Final Submission Report

Inter IIT Tech Meet 13.0

POSITION AND ALTITUDE CONTROL QUADROTOR WITH SINGLE MOTOR FAILURE

Team 34

December 4, 2024

Contents

1 Problem Understanding:

Multirotor drones have been widely used in many applications, such as inspection, delivery, surveillance, agriculture and entertainment. Among different types of multi-rotor drones, quadrotors are the most popular by virtue of their simple structure and relatively high aerodynamic efficiency. However, due to less rotor redundancy, quadrotors are also more vulnerable to motor failures. The project presented by ideaForge poses a challenging problem in the field of drone control systems. Specifically, it requires the design and implementation of a robust position and altitude control algorithm that can handle a single motor failure condition. In typical multirotor drone configurations, the failure of a single motor can lead to instability and loss of control due to the imbalance in thrust and torque. This project aims to detect motor failure, develop a control solution that compensates for such a failure, allowing the drone to maintain stable flight, altitude, and position under adverse conditions and to implement these cases in the existing PX4 firmware for multicopters.

- **Single Motor Failure:** Detecting single motor failures mid-flight presents a complex challenge, as each type of failure has distinct impacts on the quadrotor's stability and control.

  Causes of Motor Failure and its Effects:

  - **Short Circuit and Excess Current Draw:** A motor can short circuit, causing an excessive current draw that may damage the motor, ESC, or wiring. This can be triggered by external debris or internal faults and can lead to motor stoppage, which compromises stability.

    A short circuit leading to excess current draw can cause the motor to overheat or fail entirely, potentially damaging the ESC or wiring in the process. This failure results in an immediate loss of thrust from the affected rotor, compromising stability. The quadrotor may experience asymmetric thrust, causing it to tilt uncontrollably, enter a yaw spin, or lose altitude.
  - **Loss of Current and Motor Stoppage:** Some failures involve a sudden loss of current, with the motor going offline instantly. This is usually due to power or communication loss, resulting in an immediate halt in motor function and destabilizing the flight.

    Sudden current loss results in an immediate stoppage of the motor. This abrupt failure creates a significant thrust imbalance, destabilizing the drone.
  - **Synchronization Issues:** Motors can fall out of sync without stopping completely, typically due to signal or ESC timing issues. This leads to inconsistent performance and can destabilize the quadrotor by affecting control responsiveness.
  - **Damaged Propellors:** Partial propeller damage can lead to rotor imbalance, causing mechanical vibrations that introduce noise to the IMU. This noise may destabilize the controller, rendering partial fault-tolerant control (FTC) methods ineffective even if the damaged rotor still generates some thrust. In such cases, a better approach is to shut down the damaged rotor and apply FTC designed for a complete rotor failure.
  - **ESC Failures and Disconnection:** Issues originating from the ESCs, such as disconnections or abnormal current draws, can impair motor function. Component wear or wiring faults are common causes, often detectable through ESCs indicators, and lead to unpredictable motor behavior.

- **Impact of Motor Failure on Quadrotor Flight Dynamics:** When a single rotor fails on a quadrotor, it disrupts the balance of lift and torque, leading to immediate instability. This imbalance causes the drone to tilt or rotate uncontrollably toward the side of the failed rotor, making it difficult to maintain altitude or orientation. The failure also disrupts the counter-rotational balance needed for yaw control, often causing rapid spinning or yawing that the control system struggles to correct.

  The remaining three functional rotors are then forced to compensate for the loss by increasing their thrust output. However, this increased workload imposes greater mechanical and thermal stress on the motors, ESCs, and power system. If this compensation persists for an extended duration, it can

lead to overheating, accelerated wear, or secondary failures in the remaining components. In severe cases, this cascading effect can cause a complete loss of control and ultimately result in a crash.

This instability poses particular challenges for the Extended Kalman Filter (EKF), commonly used for state estimation in drone control. The EKF assumes normal operational dynamics, which a rotor failure invalidates. When one rotor stops, the EKF receives conflicting data as expected aerodynamic and inertial responses no longer align with actual sensor inputs. As a result, the filter may produce inaccurate estimates of position, orientation, and velocity, impairing the drone's ability to correct its flight path. This mismatch between the EKF's predictions and reality can exacerbate instability, making it harder for the control system to initiate effective recovery responses and potentially leading to a crash

In addition to these challenges, the dynamic redistribution of thrust can also shift the quadrotor's center of gravity, compounding instability. The control system must manage this sudden shift while operating under reduced maneuverability, often with limited time to react. These dynamics underscore the critical need for fault-tolerant control strategies and robust failure detection mechanisms to mitigate the risks associated with single motor failure. Without these, the quadrotor remains vulnerable to unrecoverable instability following even a single rotor malfunction.

2 Final Approach and Algorithm:

2.1 Core Principle and Logic:

The foundational approach centers on advancing motor failure detection, enhancing control strategies, and fine-tuning EKF parameters to ensure greater stability and reliability in multi-rotor systems.

- **Robust Motor Failure Detection:** Dual-Method Approach for Motor Failure Detection Our motor failure detection system utilized a dual approach to ensure robust monitoring and identification of motor failures.

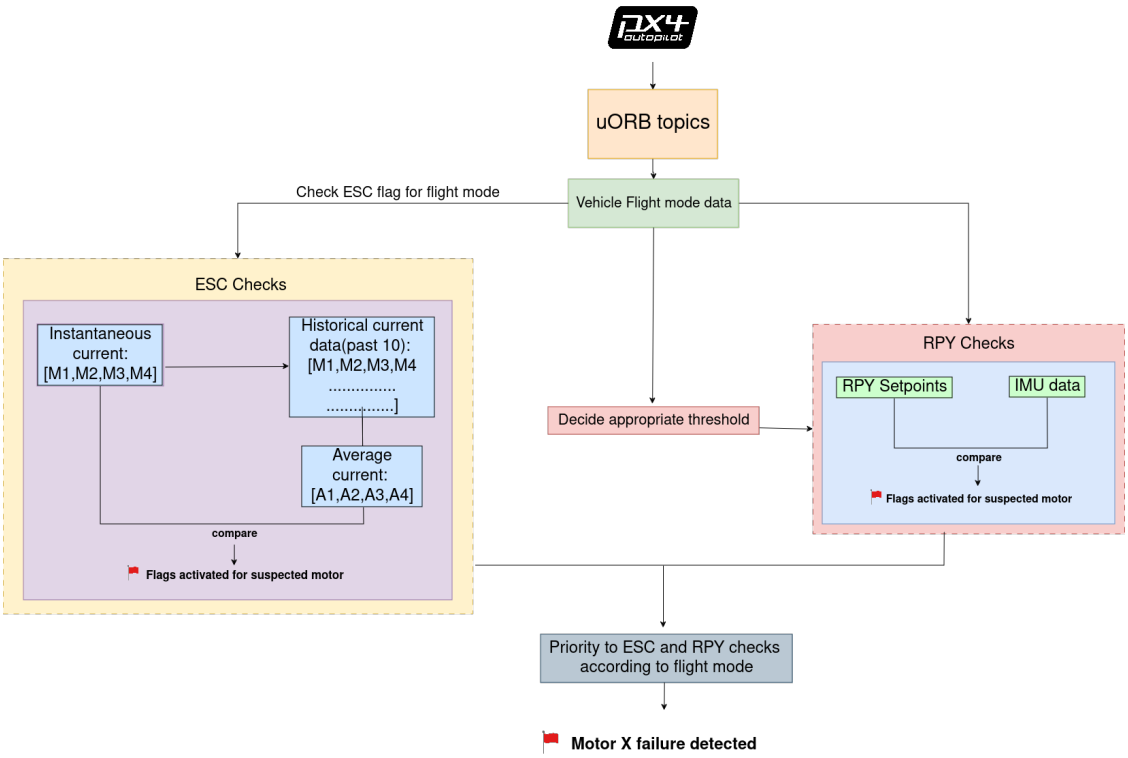


Figure 1: Logic for Motor Failure Detection

- **RPY Error-Based Detection:** This component leverages the fact that a motor failure affects the drone's ability to maintain altitude and orientation, leading to a sudden discrepancy between actual and target RPY values. The module subscribes to the vehicle attitude and vehicle attitude setpoint topics to retrieve the current attitude and the desired setpoint, respectively. By analyzing the error patterns between these values—particularly in autopilot or mission modes where orientation errors are typically minimal—the system can identify which motor has likely failed.
- **Data-Driven ESC Failure Detection:** To address limitations of RPY error-based detection, our second method incorporated ESC data, particularly focusing on current, voltage, and RPM metrics. The PX4 Autopilot system provides an esc status topic that reports ESC-specific failures and flags, helping detect direct ESC malfunctions. Additionally, by tracking the moving average of ESC current, we identify abnormal spikes or drops, further signaling potential motor or ESC issues.

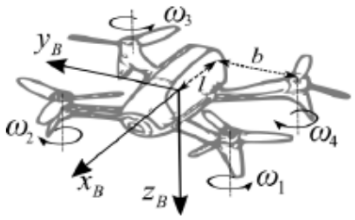


Figure 2: Body-fixed frame definition

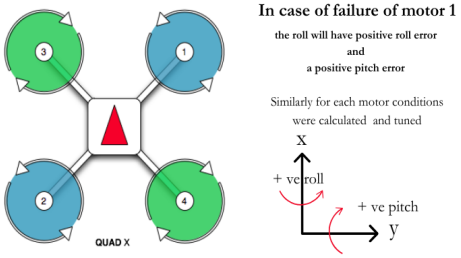


Figure 3: Assumed Quadcopter Configuration

As a complementary metric, we monitored the variance in ESC current over time. A sudden increase in variance suggested instability or inconsistent performance in a specific motor, which could precede a full motor failure. This metric was used in combination with the RPY error and current spike checks to create a multi-layered detection mechanism.
The combination of RPY error thresholds and ESC data allowed the system to localize the failed motor. If a failure was detected, the module used a modified control algorithm to adjust for the missing motor's contribution, helping the quadrotor stabilize or land safely with the remaining functional motors.

Control Logic: The fault-tolerant control logic described in [2] is designed to maintain high-speed flight for a quadrotor despite the complete loss of one rotor. This system employs a three-loop nonlinear controller, which integrates sensor measurements to reduce dependency on precise dynamic models. The outermost loop is a PID-based position controller that uses acceleration as the virtual input to regulate trajectory. This layer compensates for aerodynamic drag and aligns the quadrotor's motion with desired waypoints.The controller design integrates attitude control and rotor speed management into a unified framework tailored for a damaged quadrotor configuration.

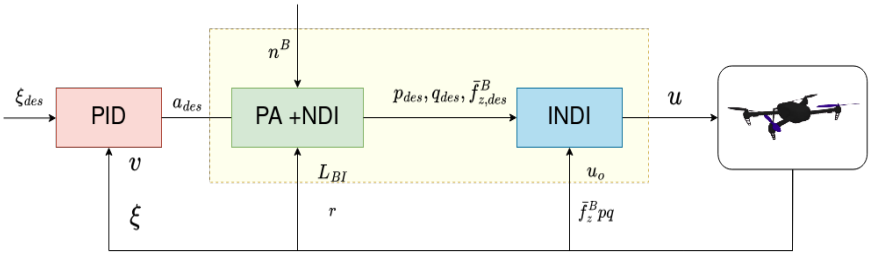


Figure 4: Control structure of the three-loop nonlinear controller.

At the intermediate layer, attitude control is achieved using a Nonlinear Dynamic Inversion (NDI) approach that accounts for the unique dynamics of the damaged configuration. With one rotor missing, the quadrotor naturally spins around a "primary axis." This layer stabilizes angular rates by aligning the quadrotor's thrust vector with the desired specific force, enabling controlled yaw motion and preventing erratic behavior.
The innermost loop uses Incremental Nonlinear Dynamic Inversion (INDI) to translate the attitude control commands into rotor speeds. By incorporating real-time sensor feedback, this method estimates forces and moments directly, compensating for unpredictable aerodynamic effects caused by high-speed flight and spinning. This eliminates the reliance on detailed aerodynamic models, enhancing robustness and stability in extreme conditions.
To implement this controller, the equations of the NDI and INDI approaches are combined into a single block, relying on external angular velocity readings. This unified design ensures seamless coordination between attitude stabilization and rotor speed control, achieving reliable performance even in challenging scenarios.

Optimization of ECL EKF Performance: The PX4 Iris drone's ECL EKF (Extended Kalman Filter) used a time delay approach to handle sensor lag, aligning sensor measurements with matching timestamps. It employed a complementary filter to bring sensor data up to the current time, smoothing sudden changes and improving stability. To evaluate the accuracy of the ECL EKF, we conducted tests and visualized position and orientation data. This filtered data was compared with ground truth values obtained from simulation to assess performance.
For accurate results, we carefully adjusted the EKF parameters to optimize sensor fusion, delay handling, and noise filtering. By fine-tuning these parameters, we minimized errors and enhanced the EKF's performance across various flight scenarios. Additional adjustments were made to improve overall accuracy, ensuring the EKF adapted effectively to all flight conditions.

2.2 Control Equations:

Based on the methodology outlined in [2], we implemented a fault-tolerant control strategy to handle scenarios involving a single rotor failure. This section details the mathematical equations applied in designing this controller.The equations of motion [1] of a quadrotor based on six-dimensional rigid body dynamics are given as:

$$\dot{\xi} = v \tag{1}$$

$$m\dot{v} = mg + L_I^B F \tag{2}$$

$$\dot{L}I^B = LI^B\,\tilde{\Omega} \tag{3}$$

$$I_v\dot{\Omega} = -\tilde{\Omega}I_v\Omega + M. \tag{4}$$

In this formulation, the superscripts B and I indicate the coordinate system on which the vector is projected. $\mathcal{F}_I$, is defined as the reference coordinate system fixed to the ground. $\mathcal{F}_B$, is fixed to the quadrotor with x_B pointing forward, y_B pointing right, and z_B opposite to the thrust vector. Subscripts x, y, and z denote the components of a three-dimensional vector.

ξ and v indicate the position and velocity vector of the quadrotor, respectively.$\Omega = \begin{bmatrix} p & q & r \end{bmatrix}^T$ represents the angular rate of the quadrotor, where p, q, and r are the components of the angular velocity.The tilde superscript denotes the cross product operation, i.e., $\tilde{\Omega}I_v = \Omega \times I_v$. m and I_v represent the mass and the inertia matrix of the vehicle, respectively. g is the gravity vector. F and M represent the resultant force and moment vector acting on the quadrotor, which include the control force and moment generated by the rotors. L_I^B is the rotation matrix that transforms coordinates from the body frame $\mathcal{F}_B$ to the inertial frame $\mathcal{F}_I$

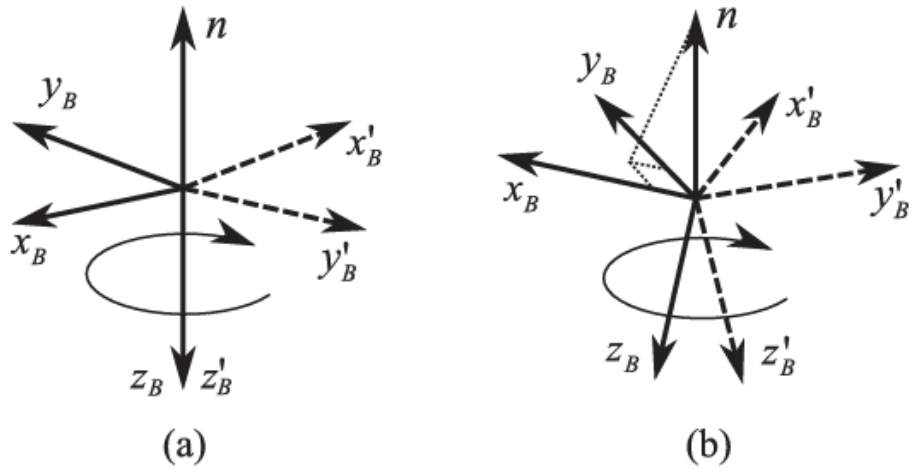


Figure 5:
a) n alligned with instant thrust direction(Less energy efficient but prevents wobbling)
b) n not parallel with instant thrust direction(more energy efficient but more wobbling)

$$L_I^B = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \tag{5}$$

This matrix L_I^B uses the ZYX Euler angle convention, which applies the rotations in the order of yaw (ψ), pitch (θ), and roll (ϕ). Let us assume the right back rotor (ω_4) fails and define the control input $\mathbf{u}$ as:

$$\mathbf{u}^T = \begin{bmatrix} \omega_1^2 & \omega_2^2 & \omega_3^2 \end{bmatrix}$$

A general expression for the resultant force and moment can be written as :

$$\mathbf{F} = \mathbf{F}_a + G_F\mathbf{u}$$

$$\mathbf{M} = \mathbf{M}_a + \mathbf{M}_g + G_M\mathbf{u} \approx \mathbf{M}_a + G_M\mathbf{u}$$

where $\mathbf{M}_g$ is the rotor-induced gyroscopic moment, which is found to be negligible. $\mathbf{F}_a$ and $\mathbf{M}_a$ represent the external aerodynamic force and moment, which are influenced by the airspeed, aerodynamic angles, angular rates, and complex interaction effects that are difficult to model accurately.

Position Control Loop: The outer loop is a PID controller using acceleration as a virtual input: The desired acceleration $\mathbf{a}_{des}$ is computed using a PID controller:

$$\mathbf{a}_{des} = k_p(\xi_{des} - \xi) + k_d(\dot{\xi}_{des} - v) + k_i \int (\xi_{des} - \xi)\, dt, \quad (k_p, k_d, k_i > 0) \tag{6}$$

Integral term is used compensate the constant bias brought by aerodynamic drag.

Attitude Control Loop (PA + NDI Block): After the removal of a single rotor, the quadrotor subsequently spins around a certain axis due to the fact that the yawing moment balance is broken . The primary axis, denoted by n, and defined as a unit vector about which the damaged quadrotor rotates and points at the average thrust direction in the relaxed hover solution. This vector is fixed to the body frame $\mathcal{F}_B$ and can be chosen arbitrarily. A more tilted primary axis leads to a larger wobbling angle and a smaller spinning rate. To this end, a normalized vector pointing in the direction of the desired specific force is defined as $\mathbf{n}_{des}$ and is calculated by:

$$\mathbf{n}_{des} = \frac{\mathbf{a}_{des} - \mathbf{g}}{\|\mathbf{a}_{des} - \mathbf{g}\|} \tag{7}$$

The unit vector $\mathbf{n}_{des}^B$ is defined as:

$$\mathbf{n}_{des}^{B} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

The primary axis reference vector $\mathbf{n}^B$ is given by:

$$\mathbf{n}^{B} = \begin{bmatrix} n_x^B \\ n_y^B \\ n_z^B \end{bmatrix}$$

The norm of the unit vector is defined as:

$$\sqrt{h_1^2 + h_2^2 + h_3^2} = 1 \tag{8}$$

Thus, accurate tracking of h_1 and h_2 to their reference values (the first two components of $\mathbf{n}^B$) will also ensure the tracking of h_3. Nonlinear Dynamic Inversion equation is given by:

$$\mathbf{n}_{des}^{B} = \tilde{\mathbf{n}}_{des}^{B}\,\Omega + LB^{I}\,\dot{\mathbf{n}}_{des}^{I}$$

Expanding this, we have:

$$\begin{bmatrix} p_{des} \\ q_{des} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{h_3} \\ -\frac{1}{h_3} & 0 \end{bmatrix} \left(\nu_{out} - \begin{bmatrix} h_2 \\ -h_1 \end{bmatrix} r - \hat{\dot{\mathbf{n}}}_{des}^{I} \right) \tag{9}$$

where

$$\nu_{out} = \begin{bmatrix} \dot{n}_x^B + k_x(n_x^B - h_1) \\ \dot{n}_y^B + k_y(n_y^B - h_2) \end{bmatrix}, \quad (k_x, k_y > 0)$$

The magnitude of the desired acceleration should be determined by thrust, which is computed using

$$n_z^B\, \bar{f}z, des^B\, \mathbf{n}des = \mathbf{a}_{des} - \mathbf{g} \tag{10}$$

where $\bar{f}z, des^B = -\frac{Tdes}{m}$ indicates the desired specific force along the z-axis in the body frame B.
INDI Block: The desired rotor speed command using INDI controller can be obtained by:

$$\mathbf{u}_{INDI} = \hat{B}^{+}(x_f)(\nu_{in} - y_f) + \mathbf{u}_f \tag{11}$$

where

$$\nu_{in} = \begin{bmatrix} \dot{p}_{des} + k_1(p_{des} - p) \\ \dot{q}_{des} + k_2(q_{des} - q) \\ \bar{f}_{z,des}^{B} + k_3 \int (\bar{f}_{z,des}^{B} - \bar{f}_{z}^{B})\, dt \end{bmatrix}, \quad (k_1, k_2, k_3 > 0)$$

and

$$\ddot{\mathbf{y}}_f = \begin{bmatrix} \frac{f}{R_{33,f}} \\ \ddot{h}_1 \\ \ddot{h}_2 \end{bmatrix}$$

The control effectiveness matrix [3] $\hat{B}(x_f)$ can be estimated as .

$$\hat{B}(x_f) = \begin{bmatrix} -\overline{\kappa} R_{33,f}/mv & -\overline{\kappa} R_{33,f}/mv & -\overline{\kappa} R_{33,f}/mv \\ -\overline{\kappa} b \sin\beta & \overline{\kappa} b \sin\beta & \overline{\kappa} b \sin\beta \\ \overline{\kappa} b \cos\beta & \overline{\kappa} b \cos\beta & -\overline{\kappa} b \cos\beta \end{bmatrix}$$

2.3 Implementation of Control Algorithm:

2.3.1 Verification of Controller using POC:

The verification of the fault-tolerant control system was conducted using a combination of theoretical formulations and simulations, as outlined in the referenced paper [2]. The implementation utilized the Incremental Nonlinear Dynamic Inversion (INDI) controller described in the paper, which is particularly robust against aerodynamic disturbances and model uncertainties. The equations of the controller, as mentioned earlier in this report, were crucial for its implementation.
The verification began with simulations conducted in the Gazebo environment using the Iris drone, facilitated by the `rotors_simulator` package. This package, part of the broader `rotors` project within ROS, offers a comprehensive framework for testing and developing quadrotor control algorithms. It includes various components such as `rotors_description`, `rotors_control`, `rotors_gazebo_plugins`, and others that enable seamless integration with the Gazebo simulation environment.
The modular nature of `rotors_simulator` simplified the process of creating and modifying test scenarios, supporting proof-of-concept development and validation of innovative control strategies like the Incremental Nonlinear Dynamic Inversion (INDI) controller.
By using `rotors_simulator`, we evaluated the quadrotor's response to scenarios such as a simulated motor failure and verified the controller's seamless switch to failure mode. The ability to adjust and fine-tune control parameters during these tests demonstrated the robustness of the control system, providing confidence in its performance under various conditions.
These simulations demonstrated the quadrotor's ability to maintain stable hover, perform controlled maneuvers, and achieve return-to-land (RTL) functionality, even in scenarios where a single rotor was completely inoperative. Tracking of the desired position, attitude, and yaw rate was verified against reference trajectories with minimal error, showcasing the robustness of the control system.

Key Observations:

- The quadrotor's exceptional stability during hovering.
- Its ability to perform controlled descents and landings.
- The precision of the RTL functionality, which followed pre-defined paths.

These results confirm the feasibility of the fault-tolerant control system under simulated real-world conditions. The proof of concept verifies the feasibility and reliability of the implemented fault-tolerant control system for autonomous quadrotor operations, as validated through simulations in Gazebo.

2.3.2 Implementation in PX4 stack:

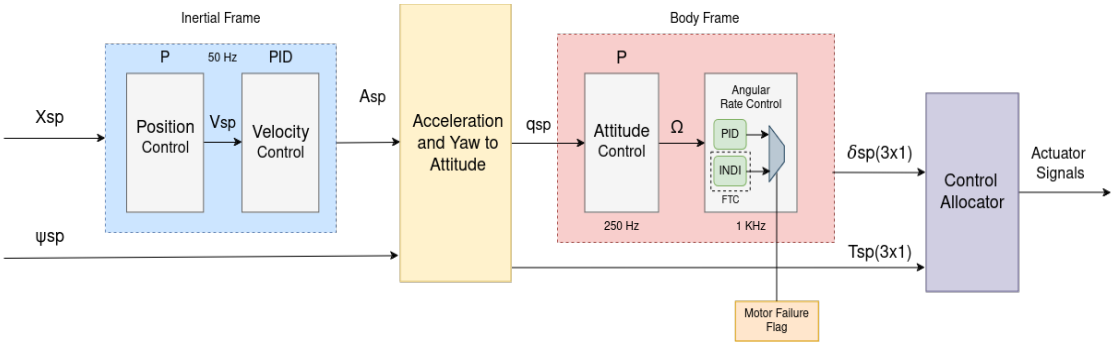


Figure 6: Modified Control Architecture in PX4

The primary goal of this project was to enhance PX4's motor failure detection and response mechanisms by implementing a custom detection method, publishing failure states via the uORB messaging system, and integrating these changes with PX4's Control Allocator. The modifications aim to achieve faster and

more accurate identification of motor failures, coupled with an adaptive response that maintains multicopter stability. These improvements address limitations in the existing system, such as delayed detection and failure to trigger failsafe responses under certain conditions.
The above figure illustrates the modified architecture designed to integrate our verified Fault-Tolerant Control (FTC) algorithm into the PX4 firmware, ensuring it is both robust and deployable. PX4's control system is inherently structured around two cascaded control loops, which were adapted to accommodate our approach.
The outer loop is tasked with managing position and velocity control, employing P and PID controllers to ensure precise stabilization and trajectory tracking. Operating at a consistent frequency of 50 Hz, this loop performs all intermediate computations within the inertial frame (North-East-Down, NED). Its primary output is an acceleration setpoint, which is subsequently transformed into thrust vectors through a defined conversion process.

$$T = \frac{a_{sp} \cdot T_h}{g} - T_h \qquad \begin{array}{l}\text{where,}\\ \text{a}_{sp} \text{ is the acceleration setpoint,}\\ T_h \text{ is the estimated hover thrust,}\\ g \text{ is the gravitational constant.}\end{array} \tag{12}$$

This structured approach enables seamless coordination between position dynamics and downstream control systems. The thrust vectors generated by the outer loop serve as critical inputs for the subsequent stages of the control process. These thrust values are used to derive the attitude setpoint for the inner loop controller, which operates at a significantly higher frequency than the outer loop, as depicted in the block diagram.
The inner loop is responsible for converting the desired attitude of the drone into corresponding moment/torque setpoints. This process involves a cascaded control structure, comprising two key components:

- **A nonlinear attitude controller:** which determines the required attitude based on the thrust setpoint.
- **PID rate controller:** which refines the angular rates setpoint into precise moment setpoints.

The outputs of the rate controller, namely the moment setpoints, are combined with the thrust setpoint to serve as control inputs for the control allocation block.
A series of enhancements were implemented to integrate the controller seamlessly into the system and meet its specific requirements:

- **Implementation of EKF2_MOT_F parameter in Motor Failure Detection:** On top of the existing custom motor failure detection module , a new parameter,'`EKF2_MOT_F`', was introduced to track motor failure states. The parameter represents motor failure conditions with values ranging from 0 (no failure) to 4 (specific motor failures). It was defined in the '`module.yaml`' file under the ekf2 module directory, declared in EKF2.hpp, and dynamically updated by the custom motor failure detection module. This parameter forms the core link between detection logic and the broader PX4 system.
- **Middleware Modifications for Controller Inputs:** Adjustments were made to the middleware to meet the controller's requirements. A custom module, '`accel_indi`', was created to act as an intermediary. This module subscribes to essential data and publishes it to a custom topic named '`estimatedpa`'. This topic provides critical information about the desired principal axis in both the inertial and body frames, calculated based on the desired acceleration setpoint generated by the outer loop controller (position control).
- **Custom Functions for PA + NDI and INDI Blocks:** Custom functions were implemented to handle the mathematical computations for the combined **Principal Axis (PA) + NDI block** and the **INDI block**. These functions were integrated into the '`mc_rate_control`' module. Since this block operates at the highest system frequency, it is directly responsible for computing and publishing the torque and thrust setpoints.

- **Enhancements in the existing motor failure Failsafes in Control Allocator :** The Control Allocator in PX4, which handles actuator command distribution, was enhanced to respond adaptively to motor failures. The built-in failsafe mechanism was modified to integrate seamlessly with the custom detection logic. The Control Allocator in PX4 has been enhanced to dynamically handle motor failures using updates from the '`failure_detector_status`' uORB topic and by referencing the '`EKF2_MOT_F`' parameter as an additional check. These improvements ensure accurate exclusion of failed motors from the control allocation process, maintaining system stability and flight safety.

2.3.3 Flow of control algorithm:

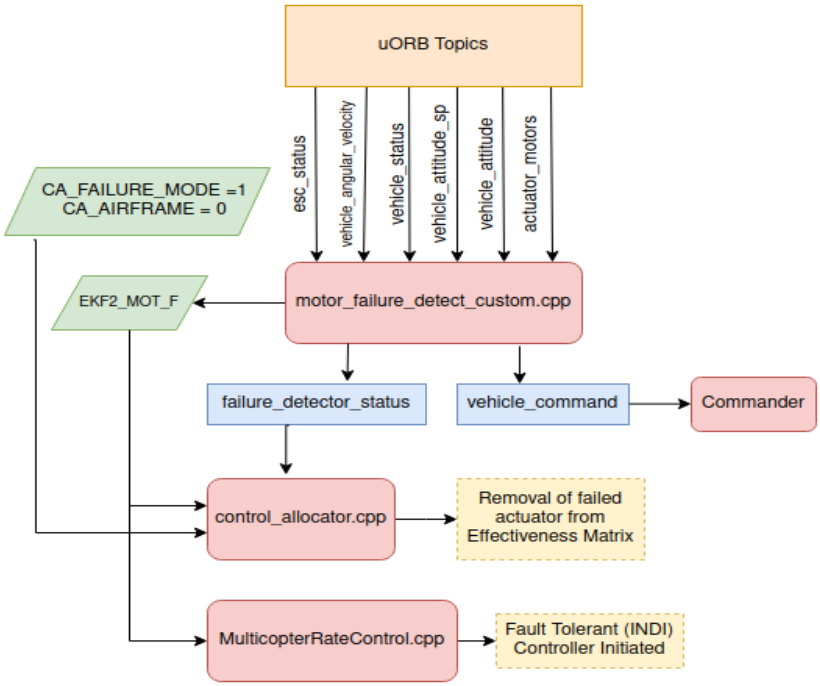


Figure 7: Flow of Control Algorithm

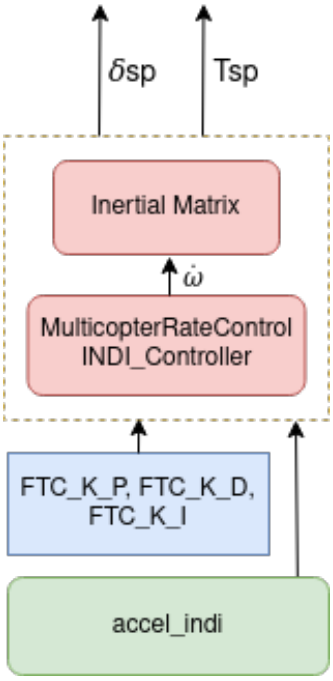


Figure 8: Control Output states

The flowchart above illustrates the current process of motor failure detection integrated with the fault-tolerant control system.

- The motor failure detection algorithm utilizes a dual-method approach for identifying motor failures, as described earlier, and is implemented within our custom module. On identification of a motor failure, the system implements a robust mechanism for publishing motor failure information using uORB topics, ensuring accurate and timely updates across the PX4 framework. Detailed failure data is published to the failure_detector_status topic, which includes critical fields such as fd_motor (set to true to indicate a detected motor failure) and motor_failure_mask (a bitmask encoding the specific motor or motors that have failed).

- The EKF2_MOT_F parameter is dynamically updated during the detection process. This parameter serves as the basis for publishing failure states. The detection logic reads the current value of EKF2_MOT_F and maps it to the corresponding motor_failure_mask in the failure_detector_status topic, ensuring that detailed and actionable information about the failure is shared across relevant modules. This seamless integration enables the control allocator and other subsystems to respond promptly to motor failures, improving overall system robustness.

- When a motor failure is detected, indicated by fd_motor = true in the failure_detector_status topic, the allocator removes the corresponding row of the failed motor from the effectiveness matrix, effectively excluding it from control allocation. The matrix is then updated to adapt to the new configuration.

The allocator identifies the failed motor(s) primarily through the motor_failure_mask bitmask provided in the failure_detector_status topic. The EKF2_MOT_F parameter serves as an additional reference, offering a clear indication of the failure state for logging or monitoring purposes. This redundancy ensures robustness in detecting and managing motor failures.

To avoid unnecessary reconfigurations, the allocator processes only the first detected failure, recording its state in the _handled_motor_failure_bitmask. If no further failures are detected, the system retains the modified matrix configuration. In recovery scenarios where motor failures are resolved, the allocator resets the bitmask and restores all motors to the effectiveness matrix, ensuring seamless transitions between failure and recovery modes.

- Figures (7) and (8) illustrate how the `EKF2_MOT_F` parameter is accessed within the `mc_rate_control` module. A small multiplexer(MUX) determines which outputs to consider based on the parameter's value. Upon detecting a failure, the MUX directs the output to the INDI control block, which estimates the necessary torque setpoints to stabilize the drone. This process is iteratively executed at a frequency of 1 kHz using the provided codebase.

  Additionally, the drone's behavior is selected based on the configured failsafe action, such as *Land*, *Return to Launch (RTL)*, or *Position Hold*. This command is sent to the commander module via the `vehicle_command` topic, ensuring an appropriate response to the detected failure.

2.4 Enhancements Post Midterm:

After the mid-evaluation submission, significant improvements were made to enhance the robustness and reliability of the motor failure detection module. These improvements encompassed modifications to the middleware, the introduction of new parameters, custom tuning methods, and enhancements to the Control Allocator to ensure efficient handling of motor failures.

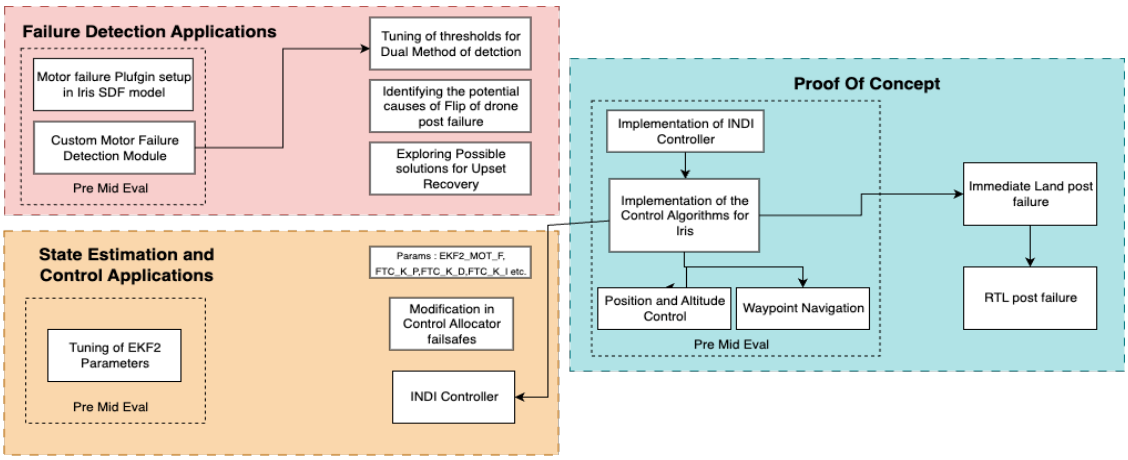


Figure 9: Complete Workflow

- **Motor Failure Detection:** The motor detection algorithm was refined to trigger flags when either the Roll-Pitch-Yaw (RPY) error or ESC error data exceeded flight mode-specific thresholds. A statistical approach was used to identify optimal thresholds that dynamically adapt according to the flight mode, leading to improved detection performance in Manual and Acro Modes, alongside an impressive 95%(refer to Figure 13) accuracy in Position Mode.Our dynamic thresholds, which adapt based on flight mode, achieve 74% accuracy in manual mode and demonstrate strong performance in acro mode.

  Note: While smaller thresholds improved accuracy, they also increased false detections, demonstrating the system's robustness under various conditions.

- **Parameter Tuning Method:** A custom tuning method was developed to optimize the motor failure detection constants (`FTC_K_P`, `FTC_K_I`, `FTC_K_D`). This approach analyzed actuator signals of diagonally

opposite motors, which exhibit predictable changes during a motor failure. By comparing the difference between these signals to a predefined threshold, the system could fine-tune the constants for detecting failures effectively. This method was crucial during the initial tuning phase to quickly adjust constants and prevent drone flips due to delayed responses. However, its use was limited to stable testing scenarios.

- **Implementation in PX4:** Enhancements were made to improve motor failure detection and control. A custom `accel_indi` module was created to handle data subscriptions and publish principal axis information. Custom functions for PA + NDI and INDI blocks were added to the `mc_rate_control` module for real-time computation of torque and thrust setpoints. A new parameter, `EKF2_MOT_F` (0: no failure, 4: specific failures), was introduced and updated by the detection module. Upon motor failure, the system published failure data via the `failure_detector_status` topic using uORB. The Control Allocator was modified to dynamically adjust by excluding the failed motor and recalibrating the configuration. The motor failure was detected using the `motor_failure_mask` and verified with `EKF2_MOT_F`, ensuring reliable detection and management.

3 Performance Analysis and Simulation Setup:

3.1 Simulation Setup:

Simulation for PX4 and POC:
For the development and testing of the PX4-based motor failure detection and control system, we utilized a robust and integrated simulation setup. The setup comprised the following key software components and their respective versions:

- **Operating System**: Ubuntu 20.04 LTS
- **PX4 Autopilot**: Recent stable version of PX4 (PX4 v1.13.x or higher)
- **Gazebo Simulator**: Gazebo 11 for accurate and realistic simulation of robotic environments and dynamics
- **Robot Operating System (ROS)**: ROS 1 (Noetic) for interfacing and communication between the PX4 autopilot and external components

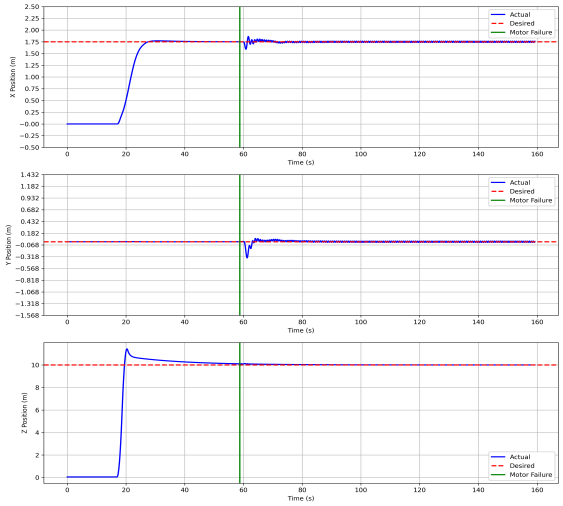
Proof-of-Concept (POC) Dependencies: To facilitate the verification process, the following dependencies and tools were necessary for the POC setup:

- **rotors_simulator Package**: Includes `rotors_description`, `rotors_control`, `rotors_gazebo_plugins`, and other modules for testing and simulation.

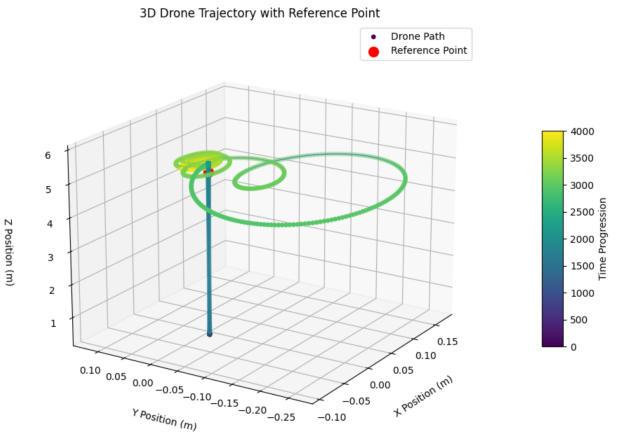
Motor Failure Injection: Motor failure in the simulation was implemented using the Gazebo motor failure plugin, activated via a `rostopic publish` command specifying the motor index. Upon activation, the motor stops functioning, simulating a sudden failure scenario. Initially, the controller compensates with a current spike to the failed motor, followed by the PX4 system's failsafe mechanism halting current supply to prevent further damage. Although PX4 offers a built-in `failure motor off` command for injecting failures, its functionality was found to be limited during simulation testing.

3.2 Performance Metrics:

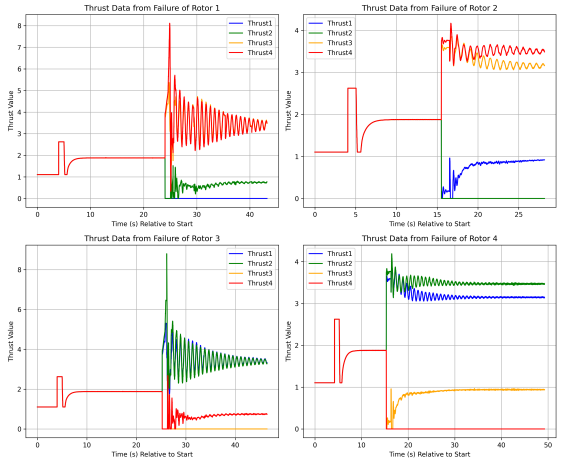
The following performance metrics were observed and analyzed during the development and testing phases of the Proof of Concept (POC). These results were obtained by implementing and evaluating the approaches described in the preceding sections. The metrics provide a comprehensive assessment of the system's capabilities and highlight its overall performance. The data collected reflects the effectiveness of the proposed algorithms and design modifications in achieving the desired outcomes.



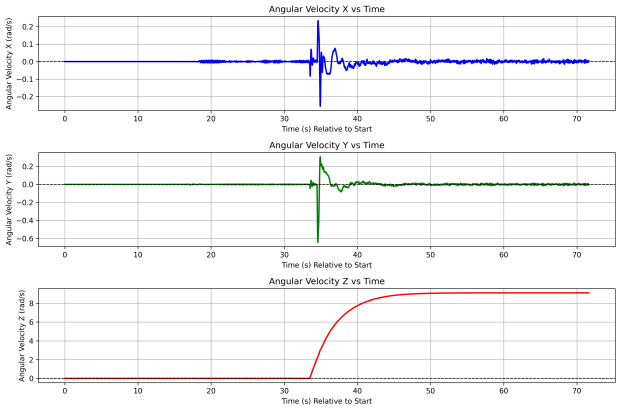
(a) Drifts in X, Y, and Z axes pre and post failure of rotor



(b) Trajectory of Quadrotor



(c) Thrust responses of the remaining three motors



(d) Resultant Angular Velocities pre and post failure of rotor

Figure 10: Performance Metrics 1

3.2.1 Drifts in X, Y, and Z Axes Pre and Post Motor Failure:

The drift matrices of the drone reveal that when a motor fails, an initial sudden, small positional change occurs due to imbalance. The controller then adjusts its output to stabilize the drone and maintain the desired hovering height, demonstrating its effectiveness in handling disturbances post-failure. From Figure 10a, it can be observed that:

The following behaviors were observed in the drift matrices:

- **X-axis**: The maximum observed drift is 0.15 m, indicating a moderate shift in the horizontal plane. Over time, this drift decreases and tends toward zero as the controller compensates.
- **Y-axis**: The maximum drift is more significant, at 0.3 m, showing a larger deviation in the horizontal plane compared to the X-axis. Similar to the X-axis, the drift decreases over time and approaches zero as stability is restored.
- **Z-axis**: No drift is observed, indicating that vertical stability is maintained, and the controller effectively compensates to maintain the desired altitude. This stability persists over time.

Additionally, as shown in Figure 10b, the 3D position trajectory of the drone illustrates the accuracy and precision of the controller in maintaining its desired path even after the motor failure.

3.2.2 Impact of Motor Failure on Actuator Signals And Angular Velocities:

From Figure 10c, it can be observed that when motors 1 and 3 fail (front motors), the system experiences greater disturbance and takes longer time to stabilize the controller compared to when motors 2 and 4 (rear motors) fail.

The following factors contribute to greater disturbance when specific motors fail:

1. **Thrust Vectoring and Lever Arm Effect:** Front motors, positioned farther from the center, generate significant torque for stabilization. Their failure leads to larger disturbances.
2. **Center of Gravity (CG) and Motor Positioning:** Motors closer to the CG or bearing more weight induce greater imbalance upon failure. Front motors are often more critical in this regard.
3. **Aerodynamic Drag and Airflow:** Front motors experience less disturbed airflow compared to rear motors. Their failure causes more instability, as they typically provide more stable thrust.

Following the failure, the quadcopter begins to spin in the opposite direction of the failed motor. The angular velocity plot Figure 10d confirm this behavior, showing that the quadcopter starts with a slow spin, which gradually accelerates and eventually reaches a saturated value of approximately 9-10 rad/s.

3.2.3 Return-to-Land (RTL) Comparison:

Figure 11 compares the RTL matrices before and after motor failure, showing minimal differences between them. This indicates that the controller maintains high accuracy and reliability in executing safe landings even after motor failure. The slight deviations suggest that while there is some impact from the failure, the overall RTL functionality remains effective, ensuring a stable and safe return and landing. This demonstrates the controller's robustness in managing disturbances and sustaining precise autonomous operations in failure scenarios.

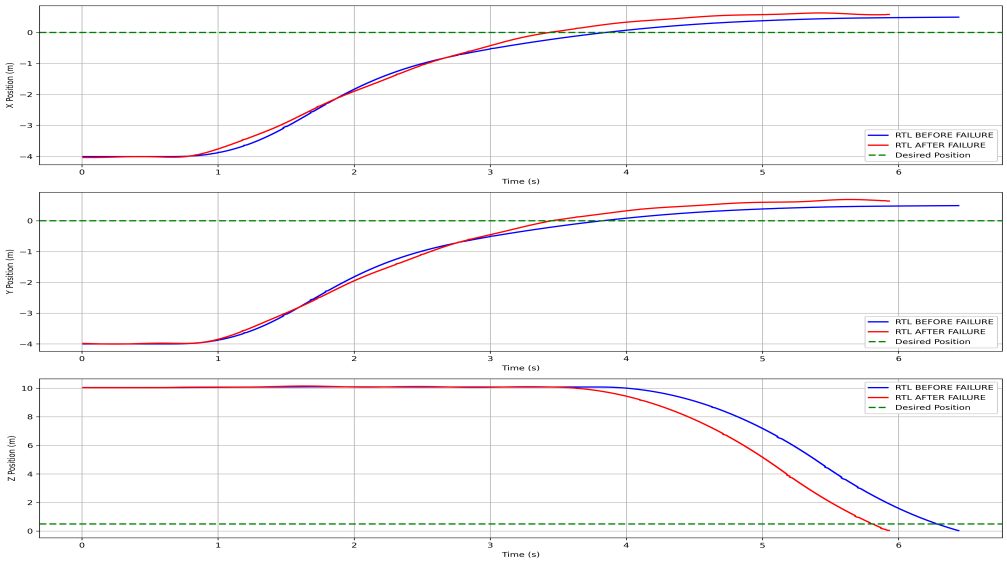


Figure 11: RTL Comparison between failed rotor condition and nominal condition

3.2.4 Stability and Responsiveness:

The system's responsiveness to the failure is relatively fast, especially in the Z-axis. This suggests a well-tuned altitude controller, but slight delays in the X and Y axes indicate lateral control challenges. The oscillatory response indicates marginally underdamped behavior. The lateral axes (X and Y) exhibit overshoot and oscillatory behavior post-failure. This indicates that while the control system is responsive.

4 Challenges and Future Work:

4.1 Challenges and Lessons Learned:

1. **Integration of Motor Failure Detection with Control Allocator:**
   Initially, the control allocator failed to detect the injected motor failure because the custom motor failure detection module was not publishing the required motor failure message to the `uORB` topic `FailureDetectorStatus`. This prevented the control allocator from recognizing and excluding the failed motor from the allocation process. After diagnosing the issue, we updated the motor failure detection module to ensure the correct message was published to the `uORB` topic. With this update, the control allocator successfully identified the motor failure and dynamically removed the non-functional motor from the thrust allocation process.

2. **Tuning of Params for INDI Controller and Unreliable Acceleration Setpoint Data:**
   The custom parameters required for the INDI controller need further tuning to achieve stable hover and position hold conditions within the PX4 implementation. A significant issue observed was the unreliable acceleration setpoint data generated by the outer control loop. This inconsistency may stem from the limiting conditions defined in the `PositionControl` file, where fixed acceleration values are assigned under specific scenarios, potentially leading to suboptimal performance.

3. **Thrust Deficiency During Motor Failure:**
   During the implementation of the fault-tolerant control logic, it was observed that the thrust outputs from the remaining three functional motors were insufficient to stabilize and control the drone after a motor failure. This deficiency primarily stemmed from the high mass and moment of inertia of the drone, which exceeded the thrust-generating capacity of the surviving motors. To address this limitation, we reduced the drone's mass and inertia by approximately 50% in simulation. This adjustment ensured that the remaining motors could generate adequate thrust to maintain stability and control under the new flight dynamics, effectively compensating for the loss of one motor. Thus even in practical situations the thrust to weight ratio needs to be considered before implementation of any motor failure recovery algorithm.

4. **Dealing with existing failsafes while development:**
   When modifying the PX4 codebase, it is crucial to prioritize failsafe mechanisms to ensure system safety under all conditions. Failsafes are essential for managing unexpected events like sensor failures, communication disruptions, or motor malfunctions. During the implementation of custom modules and control algorithms, special attention was given to handling such scenarios effectively. Extensive debugging was conducted to verify functionality, using numerous PX4_INFO and PX4_WARN statements throughout various loops. These debug statements provided clear insights into which conditions were triggered under specific scenarios, aiding in fine-tuning and ensuring robustness in the modified codebase.

5. **Dealing with existing failsafes while development:**
   When making changes to the controller files in PX4, extra care is essential since these modules operate at specific frequencies. The controllers must only activate and publish values when the relevant data in the subscribed topics is being updated. Failure to ensure this can lead to outdated or inconsistent data being used.

6. **Computational Challenges with Upset Recovery and NMPC:**
   Delays in motor failure detection can result in unpredictable drone behavior before the controller can take corrective action. Addressing such scenarios is a critical challenge to ensure stability and prevent further system degradation during the transition period.Implementing Upset Recovery and Nonlinear Model Predictive Control (NMPC) posed significant computational challenges due to the resource-intensive nature of quadratic programming libraries. These libraries rely on numerous dependencies and demand high computational power, which is not always feasible for resource-constrained hardware like some Pixhawk Cube variants.

4.2 Future Scope:

- **Seperate Library for INDI Controller Implementation:** To enhance modularity and scalability, all mathematical computations for the INDI controller can be encapsulated in a separate library, similar to the existing `rate_control` module in PX4. This separation ensures that the controller's core logic is cleanly abstracted, simplifying maintenance and further development. By isolating the computations, the library can be independently tested, tuned, and optimized for better performance. Such a design will also facilitate the development of an end-to-end deployable software solution, ensuring that future updates or modifications can be seamlessly integrated without affecting other system components.

- **Upset Recovery:**

  Upset recovery techniques address delayed motor failure detection by minimizing the error between actual and desired roll/pitch angles when the deviation becomes significant. The approach uses the following key equations:

  – Scaling Factor for Recovery:

$$\epsilon = \max\left(\sqrt{\frac{a_{x,\text{des},0}^2 + a_{y,\text{des},0}^2}{a_{z,\text{des},0}^2}}\tan\theta_1, 1\right) \tag{13}$$

  – Desired Acceleration Normalization:

$$\mathbf{n}_{\text{des}} = \frac{\mathbf{a}_{\text{des}}}{\|\mathbf{a}_{\text{des}}\|} \tag{14}$$

  – Inclination and Desired Thrust:

$$\theta = \arccos\left(-\frac{\mathbf{g}\cdot\mathbf{n}}{\|\mathbf{g}\|}\right), \quad T_{\text{des},0} = -m\cdot\frac{a_{z,\text{des}}}{\cos\theta} \tag{15}$$

  – Thrust Scaling:

$$T_{\text{des}} = -\beta\cdot m\cdot\frac{a_{z,\text{des}}}{\cos(\min(\theta,\theta_1))} \tag{16}$$

  – Cost Function for Optimal Control:

$$\min\left(\boldsymbol{\mu}_{\text{des}} - \mathbf{G}\mathbf{u}\right)^\top \mathbf{W}\left(\boldsymbol{\mu}_{\text{des}} - \mathbf{G}\mathbf{u}\right) + \boldsymbol{\lambda}\mathbf{u}^\top\mathbf{u} \tag{17}$$

  The Upset Recovery controller, a three-stage cascaded system, stabilizes the drone by reducing roll/pitch errors and unwanted angular velocities. Prioritizing stabilization, it may sacrifice altitude briefly. Once stabilized, the Fault Tolerant controller reactivates to resume nominal operations. These techniques and equations can be implemented in the future to enhance system robustness, particularly in scenarios involving delayed motor failure detection. They can be integrated into the existing PX4 framework for improved stability and control under extreme conditions.

- **Nonlinear Model Predictive Control (NMPC):**

  For future work, we plan to integrate a Nonlinear Model Predictive Control (NMPC) layer before the existing Incremental Nonlinear Dynamic Inversion (INDI) controller. NMPC will optimize the system's trajectory by solving a constrained nonlinear optimization problem at each control step while ensuring adherence to constraints like actuator limits, state boundaries, and obstacle avoidance. It will predict future states to anticipate disturbances and dynamically adapt control strategies, improving performance. Additionally, NMPC will manage trade-offs, such as minimizing energy consumption and maximizing stability and accuracy. For implementation, we will use optimization libraries like CasADi and leverage parallel processing for real-time feasibility.

5 Appendix: Additional Images and Screenshots and Notations:

5.1 Notations:

This section explains the symbols used in the context of Upset Recovery. The symbols are listed in alphabetical order.

Table 1: List of Symbols and Their Descriptions

Symbol	Description
$\mathbf{a}_{\text{des}}$	Desired acceleration vector after scaling.
$a_{x,\text{des},0}$	Initial desired acceleration along the x-axis.
$a_{y,\text{des},0}$	Initial desired acceleration along the y-axis.
$a_{z,\text{des},0}$	Initial desired acceleration along the z-axis.
β	Scaled and wrapped incline angle.
$\mathbf{g}$	Gravitational acceleration vector.
$\mathbf{n}_{\text{des}}$	Normalized desired acceleration vector.
ϵ	Scaling factor for adjusting acceleration based on inclination.
θ	Current inclination angle of the desired acceleration vector relative to gravity.
θ_1	Minimum threshold inclination angle (30°).
θ_2	Maximum threshold inclination angle (70°).
m	Mass of the system.
$T_{\text{des},0}$	Initial desired thrust.
T_{des}	Adjusted desired thrust after scaling.
ρ	Angle between desired thrust and the z-axis acceleration component.
μ_{des}	Desired control inputs including torque and thrust.
$m^B_{c,des}$	Desired moment in the body frame.
I^B_v	Inertia tensor in the body frame.
α^B_{des}	Desired angular acceleration in the body frame.
ω^B	Angular velocity in the body frame.
$\mathbf{G}$	Control allocation matrix.
$\mathbf{u}$	Control input vector.
$\boldsymbol{\lambda}$	Regularization parameter for the control input cost.
$\mathbf{W}$	Weighting matrix for the cost function.

5.2 Figures:

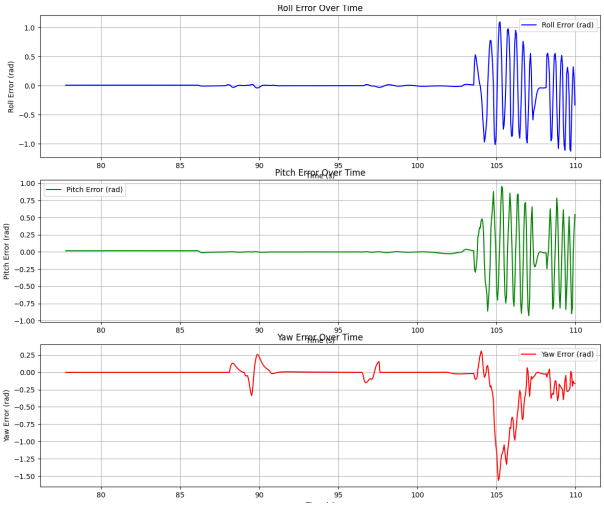


Figure 12: RPY Error Plot (failure at 103.5s)

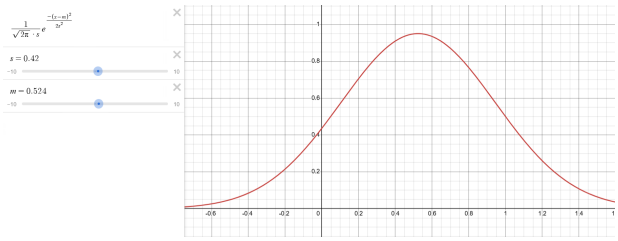


Figure 13: Probability Distribution for Thresholds for Position Mode

Figure 14: Current Hardware for Testing

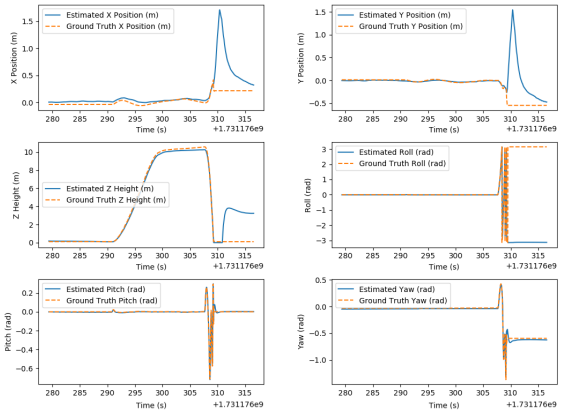


Figure 15: Output of EKF2 after Tuning

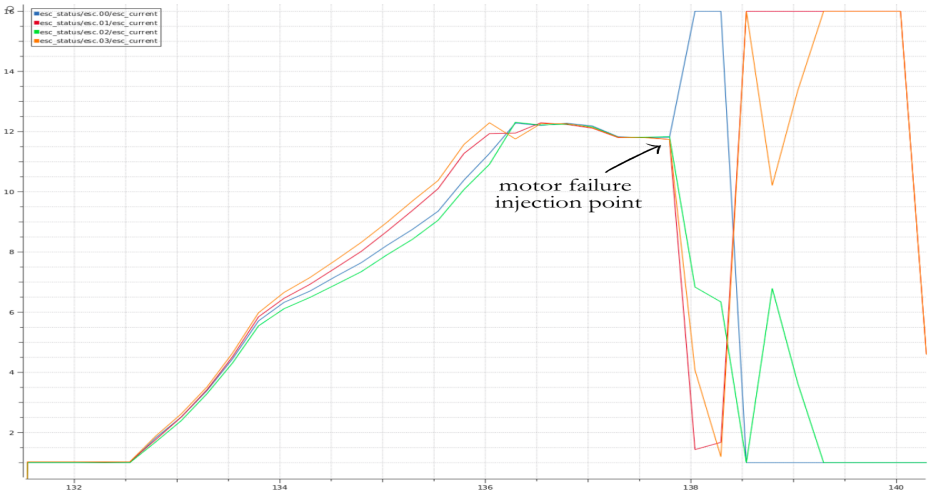


Figure 16: ESC data during motor failure

References

[1] R. Mahony, V. Kumar, and P. Corke, ”Multirotor aerial vehicles: modeling, estimation, and control of quadrotor,” *IEEE Robotics & Automation Magazine*, vol. 19, pp. 20, 2012. Link

[2] S. Sun, L. Sijbers, X. Wang, and C. de Visser, ”High-speed flight of quadrotor despite loss of single rotor,” *IEEE Robotics and Automation Letters*, 2018. Link

[3] Sihao Sun, Xuerui Wang, Qiping Chu, and Coen de Visser, ”Incremental Nonlinear Fault-Tolerant Control of a Quadrotor With Complete Loss of Two Opposing Rotors,” *IEEE Transactions on Robotics*, vol. 37, no. 1, February 2021. Link

[4] Sihao Sun, Matthias Baert, Bram Strack van Schijndel, and Coen de Visser, ”Upset Recovery Control for Quadrotors Subjected to a Complete Rotor Failure from Large Initial Disturbances,” *IEEE International Conference on Robotics and Automation (ICRA)*, 2020. Link