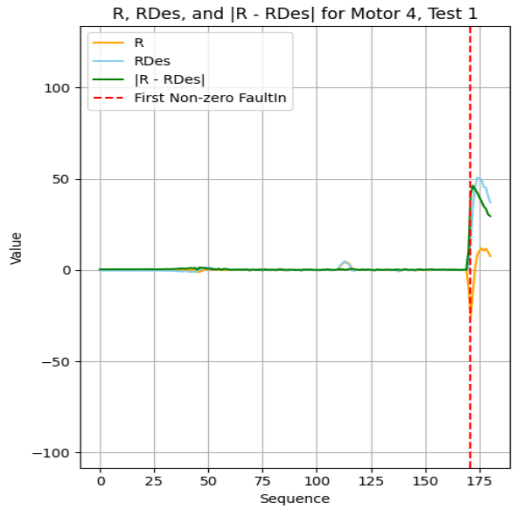


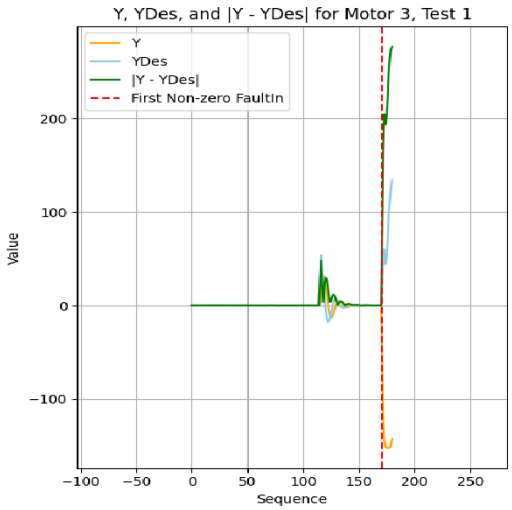
(a) Motor 1

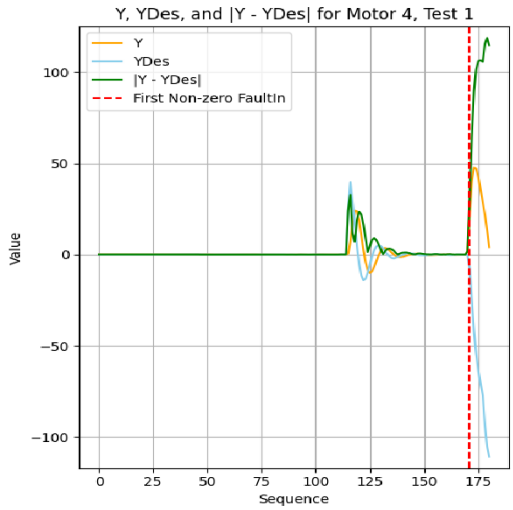
(b) Motor 2

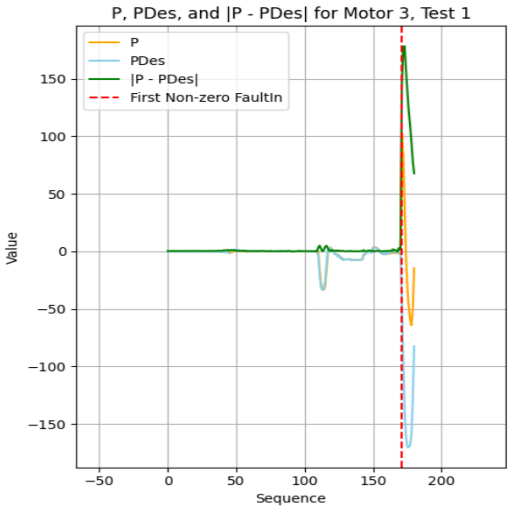
Figure 2.3: **Training Simulated Data:** For Motor 1 (Fig. (a)) and Motor 2 (Fig. (b)) during simulated flight 1, each subfigure describes the roll (R), yaw (Y), and pitch (P) angular rates with respect to the body frame.

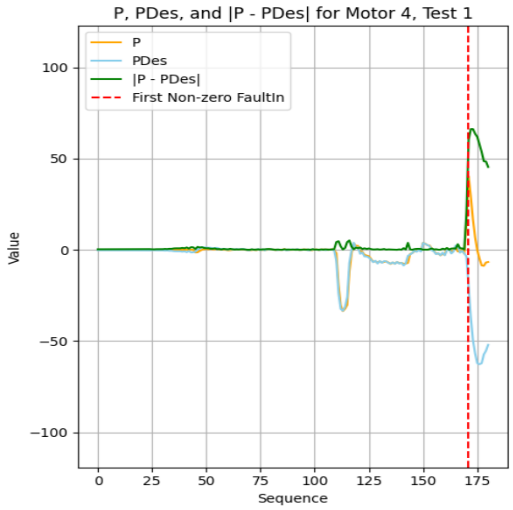








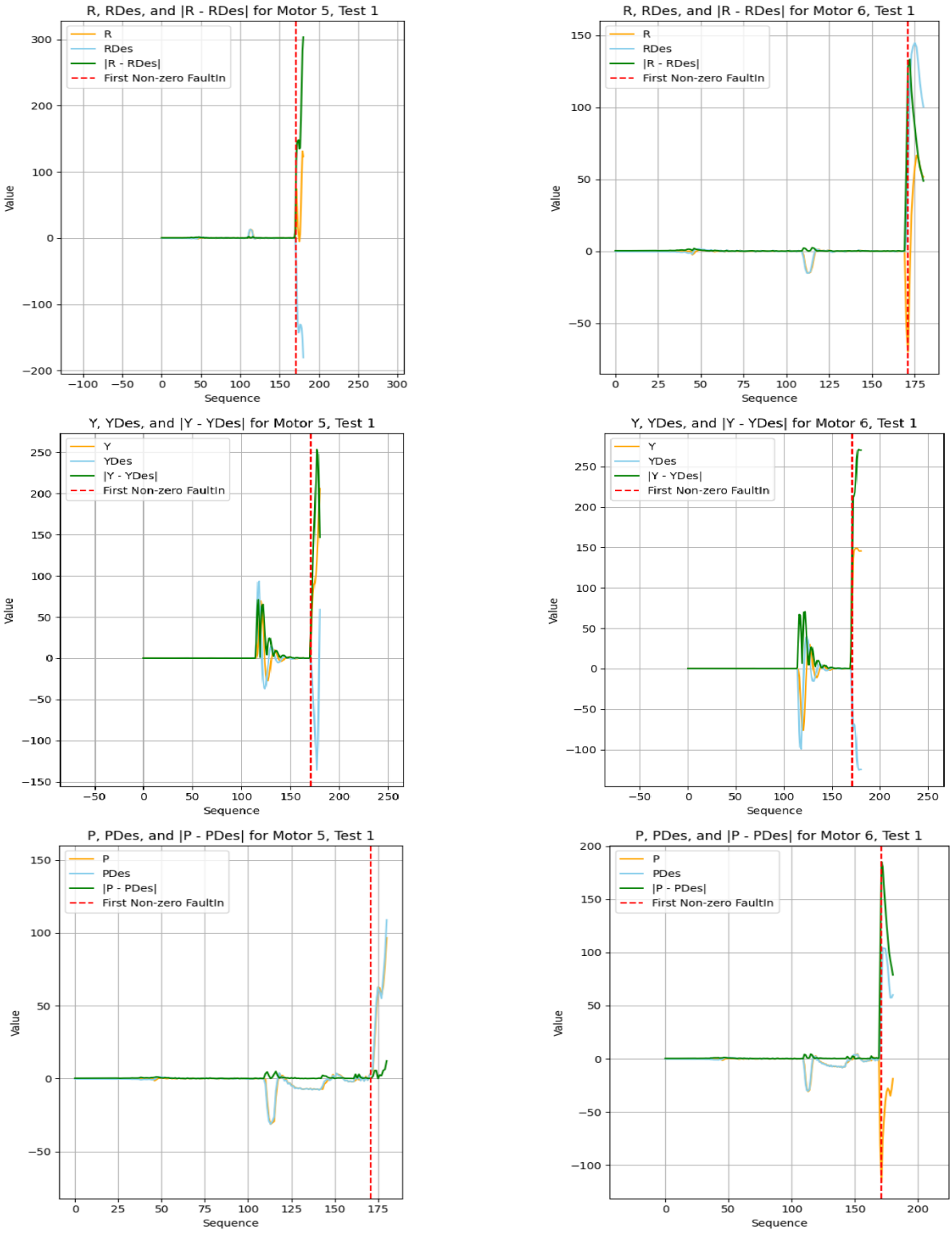




(a) Motor 3

(b) Motor 4

Figure 2.4: **Training Simulated Data:** For Motor 3 (Fig. (a)) and Motor 4 (Fig. (b)) during simulated flight 1, each subfigure describes the roll (R), yaw (Y), and pitch (P) angular rates with respect to the body frame.



(a) Motor 5

(b) Motor 6

Figure 2.5: **Training Simulated Data:** For Motor 5 (Fig. (a)) and Motor 6 (Fig. (b)) during simulated flight 1, each subfigure describes the roll (R), yaw (Y), and pitch (P) angular rates with respect to the body frame..

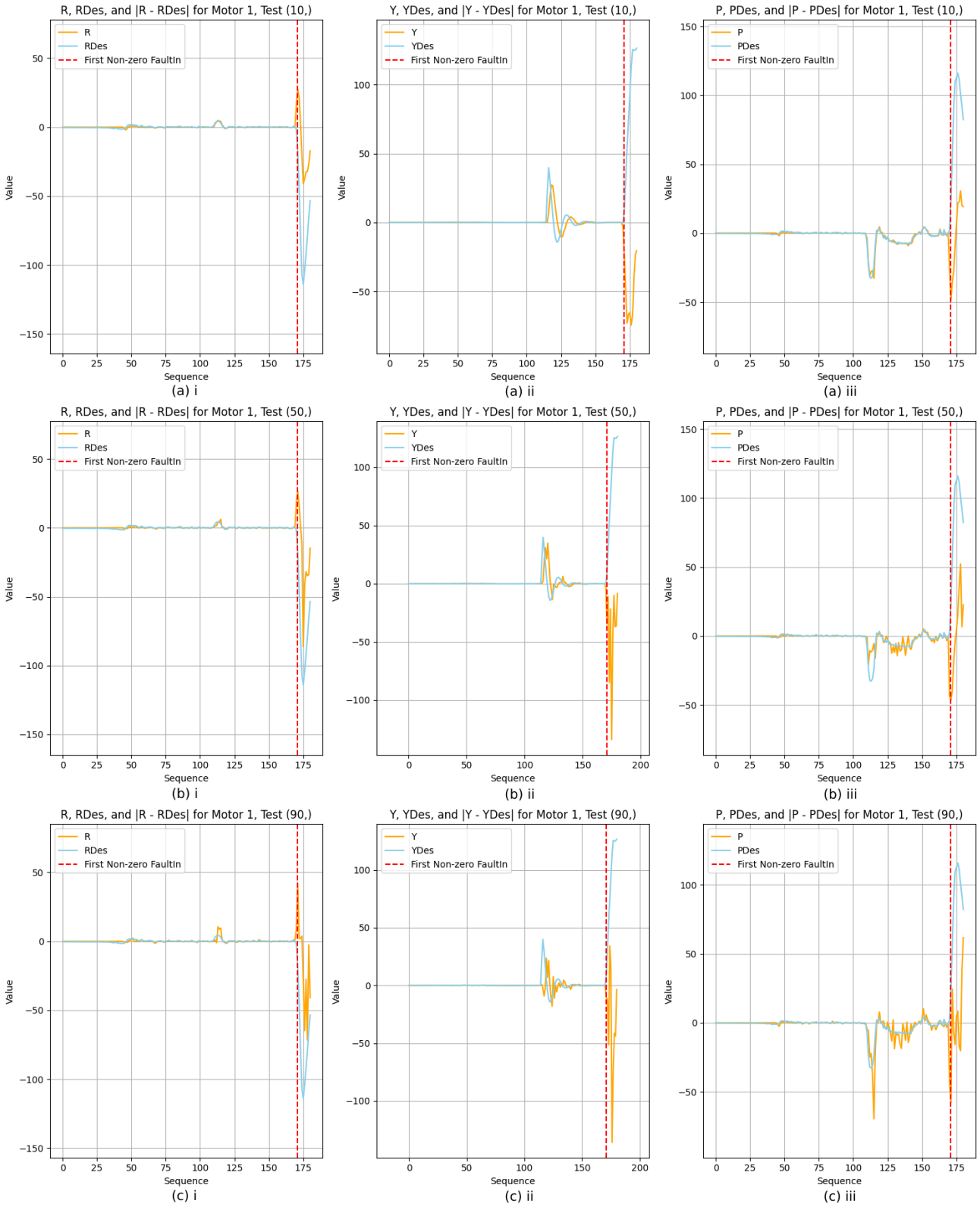


Figure 2.6: **Test Data with Noise:** For Motor 1, (a), (b), and (c) represent the varying levels of noise at 10%, 50%, 90%, respectively. Within each fig., i, ii, and iii are the roll(R), yaw(Y), and pitch(P) angular rates.

Chapter 3

Results and Analysis

Here we presents a comprehensive analysis of the results obtained from both simulated and real-world datasets, along with an evaluation of the system's performance on a Raspberry Pi platform. The chapter is structured into several sections, each focusing on distinct aspects of the experiment. Initially, the overall performance of the system is discussed, highlighting key metrics and observations derived from the simulated dataset. Subsequently, the analysis delves into motor-wise performance and noise tolerance, shedding light on the system's robustness in varying conditions. Transitioning to real-world scenarios, the chapter outlines the dataset collection process and experiment setup for outdoor conditions, followed by a comparative analysis of model performance. Furthermore, insights into motor-wise results and detection time are provided. Lastly, the chapter concludes with an exploration of the system's performance on a Raspberry Pi, encompassing experimental setups and statistical analyses of inference time. This comprehensive examination aims to provide a holistic understanding of the proposed system's efficacy across different environments and platforms.

3.1 Results on Simulated Dataset

In this section, we present the testing results on simulated data, comparing various baseline models with our proposed model. We initiate the analysis by evaluating the overall performance and comparing metrics defined in Chapter 2. Additionally, we delve into motor-wise performance for a comprehensive examination of various baseline models, encompassing classical models such as Logistic Regression, SVM, Random and Rotation Forest, as well as deep learning approaches like MLP, LSTM, and our proposed model. Moreover, a crucial aspect of our evaluation involves conducting noise tolerance analysis. Here, we assess the robustness of our model by subjecting it to real-world-like noise introduced to the testing data. This analysis is essential for understanding how well our model can handle variations and disturbances, mimicking the challenges encountered in practical scenarios.

Finally, we conduct experiments on detection time for the simulated dataset. This involves measuring the number of time steps required for each model, including our proposed one, to detect faults within

the simulated data. Examining detection time is crucial for applications where timely responses to anomalies are essential, ensuring the efficiency and effectiveness of the models in real-world scenarios.

3.1.1 Overall Performance

In Table 3.1, we present a comprehensive benchmark comparison of various models for fault detection and localization. It demonstrates the effectiveness of our method in accurately identifying faulty motors and localizing faults, even in the presence of complex real-world scenarios.

	Logistic Regression	**SVM**	**Random Forest**	**Rotation Forest**	**MLP Classifier**	**LSTM Classifier**	**Ours**
Accuracy	0.71	0.71	0.76	0.81	0.87	0.94	**0.968**
Precision	0.73	0.71	0.79	0.80	0.81	0.91	**0.966**
Recall	0.63	0.69	0.72	0.82	0.85	0.94	**0.972**
F1-Score	0.67	0.70	0.75	0.81	0.83	0.93	**0.969**

Table 3.1: **Benchmark Comparison**: The table denotes results aggregated over all types of faults on motors and no fault in the dataset mentioned above. Our method outperforms the baselines in all metrics.

Despite the similar accuracies of 70.97% and 71.42% for Logistic Regression and SVM, their precision and recall values differ significantly. Logistic Regression tends to be conservative, making fewer positive predictions, but these predictions have a higher confidence level in their correctness. In contrast, SVM is slightly more liberal in making positive predictions, leading to different precision and recall values. Random Forest and Rotation Forest exhibit better accuracies than SVM, hovering around 75-80%. Rotation Forest performs better than Random Forest due to its feature transformation techniques, as discussed in [22]. The MLP classifier achieves a significantly better accuracy(87.29%) than statistical models above. This performance gap is due to their limitations in effectively capturing temporal features, which are crucial for fault detection and localization.

The LSTM classifier model fairs further better in performance than MLP classifier. This gain can be attributed to its specialized architecture designed for temporal sequence modeling. Its role as a feature extractor is pivotal in transforming raw sensor data into meaningful representations. The LSTM excels in identifying actual positive cases (recall of 0.941), minimizing the chances of missing faulty motors. Our approach, LSTM+RF, combines the strengths of LSTM's feature extraction and temporal modeling with the robust classification capabilities of Random Forest. Our method outperforms all other models not only along the overall accuracy (96.83%), but also beats past approaches along precision and recall.

3.1.2 Motor-Wise Performance

Across the different motor types, the results in table 3.2 show that the classical methods, including Logistic Regression, SVM, and Random Forest, perform relatively well in situations with 'No Faulty Motor' ($Y_t = 0$) achieving high accuracy close to 1.0. However, their performance drops significantly when it comes to accurately detecting and localizing faulty motors. For instance, the accuracy of Logistic Regression and SVM drops to around 0.70 to 0.78 for Motor 1, indicating that these methods have difficulty distinguishing between faulty and non-faulty motors. On the other hand, deep learning approaches, specifically MLP classifier and LSTM, exhibit significantly improved performance across all motor types, with accuracy consistently close to 1.0. This demonstrates their capability to effectively detect and localize faults in various motors. Our proposed approach outperforms all other methods, achieving perfect accuracy (1.0) for 'No Faulty Motor' and near-perfect accuracy for the detection and localization of faulty motors, with accuracy ranging from 0.88 to 0.99 across different motors M_i. These results emphasize the effectiveness of our approach, where a combination of LSTM feature extraction and Random Forest classification excels in fault detection and localization tasks for hexacopter motors.

		Motor No Fault	Motor 1	Motor 2	Motor 3	Motor 4	Motor 5	Motor 6
Classical Approach	Logistic Regression	0.95	0.70	0.24	0.75	0.80	0.61	0.92
	SVM	0.98	0.78	0.39	0.72	0.65	0.68	0.80
	Random Forest	1.00	0.76	0.55	0.83	0.72	0.63	0.86
	2-Stage Rotation Forest	1.00	0.93	0.42	0.92	0.88	0.57	0.94
Deep Learning Approach	ANN	1.00	0.82	0.77	0.72	0.93	0.91	0.96
	Vanilla-LSTM	1.00	0.97	0.91	0.98	0.96	0.85	0.89
	Ours (LSTM+RF)	**1.00**	**0.99**	**0.95**	**0.99**	**1.00**	**0.88**	**0.97**

Table 3.2: **Motor-Wise Accuracy**: The table presents a comprehensive comparison of models' performance for fault detection and isolation across various motors, including scenarios where there are 'No Faults.' The models evaluated include both classical statistical methods (Logistic Regression, SVM, Random Forest, Rotation Forest) and deep learning approaches (ANN, LSTM), with our proposed approach highlighted for comparison.

3.1.3 Noise Tolerance Analysis

This section seeks to evaluate the robustness of various fault localization models under different noise conditions, emphasizing the importance of noise tolerance in ensuring reliable UAV operations in real-world, unpredictable environments. We performed an analysis by examining the impact of varying noise levels on the performance of different models on the task of fault localization. As we increase the noise level, we observe a consistent decline in accuracy across all approaches, which is an anticipated outcome. However, it's worth noting the significant variations in accuracy among the different methods. Random Forest, (depicted by blue), exhibits a steady decline in accuracy as noise levels increase. It remains relatively stable until the noise level reaches 100%, after which it experiences a sharper drop whereas rotation forest, (depicted in orange), displays a more resilient performance, maintaining a higher accuracy even as noise levels rise. This method proves to be more noise-tolerant compared to random forest. MLP (green) and LSTM (red) classifier approaches demonstrate the advantages of deep learning approaches. Both methods maintain reasonably high accuracy levels, even at elevated noise levels. They showcase robustness and resilience to noise, highlighting their potential in real-world, noisy UAV environments. Our proposed approach (purple) consistently outperforms all other methods, maintaining the highest accuracy across the entire noise spectrum. Even at the highest noise level of 180%, our approach maintains a remarkable accuracy level, emphasizing its adaptability and noise-resilient characteristics. Our innovative approach leverages the LSTM's pretraining capabilities for feature extraction and complements them with traditional features, contributing to enhanced fault localization in hexacopter UAVs.

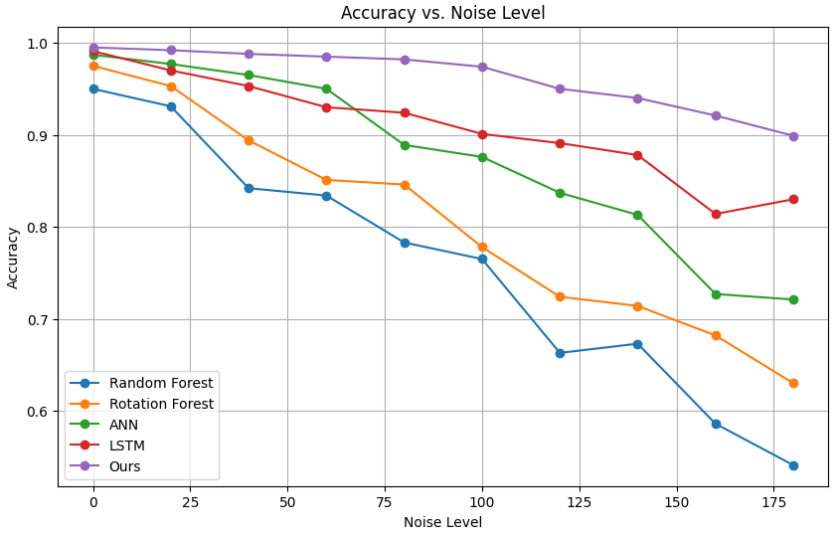


Figure 3.1: **Accuracy with Noise Level:** The x-axis represents the range of noise levels, spanning from 0% to 180%, in 20% increments. The y-axis shows the accuracy of each approach. The proposed LSTM-based fault localization approach outperforms traditional methods.

3.1.4 Detection Time

We showcase the detection time within a sequence in Fig. 3.2. Our classifier is able to detect and localize the fault within 3-5 timesteps of introduction. The UAV thus has a buffer of $3\Delta t$ to $5\Delta t$ delay from the time the fault was introduced, so that the UAV can reconfigure its control allocation matrix, and ensure a stable system.

3.2 Results on Real-World Dataset

The collection and analysis of real-world outdoor data for fault detection in UAVs represent a critical step towards understanding model performance in practical scenarios. Despite the challenges posed by real-world conditions, such as noise and dynamic environmental factors, the insights gained from this data contribute to the development of robust fault detection algorithms. Through deliberate fault introduction and data collection, we simulate real-world scenarios, providing valuable insights into model performance and its ability to generalize beyond simulated data. Additionally, our models undergo benchmarking against alternative methodologies, enabling a comparative evaluation of their performance. We conduct an in-depth analysis of motor-wise performance and examine the number of detection steps necessary for accurate fault identification.

3.2.1 Outdoor Dataset Collection

Real-world data is collected through a series of multiple test flights conducted on a hexacopter. The hexacopter ZD850 Carbon Fiber Frame Hexacopter 850 [43], characterized by parameters outlined in Table 3.3, is launched at a height ranging from 1 to 3 meters from the ground and placed on hold. Subsequently, a fault is deliberately introduced in one of the motors. Data is then gathered while the UAV is stabilized within a timeframe of 2 seconds.

We present a selection of real-world flight logs in Figure 3.4. It is evident that these logs exhibit a significantly higher level of noise in comparison to simulated data. However, they closely resemble the noisy data employed for testing our model. This disparity in noise levels may be attributed to various factors inherent in real-world flight environments, such as atmospheric disturbances, sensor inaccuracies, and mechanical imperfections in the UAV hardware.

Furthermore, an observation can be made regarding the instability of the desired control by the controller[44], which contrasts with the smoother trajectories evident in the simulated data. This discrepancy in control stability could stem from differences in the dynamic response of the UAV between simulation and reality. Factors such as unmodeled aerodynamic effects, varying payload configurations, and environmental conditions not accurately represented in simulation may contribute to the observed instability in real-world control.

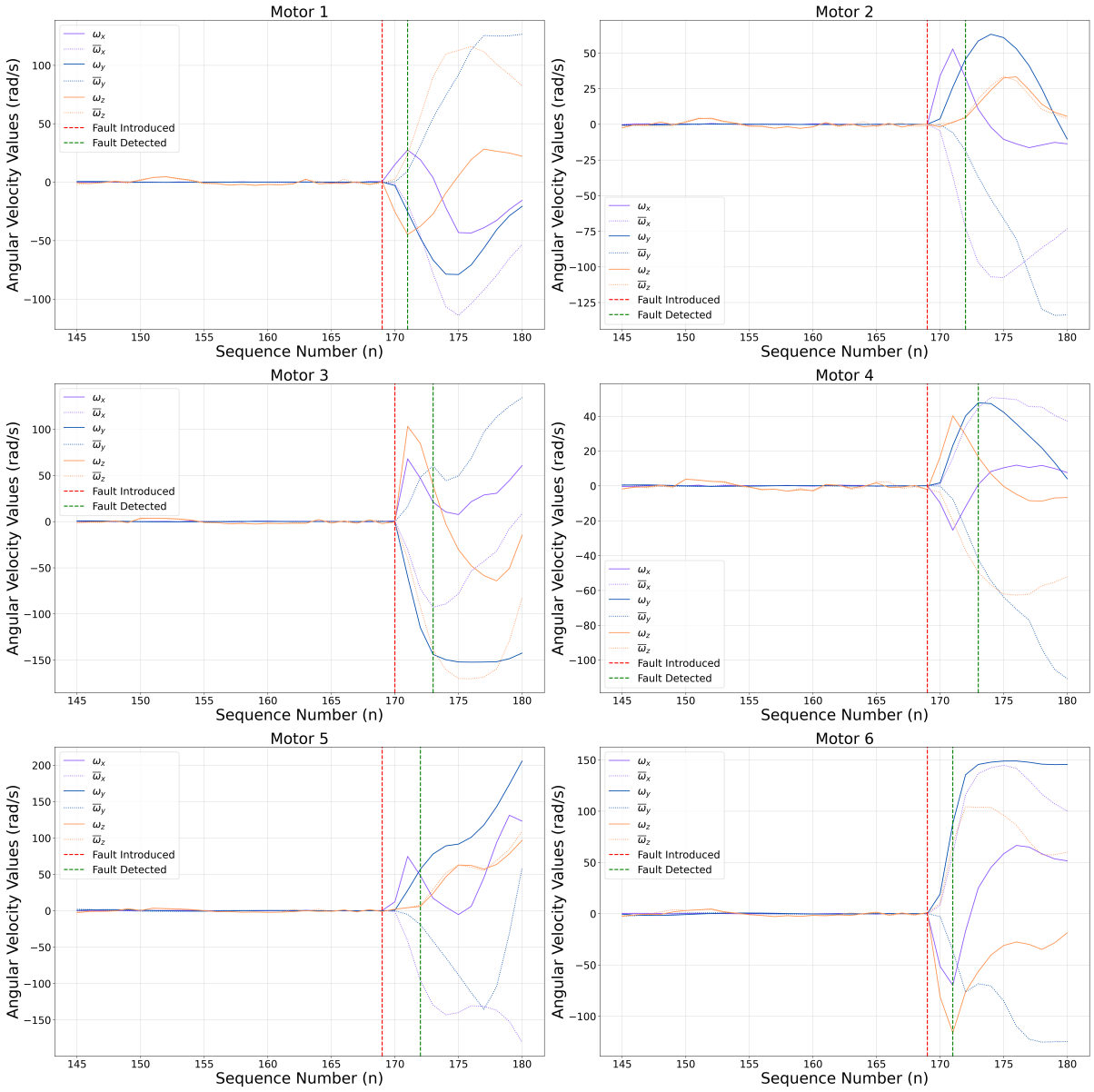


Figure 3.2: **Trajectory Features and Fault Detection Motorwise:** The above graphs showcase trajectory features with time steps (sequence numbers) along with fault introduction and detection times. The x-axis indicates the sequence number (n), with each unit corresponding to a time-step at a rate of 36 frames per second, thus showcasing a truncated sequence of the full operation. The y-axis displays the angular velocity values in radians per second (rad/s). Red vertical lines represent the sequence step at which a fault is introduced, and green vertical lines denote the step at which the fault is detected by our model. Our detection system identifies the occurrence of a fault within 2-5 frames of its inception.

Figure 3.3: Hexacopter UAV ZD850 used for data collection.

3.2.2 Experiment Setup

The flight sequences are partitioned into a training-test split with a ratio of 70%:30%. Similar to the methodology employed for simulated data, each data point is constructed with P time steps, encompassing the current time step T and the preceding T-P time steps. These data points are utilized to predict the state of the drone at time T. The state of the drone may encompass either a ”No Fault” state denoted by 0, or a state belonging to the set 1, 2, ..., 6, indicating a fault in the corresponding motor.

We present a comprehensive analysis of the overall and motor-wise performance against our benchmark models, which encompass classical models such as Logistic Regression, Support Vector Machine (SVM), Random Forest, and Rotation Forest. Additionally, we include deep learning models such as Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM) networks in our evaluation. Furthermore, we observe and analyze the number of time steps required for our model to predict faults. This metric holds crucial significance as it reflects the temporal aspect of fault prediction. Understanding the number of time steps necessary for fault detection aids in assessing the responsiveness and efficiency of the model in identifying potential issues in real-time scenarios.

3.2.3 Model Performance Comparison

Table 3.5 analyse model performance on real-world data for fault detection and localization highlights distinct challenges compared to simulated data, primarily due to the complexities and noise inherent in real-world scenarios. Logistic Regression (LR) and Support Vector Machines (SVM) exhibit significantly reduced effectiveness on real-world data, with accuracies of 34% and 21%, respectively. Their limited ability to handle the non-linear patterns typical of real-world data results in poor fault detection capabilities. Random Forest (RF) and Rotation Forest (RoF) perform better, with accuracies

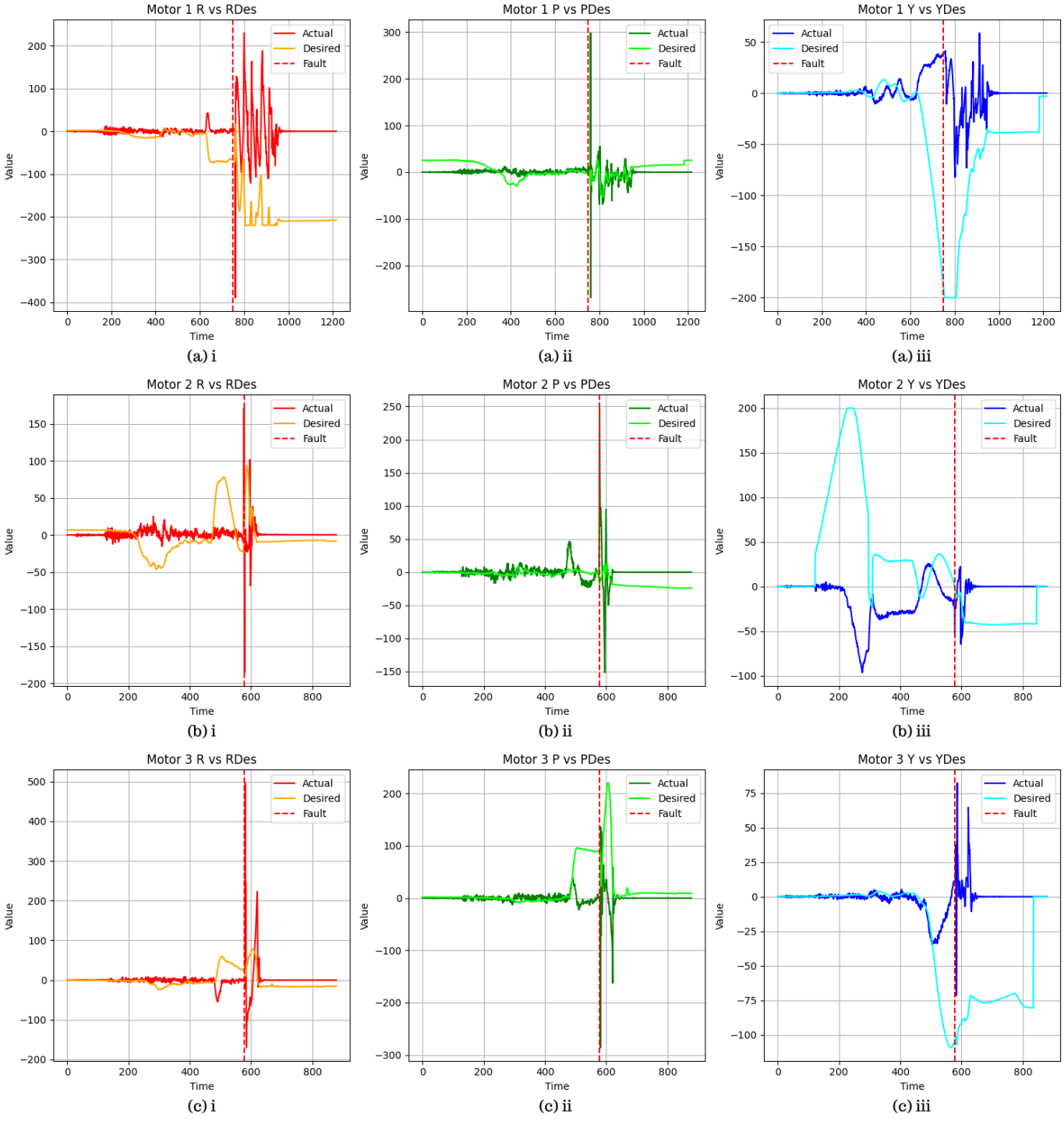


Figure 3.4: The figure presents real-world data illustrating UAV Hexcopter motion, detailing the actual and desired values for roll (R), yaw (Y), and pitch (P). As evident, real data often exhibits significantly more noise and uncertainty in behavior compared to simulated datasets.Figures (a), (b), and (c) denote the data for Motors 1, 2, and 3, respectively, with subfigures (i), (ii), and (iii) corresponding to Roll, Pitch, and Yaw, respectively.

Table 3.3: Hexacopter parameters

Parameter	**Value**
Frame Material	Carbon Fiber
Frame Size	850mm
Carbon Fiber Arm Thickness	20mm
Tripod Stand Tube Thickness	16mm + 10mm
Total Frame Weight	1210 grams
Propeller Size	15 inches
Motor Size	34mm
Motor	Emax ECO II Series 2807
Motor Testltem	1500KV
Motor Weight	46.9g
Flight Controller	px4 cauv5 nano

of 45% and 61%. These methods, especially RoF with its feature transformation strategy, show improved handling of real-world data complexities but still fall short of their simulated data performance. The Multi-Layer Perceptron (MLP) and Long Short-Term Memory (LSTM) networks mark a further improvement, with accuracies of 69% and 74%, respectively. Their advanced capabilities in capturing non-linear relationships and temporal dependencies demonstrate their suitability for complex real-world applications.

Our approach, which combines LSTM's temporal feature extraction with robust classification, outperforms all others with an 83% accuracy. This high performance indicates a substantial step forward in fault detection and localization under real-world conditions, showcasing our method's ability to not only accurately identify faults but also minimize false positives, a critical aspect in practical scenarios.

3.2.4 Motor-wise Results

The tables 3.6 provided offer a comparative analysis of fault detection and isolation models' performance across different motors, encompassing scenarios with and without faults, based on both real-world. In the case of real-world data, classical approaches like Logistic Regression and SVM exhibit varied performance across different motors, highlighting challenges in generalizing well to real-world conditions. While 2-Stage Rotation Forest and Random Forest show improvements, they still encounter difficulties in accurately detecting faults across all motors. On the other hand, deep learning approaches such as ANN and Vanilla-LSTM demonstrate higher accuracy, indicating their potential to capture com-

Parameter	ICM-20602	ICM-20689	BMI055
Accelerometer			
Acceleration Ranges	±2g, ±4g, ±8g, ±16g	±2g, ±4g, ±8g, ±16g	±2g/±4g/±8g/±16g
Filter Bandwidths	100 Hz	1kHz $-<$ 3.91Hz	1kHz $-<$ 8Hz
On-chip FIFO	1 KB	4 Kbyte	32 frames depth
Temperature Sensor	16-bit ADCs	16-bit ADCs	8-bit
Ultra-low Power IC	9.4-171.1 µA	8.4-194.9 µA	130µA
Gyroscope			
Ranges	±250,500,1000,2000/s	±250,500,1000,2000/s	±125°/s to ±2000°/s
Low Power IC	0.7-1.78 µA	1.3-2.6 µA	$<$ 5mA

Table 3.4: Comparison of Accelerometer and Gyroscope Specifications present in flight controller **px4 cauv5 nano**[3] which has 3 sensors ICM-20602,ICM-20689, and BMI055.

	Logistic Regression	SVM	Random Forest	Rotation Forest	MLP Classifier	LSTM Classifier	Ours
Accuracy	0.34	0.21	0.45	0.61	0.69	0.74	**0.832**
Precision	0.26	0.29	0.43	0.72	0.75	0.83	**0.922**
Recall	0.34	0.21	0.45	0.61	0.70	0.74	**0.857**
F1-Score	0.29	0.24	0.44	0.66	0.72	0.78	**0.888**

Table 3.5: **Outdoor Benchmark Comparison**: This table presents aggregated results across various fault types in outdoor motor conditions, as well as scenarios without faults, using the dataset specified.

plex patterns present in real-world data. Notably, our proposed LSTM+RF approach consistently outperforms other models, achieving remarkable accuracy in both fault and no-fault scenarios.

The disparity in performance between real-world and simulated data underscores the importance of evaluating models in real-world conditions to ensure their practical applicability. Despite the challenges posed by real-world data, the LSTM+RF model emerges as a promising solution, offering reliable performance across different data settings and demonstrating its potential for real-world fault detection and isolation tasks.

		Motor No Fault	Motor 1	Motor 2	Motor 3	Motor 4	Motor 5	Motor 6
Classical Approach	Logistic Regression	0.96	0.03	0.90	0.12	0.05	0.33	0.02
	SVM	0.98	0.04	0.02	0.11	0.02	0.29	0.02
	2-Stage Random Forest	0.98	0.05	0.89	0.07	0.83	0.26	0.07
	Rotation Forest	0.99	0.10	0.98	0.17	0.50	0.83	0.68
Deep Learning Approach	ANN	1.00	0.01	0.98	0.52	0.92	0.99	0.41
	Vanilla-LSTM	0.99	0.10	0.99	0.16	0.99	0.99	0.98
	Ours LSTM+RF	**1.00**	**0.87**	**0.98**	**0.19**	**0.99**	**0.82**	**0.98**

Table 3.6: **Outdoor Motor-Wise Accuracy**: This table compares fault detection and isolation accuracy across motors for various models, including classical methods (Logistic Regression, SVM, Random Forest, Rotation Forest) and DL approaches (ANN, LSTM), highlighting our proposed approach.

3.2.5 Detection Time

We conducted an analysis to determine the temporal extent required by our model to anticipate faults subsequent to their occurrence. Table 3.7 delineates and contrasts the Median Detection Steps observed for both Simulated and Real Data within our model framework, utilizing a data window encompassing 5 timesteps. As evident from the analysis, the model exhibited prompt fault detection in the case of simulated data, characterized by sudden and drastic changes in values. This efficiency in detection can be attributed to the model's ability to recognize abrupt deviations from normal operating conditions. However, the scenario differs significantly in real-world data, where changes in rotational values tend to occur gradually over time.

The table presents data on the median detection steps, along with lower and upper bounds, for six different motors under both simulated and real conditions, as well as the number of datapoints available for each motor. The median detection steps vary across motors and conditions, indicating differences in performance. Motors like M1 and M6 exhibit notable changes in detection steps between simulated and real conditions, while others, like M3, demonstrate consistency. The bounds provide insight into the variability of detection steps, with some motors showing wider ranges than others. Overall, the table offers valuable insights into the performance characteristics of these motors, highlighting potential areas for further analysis and comparison.

Motor	Median Detection Steps (Simulated)	Median Detection Steps (Real)	Lower Bound	Upper Bound	Number of Datapoints
M1	2	6	2	8	3
M2	3	6	3	9	7
M3	3	4	0	6	2
M4	4	5	1	7	6
M5	3	4	1	7	9
M6	2	6	2	8	3

Table 3.7: Median Detection Steps and Bounds for Each Motor. The timesteps are measured at a sampling rate of 50Hz.

In conclusion, the analysis of real-world outdoor data underscores the importance of evaluating fault detection models under realistic conditions. By comparing model performance across different motors and fault scenarios, we gain valuable insights into their effectiveness and generalizability. These findings pave the way for further refinement and optimization of fault detection algorithms, ultimately enhancing the reliability and safety of UAV systems in real-world applications.

3.3 Performance on Raspberry Pi

The efficiency of fault detection algorithms in real-time scenarios holds significant importance. In this section, we present a detailed account and analysis of the real-time experiments conducted. Our experiments were conducted on a Raspberry Pi microcontroller, which was interfaced with a UAV flight controller, as depicted in Figure 2.2. We evaluate the time performance comprehensively, encompassing both overall system performance and individual motor performance. Additionally, we delve into the statistical distribution of time-related metrics.

3.3.1 Experiment Setup

We conducted time experiments using two distinct computing platforms: the Raspberry Pi 3 [45] and the Apple MacBook Pro. Table 3.8 provides detailed specifications for both machines, highlighting disparities in processing speed and architectural nuances.

The Raspberry Pi 3, chosen as the onboard computer, offers several advantages conducive to our fault detection algorithm's real-time execution. Firstly, its compact size and low power consumption make it well-suited for integration into UAV systems. Furthermore, the Raspberry Pi's ARM-based architecture aligns closely with embedded systems, facilitating seamless interfacing with the UAV flight

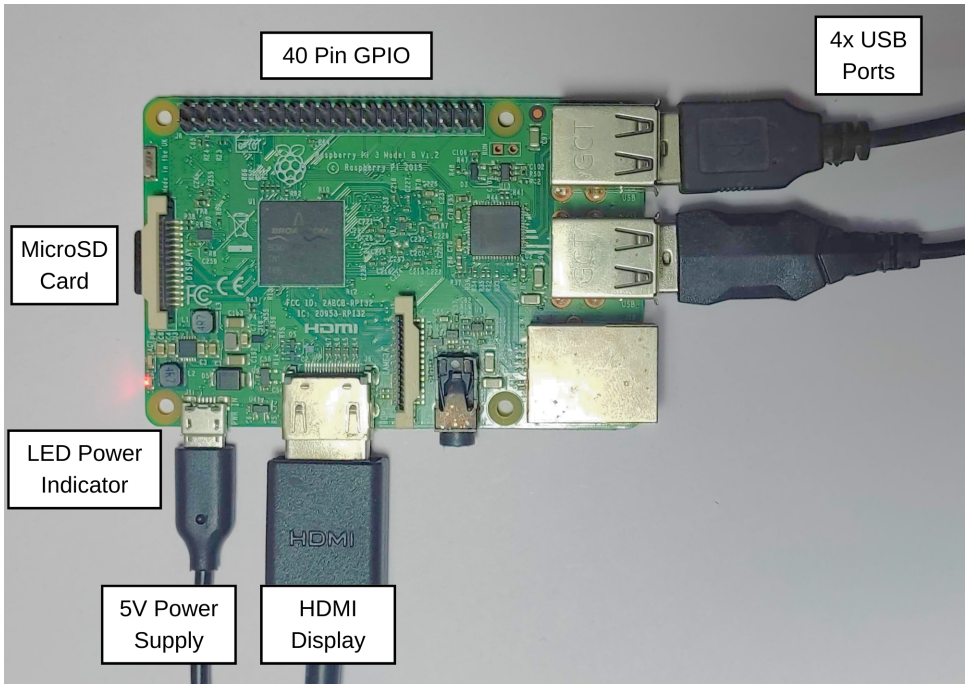


Figure 3.5: **Raspberry Pi**: This figure depicts a Raspberry Pi Model B, a low-cost, single-board computer employed in this research as a micro controller. The image highlights several key components: a microSD card slot for expandable storage, HDMI and USB ports for connecting peripherals, GPIO pins for interfacing with electronic components, and an LED power indicator.

controller. Despite its relatively modest processing power compared to the MacBook Pro, the Raspberry Pi's real-time capabilities are commendable, owing to its optimized design for embedded applications.

Specification	**Raspberry Pi 3 Model B**	**16" MacBook Pro with M1 Pro**
Processor	Quad Core Broadcom BCM2837	Apple M1 Pro
Frequency	1.2 GHz	2.06 - 3.22 GHz
AMU	64 bit	64 bit
RAM	1GB	16GB
Wireless Connectivity	BCM43438 wireless LAN & Bluetooth Low Energy (BLE)	Wi-Fi 6 (802.11ax)
GPIO	40-pin extended GPIO	Not available
Storage	32 GB Micro SD	512GB SSD storage
Power Source	Switched Micro USB power source up to 2.5A	140W USB-C Power Adapter
Weight	42 g	2.1 kg

Table 3.8: Comparison of Raspberry Pi 3 Model B and 16" MacBook Pro with M1 Pro

3.3.2 Overall Performance on Raspberry Pi

The data presented in Table 3.9 offers a comprehensive comparative analysis of the inference performance of various machine learning models across two distinct hardware platforms: Raspberry Pi 3 and the Apple M1 Pro, utilizing a dataset of 1000 data points. The table delineates the inference times, measured in seconds, thereby illustrating the computational efficiency of each model on the aforementioned hardware configurations. Notably, our proposed model demonstrates the capability to detect and isolate faults within a remarkably swift timeframe of 6ms when deployed on an onboard microcontroller. This slight increment in inference time, as observed in comparison to the LSTM classifier, can be attributed to the integration of an RF (Random Forest) classifier within our model's architecture. However, it is imperative to highlight that our model employs a reduced number of trees (L=20) in the RF classifier, a modification made feasible by the sophisticated data processing technique that extracts temporal features utilizing LSTM. This strategic approach not only enhances the model's fault detection capabilities but also optimizes its computational efficiency, making it a viable solution for real-world applications where speed and accuracy are paramount.

Model	Raspberry Pi 3	Apple M1 Pro
Logistic Regression	0.7252	0.0294
SVM	1.3160	0.0504
Random Forest	46.1344	1.9090
MLP Classifier	4.7844	0.5204
LSTM Classifier	5.9172	0.8247
Ours	6.5129	0.8946

Table 3.9: **Inference Time (in seconds)**: Comparative Analysis of Model Performance on Differing Hardware Configurations summed for 1000 Data Points. The table presents the inference time for various machine learning models executed on Raspberry Pi 3 and Apple M1 Pro hardware platforms.

Model	Avg. Time (s)	Min. Time (s)	Max. Time (s)	Std. Dev.
Logistic Regression	0.000725	0.000647	0.065198	0.000673
Support Vector Machine	0.001316	0.001211	0.024925	0.000342
Random Forest	0.046134	0.043446	0.210100	0.003772
MLP Classifier	0.004784	0.002045	0.885560	0.016501
LSTM Classifier	0.005917	0.001837	0.296446	0.006982
Ours	0.006512	0.003810	0.329204	0.004469

Table 3.10: Comparison of Model Performance in Terms of Execution Time over 1000 Data Points.

3.3.3 Statistical Analysis on Inference Time

The table 3.10 provides a detailed statistical analysis of various machine learning models in terms of their execution time metrics, including average time, minimum time, maximum time, and standard deviation. Logistic Regression, Support Vector Machine (SVM), Random Forest, MLP Classifier, LSTM Classifier, and an unspecified "Ours" model are evaluated based on their computational efficiency. Among these models, Logistic Regression exhibits the lowest average execution time of 0.000725 seconds, while the MLP Classifier shows the highest average time of 0.004784 seconds. Interestingly, despite the MLP Classifier's higher average time, it demonstrates the lowest minimum time of 0.002045 seconds, suggesting efficient processing in some instances. On the other hand, the Random Forest model exhibits the highest maximum execution time of 0.210100 seconds, indicating potential variability in its computational complexity. The LSTM Classifier and "Ours" model present interme-

diate performance, with average times of 0.005917 and 0.006512 seconds, respectively. Both models show moderate standard deviations, implying relatively stable performance across different predictions.

The image 3.6 depicts a heatmap that shows prediction time. The heatmap is comprised of a grid of squares, each containing a numerical value. The color intensity of each square corresponds to the value it contains, with warmer colors indicating higher values and cooler colors indicating lower values. The purpose of a prediction time heatmap is to visually represent the amount of time it takes to make a prediction across different predicted labels and actual labels. The x-axis represents the predicted label, and the y-axis represents the actual label. Each square in the heatmap corresponds to a combination of predicted and actual labels, and the color of the square encodes the average prediction time.

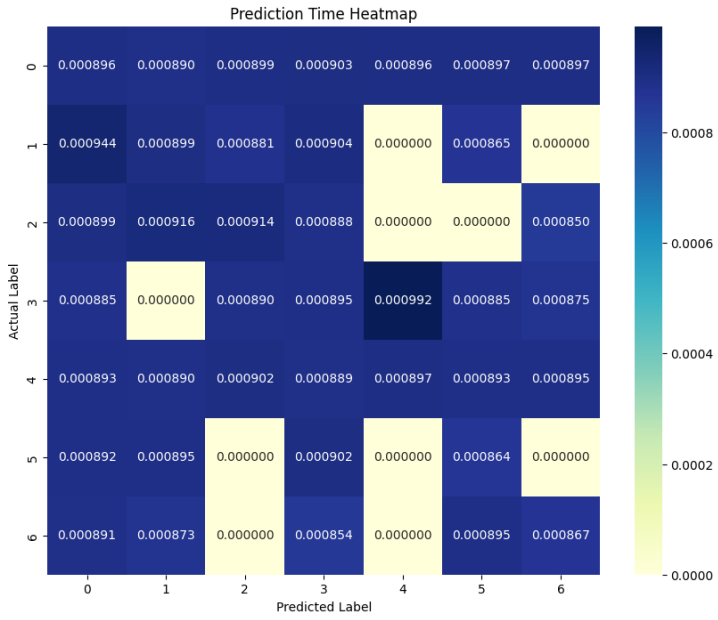


Figure 3.6: This heatmap visualizes the prediction time of our model.

In conclusion, the analysis presented offers valuable insights into the temporal extent required by our fault detection model, delineating its performance across different scenarios. The table comparing median detection steps for simulated and real data highlights the model's efficiency in promptly detecting faults characterized by abrupt changes in values, while also acknowledging the challenges posed by gradual changes in real-world data. Furthermore, the comprehensive statistical analysis of model execution times provides a nuanced understanding of each model's computational efficiency, aiding in informed decision-making regarding model selection for real-world applications.

The subsequent section delves into the performance of our fault detection algorithm on the Raspberry Pi microcontroller, emphasizing its real-time capabilities and efficiency in detecting faults within a constrained computing environment. The comparative analysis between the Raspberry Pi 3 and Apple M1 Pro platforms underscores the adaptability and optimization of our model across diverse hardware configurations.

In summary, our fault detection algorithm demonstrates promising performance across various conditions, showcasing its potential for real-world deployment in scenarios requiring rapid and accurate fault detection. The insights gained from these analyses pave the way for further optimization and refinement, ultimately enhancing the model's effectiveness in practical applications.

Chapter 4

Conclusions

In conclusion, our study introduces an innovative approach to fault detection and localization in hexacopter UAVs, utilizing a synergistic combination of Long Short-Term Memory (LSTM) networks and Random Forest classifiers. This method demonstrates superior performance compared to both classical statistical approaches and other deep learning techniques, through extensive evaluation. Our findings highlight the approach's robustness against varying noise levels, showcasing its applicability in dynamic and unpredictable environments, which is crucial for UAV technology.

Looking ahead, the future of fault detection and localization within hexacopter UAVs is ripe with potential for significant advancements. A key area for further development is enhancing the model's adaptability to dynamic environmental conditions, which are prevalent in real-world applications. Additionally, extending the model to effectively manage multi-motor failures presents a substantial challenge. Given that multiple motors can fail simultaneously under real conditions, developing a more sophisticated fault detection system capable of accurately diagnosing and localizing such complex issues becomes imperative.

Moreover, we are exploring avenues to integrate the fault detection system directly with the UAV's onboard control mechanism to facilitate real-time reconfiguration in the event of detected faults. This integration promises to enable rapid and effective responses to faults, potentially revolutionizing UAV reliability and safety. The goal is to navigate the complexities of different fault modes and conceive advanced fault-tolerant control strategies that surpass the conventional methods of addressing motor failures.

Further research could also delve into anomaly detection, equipping the model to identify unforeseen deviations in UAV behavior. This proactive fault management strategy aims to provide a comprehensive solution capable of swiftly addressing novel challenges, including those outside the scope of pre-defined fault categories. Through these advancements, we envision a future where UAVs operate with unprecedented reliability and safety, significantly expanding their application potential across various domains.

Related Publications

1 **Shivaan Sehgal**, Aakash Maniar, Deepak Gangadharan and Harikumar Kandath, " Leveraging Latent Temporal Features for Robust Fault Detection and Isolation in Hexacopter UAVs" in *IEEE 10th International Conference on Automation, Robotics and Application (ICARA 2024)*.IEEE, 2024 [Accepted]

Bibliography

[1] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[2] L Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001.

[3] PX4 Autopilot Development Team. CUAV V5 Nano - PX4 User Guide. `https://docs.px4.io/main/en/flight_controller/cuav_v5_nano.html`, Accessed: ¡access date¿.

[4] Rodrigo Kuntz Rangel and Alex Coschitz Terra. Development of a surveillance tool using uav's. In *2018 IEEE Aerospace Conference*, pages 1–11, 2018.

[5] Jeongeun Kim, Seungwon Kim, Chanyoung Ju, and Hyoung Son. Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. *IEEE Access*, PP:1–1, 07 2019.

[6] Abd. Manan Samad, Nazrin Kamarulzaman, Muhammad Asyraf Hamdani, Thuaibatul Aslamiah Mastor, and Khairil Afendy Hashim. The potential of unmanned aerial vehicle (uav) for civilian and mapping application. In *2013 IEEE 3rd International Conference on System Engineering and Technology*, pages 313–318, 2013.

[7] Ivan H. Beloev. A review on current and emerging application possibilities for unmanned aerial vehicles. *Acta Technologica Agriculturae*, 19(3):70–76, 2016.

[8] Ayhan Altinors, Ferhat Yol, and Orhan Yaman. A sound based method for fault detection with statistical feature extraction in uav motors. *Applied Acoustics*, 183:108325, 2021.

[9] V. Artale, Angela Ricciardello, and C. Milazzo. Mathematical modeling of hexacopter. *Applied Mathematical Sciences*, 7:4805–4811, 07 2013.

[10] L. Breiman. *Classification and Regression Trees*. Routledge, 1st edition, 1984.

[11] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):832–844, aug 1998.

[12] Jehad Ali, Rehanullah Khan, Nasir Ahmad, and Imran Maqsood. Random forests and decision trees. *International Journal of Computer Science Issues(IJCSI)*, 9, 09 2012.

[13] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 10 2001.

[14] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. 04 1991.

[15] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[17] Inseok Hwang, Sungwan Kim, Youdan Kim, and Chze Eng Seah. A survey of fault detection, isolation, and reconfiguration methods. *IEEE Transactions on Control Systems Technology*, 18(3):636–653, 2010.

[18] Radoslaw Puchalski and Wojciech Giernacki. Uav fault detection methods, state-of-the-art. *Drones*, 2022.

[19] Alessandro Freddi, Sauro Longhi, Andrea Monteriù, and Mario Prist. Actuator fault detection and isolation system for an hexacopter. In *2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA)*, pages 1–6, 2014.

[20] Ngoc Phi Nguyen, Nguyen Xuan Mung, and Sung Kyung Hong. Actuator fault detection and fault-tolerant control for hexacopter. *Sensors*, 19(21), 2019.

[21] Claudio D. Pose, Alessandro Giusti, and Juan I. Giribet. Actuator fault detection in a hexacopter using machine learning. In *2018 Argentine Conference on Automatic Control (AADECA)*, pages 1–6, 2018.

[22] Aditya Mulgundkar, Mayank Singh, Munjaal Bhatt, Prudhvi Raj Turlapati, Deepak Gangadharan, and Harikumar Kandath. Fault detection and isolation on a hexacopter uav using a two-stage classification method. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–6, 2023.

[23] J. J. Tong, W. Zhang, Fangli Liao, C. F. Li, and Y. F. Zhang. Machine learning for uav propeller fault detection based on a hybrid data generation model. *ArXiv*, abs/2302.01556, 2023.

[24] Wansong Liu, Zhu Chen, and Minghui Zheng. An audio-based fault diagnosis method for quadrotors using convolutional neural network and transfer learning. In *2020 American Control Conference (ACC)*, pages 1367–1372, 2020.

[25] Pu Yang, Huilin Geng, Chenwan Wen, and Peng Liu. An intelligent quadrotor fault diagnosis method based on novel deep residual shrinkage network. *Drones*, 5:133, 11 2021.

[26] Xiaojun Yang, Chuan Wan, Tongshuai Zhang, and Zhihua Xiong. Feature extraction of sequence data based on lstm and its application to fault diagnosis of industrial process. In *2022 IEEE 11th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 693–698, 2022.

[27] Jeevesh Vanga, Durga Prabhu Ranimekhala, Swathi Jonnala, Jhansi Jamalapuram, Balaji Gutta, Srinivasa Rao Gampa, and Amarendra Alluri. Fault classification of three phase induction motors using Bi-LSTM networks. *Journal of Electrical Systems and Information Technology*, 10(1):28, December 2023.

[28] Burak Gülmez. Stock price prediction with optimized deep lstm network with artificial rabbits optimization algorithm. *Expert Systems with Applications*, 227:120346, 2023.

[29] Khaled A. Althelaya, El-Sayed M. El-Alfy, and Salahadin Mohammed. Evaluation of bidirectional lstm for short-and long-term stock market prediction. In *2018 9th International Conference on Information and Communication Systems (ICICS)*, pages 151–156, 2018.

[30] Amber C. Kiser, Karen Eilbeck, and Brian T. Bucher. Developing an lstm model to identify surgical site infections using electronic healthcare records. *AMIA Joint Summits on Translational Science proceedings. AMIA Joint Summits on Translational Science*, 2023:330–339, 2023.

[31] G. Maragatham and Shobana Devi. Retraction note: Lstm model for prediction of heart failure in big data. *Journal of Medical Systems*, 46(6):42, 2022.

[32] Yisak Debele, Ha-Young Shi, Assefinew Wondosen, Tae-Wan Ku, and Beom-Soo Kang. Deep learning-based robust actuator fault detection and isolation scheme for highly redundant multirotor uavs. *Drones*, 7(7), 2023.

[33] Vidyasagar Sadhu, Khizar Anjum, and Dario Pompili. On-board deep-learning-based unmanned aerial vehicle fault cause detection and classification via fpgas. *IEEE Transactions on Robotics*, 39(4):3319–3331, 2023.

[34] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53, 12 2020.

[35] Joanne Peng, Kuk Lee, and Gary Ingersoll. An introduction to logistic regression analysis and reporting. *Journal of Educational Research - J EDUC RES*, 96:3–14, 09 2002.

[36] Theodoros Evgeniou and Massimiliano Pontil. Support vector machines: Theory and applications. volume 2049, pages 249–257, 09 2001.

[37] Ludmila Kuncheva and Juan Rodríguez. An experimental study on rotation forest ensembles. volume 4472, pages 459–468, 05 2007.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[39] Abien Fred Agarap. Deep learning using rectified linear units (relu), 2019.

[40] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[41] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 2004.

[42] Ameen Abd Al-salam Selami and Ahmed Fadhil. A study of the effects of gaussian noise on image features. *Kirkuk University Journal / Scientific Studies (1992-0849)*, 11:152 – 169, 04 2016.

[43] RC Hyderabad. Zd850 hexacopter frame, Accessed: 2024.

[44] Andrea Alaimo, Valeria Artale, Cristina Lucia Rosa Milazzo, and Angela Ricciardello. Pid controller applied to hexacopter flight. *Journal of Intelligent & Robotic Systems*, 73(1):261–270, 2014.

[45] Raspberry Pi 3 Model B. `https://www.raspberrypi.com/products/raspberry-pi-3-model-b/`. Accessed: March 9, 2024.