Design, Implementation and Evaluation of a Fault Detection and Emergency Control Strategy for a Quadcopter System

Björn Fredlund

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6257
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2024

Abstract

In this thesis, a fault detection and diagnosis system is developed for the Crazyflie 2.1 unmanned aerial vehicle. Different types of faults are investigated and suitable methods and tests are implemented. The system is mainly developed for the lighthouse positioning, but the tests and methods are generic and suitable for other parts of the Crazyflie eco-system. For the faults investigated, different methods of bringing the quadcopter to a landed state are explored. To keep user customizability high, different default actions are available to specify for each type of fault. The different actions are implemented as an extension of the onboard state machine and will cause the Crazyflie to respond to a fault in a certain way. The results indicate promising results for the developed control strategy but, because of simplicity, already implemented strategies are preferred. Overall, the implementation is simple, flexible and robust. Additionally, a propeller unbalance estimator developed in previous work was implemented and evaluated. The estimator is able to identify the damaged propeller and estimate the unbalance within the correct order of magnitude. It was, however, deemed too model dependent and thus not suitable as a general service indicator across different Crazyflie setups and configurations.

Contents

1

Introduction

In the field of autonomous flight, robust identification and management of fault conditions are crucial for the safe and reliable operation of unmanned aerial vehicles. In autonomous flight, ensuring both the vehicle's and its surrounding safety is paramount. It is essential to identify and appropriately respond to erroneous conditions, such as the loss of position data or anomalous sensor readings, to prevent accidents and minimize potential risks. Sensors can sometimes provide inaccurate or inconsistent data. This thesis aims to equip the Crazyflie 2.1 unmanned aerial vehicle developed by Bitcraze AB with the capability to recognize abnormal sensor readings and initiate controlled recovery mechanisms.

1.1 Goals and objective

The aim of this thesis is to investigate how erroneous states in the Crazyflie can be detected, and how they should be handled. The aim of the thesis is to develop a system to autonomously identify and manage erroneous conditions during autonomous flight. The solution is intended to be implemented and evaluated on the Bitcraze Crazyflie platform. An emergency control strategy is to be developed using only IMU measurements, as it is the only sensor which is always available on the system. The ultimate goal is for this solution to become a permanent part of the system. Following the literature study in Section 2.1 the following is to be investigated in this thesis.

Research questions.

- What strategies can be implemented to ensure autonomous isolation and handling of errors in Bitcraze quadcopters during autonomous control?
- Which control strategies are most effective for different types of sensor faults, and how can they be implemented?
- How transferable are the developed algorithms and methods to different quadcopter models and configurations?

Delimitations

The sensor fault detection system will be developed with the assumption that only one sensor fault will be present at a time, which for the realistic scenario, is a reasonable assumption. The modular design of the Crazyflie's expansions boards proposes an interesting challenge in the realm of sensor fault detection. With different configurations, different sensor measurements will be available to the system. This approach limits the fault detection to errorenous states related to the lighthouse positioning system, but the approach is modular and thus acts as a proof of concept directly applicable to other modules.

1.2 Thesis outline

In Chapter 2 a general introduction to fault detection and different approaches is presented as well as a brief introduction on key components of the Crazyflie. Chapter 3 presents a general framework for quadcopter modelling and contains results needed for later chapters. In Chapter 4, motivation, discussion and implementation details of the chosen approach are presented and in Chapter 5 the conducted experiments and results are presented. The discussion in Chapter 6 is centered around the presented results as well as the methods chosen.

2

Background

2.1 Literature study

There is extensive literature on Fault Detection and Isolation (FDI) and Fault Tolerant Control (FTC) for quadcopters. Much of the focus in literature is on the identification and management of motor faults. However, there is less on sensor faults and their management. Fault diagnostics as a whole commonly consists of two main components, fault detection and fault isolation. These two together constitute Fault detection & isolation (FDI). The basic concept involves using an algorithm that reacts to errors, determining their type and severity, with the goal of implementing changes to mitigate their impact.

Fault detection can be achieved by means of hardware redundancy or analytical redundancy [Chen, 1995]. Hardware redundancy, as the name implies, relies on having access to redundant sensors measuring the same states. Error detection can then be carried out by means of comparing measurements from each of the sensors to one another. Analytical redundancy instead relies on information, analysis and usually the prediction of expected behavior of the system in question. Several methods of analytical fault detection exist which can typically be divided into the following categories, *model-based methods*, *signal-based methods*, *knowledge-based methods*, *hybrid/active methods* [Han, 2021].

Model-based methods

Model-based fault diagnosis relies on a well-established mathematical model of the system, derived from physical principles or system identification techniques [Han, 2021]. The principle idea is to run the model in parallel to the process with the same inputs. In this way, a fault-free system (or nominal system) is created and any deviation between the real process and the modelled one can be detected, and thus indicate that a fault has occurred. This difference between the expected output and the measured one is referred to as the residual. Residual generation is the first part of model-based fault approaches. Common and well-known model-based approaches for residual generation techniques include observer-based techniques,

parity space approaches, and parameter estimation approaches [Khan, 2010][Han, 2021]. A general model-based fault detection scheme and its components including a decision step is illustrated in Figure 2.1. Typical sources of errors could generally be introduced in either the actuators, process or sensors as highlighted in the figure.

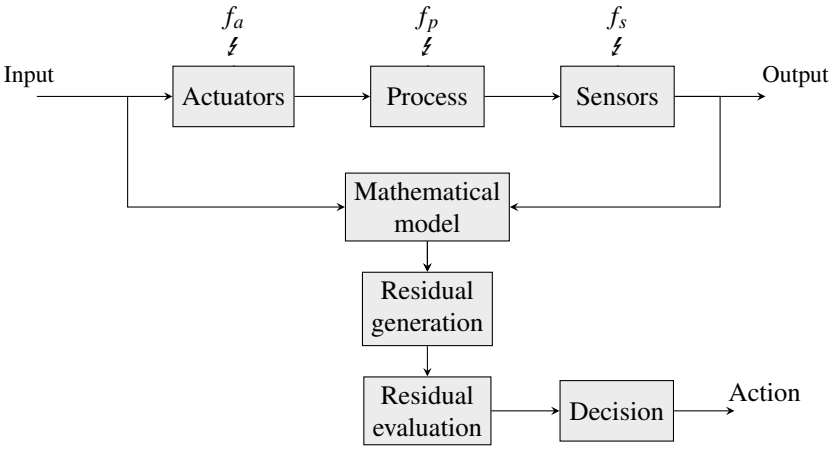


Figure 2.1 Model-based fault detection scheme including post processing and action output.

Residual generation. In observer based fault detection, the idea is to generate the residual by comparing the system outputs with their estimates. It is one of the more commonly applied approaches in fault detection because of their flexibility and simplicity. Different types of observers can be used depending on the situation, for example can the traditional Luenberger observer be used in a deterministic setting or the Kalman filter in a stochastic setting [Frisk, 2001] [Chen, 1995]. Different nonlinear observers are also viable such as the Extended Luenberger observer, Nonlinear identity observer, Unknown input observer (UIO) or the Extended Kalman filter [Khan, 2010].

Residual evaluation. Residual evaluation is the second part of model-based fault detection. The residual should normally be zero or close to zero when no fault is present, but distinguishable different from zero when a fault is present. However, due to the presence of disturbance, noise, and modeled uncertainties, the residual is almost never exactly zero. For stochastic systems specifically, the residual is typically very noisy. The residual generator is then often designed to specifically suppress the effect of this noise. A common approach is to use a Kalman filter-based residual generator [Chen, 1995]. With the knowledge and assumption of process noise in the model, the residual is often evaluated with statistical means such as the weighted sum-squared residual (WSSR), χ^2, generalized likelihood ratio (GLR) or using a cumulative sum (CUSUM) [Chen, 1995].

Fault tolerant control

Fault tolerant control (FTC) can be divided into two main categories, passive FTC (PFTC) and active FTC (AFTC) [Ducard, 2007]. PFTC revolves around employing inherently robust control strategies, which are largely unaffected and good at dealing with large uncertainties. For PFTC, fault detection is often not considered as the controllers are designed to cope with different types of faults on the actuators or sensors. AFTC on the other hand involves switching control strategies when a fault has been detected. The control strategy could then be better tailored to cope with the detected fault but this solution requires more infrastructure and a well behaving FDI. The general outline of an AFTC scheme is illustrated in Figure 2.2.

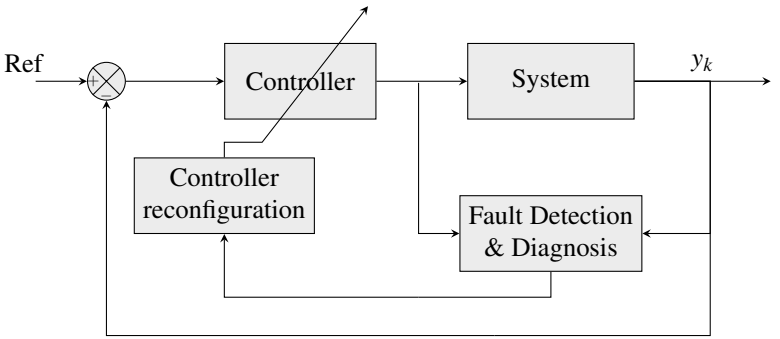


Figure 2.2 Active fault tolerant control scheme.

2.2 PID control

Continuous form

A proportional-integral-derivative controller (PID controller) is a commonly used control method used in control engineering [Åström and Hägglund, 1995]. It utilizes feedback to control a measured control variable to a desired setpoint. It does this by continuously calculating the error, $e(t)$, between the measured value and the desired setpoint. In the continuous case, the PID control signal takes the form of

$$u = K_p e(t) + K_i \int e(t)dt + K_d \frac{de}{dt} \tag{2.1}$$

where K_p, K_i and K_d are the gains of the proportional, integral and derivative terms respectively, which are all tuning parameters, that makes the controller behave differently to disturbances, noise and reference changes.

Discrete form

To implement a continuous PID controller in a digital setting, for example in a computer, the integral and derivative parts have to be discretized. The continuous form of (2.1) then takes the form of (2.2).

$$\begin{aligned} u(k) &= K_p e(k) + K_i I(k) + K_d D(k) \\ I(k) &= I(k-1) + e(k) \\ D(k) &= \frac{e(k) - e(k-1)}{h} \end{aligned} \tag{2.2}$$

where k denotes the time step and h the sampling period. The formulation of (2.2) is generally not a sufficient implementation for a well functioning PID-controller and several improvements are needed. The reader is referred to [Hägglund, 2021], [Wittenmark et al., 2002] or any other textbook on control theory for an in depth derivation of the discrete-time PID control formulation.

2.3 Crazyflie overview

The Crazyflie 2.1 is a small quadcopter developed by Bitcraze AB [Bitcraze, 2021a]. It is a versatile tool used by researchers and more ambitious laymen. The project is entirely open-source, which enables full customization and modification of the firmware. Onboard is a gyroscope, accelerometer and barometer.

Lighthouse positioning system

The lighthouse positioning system is a highly accurate indoor positioning system developed by Bitcraze. It uses SteamVR Base Stations as optical beacons and light receivers (photo-diodes) on the lighthouse deck, mounted on the Crazyflie (or any other compatible device) to calculate its position with millimeter precision [Bitcraze, 2023d]. The basestations spin drums with a laser to create a "sweeping" effect around its environment. With the sweep information, the position and orientation of an object can be computed if the corresponding basestation position is known. Even though millimeter precision is possible, because of geometry, the performance severely degrades whenever the receiving object comes closer than approximately 50 cm below the basestations. As it is an optical system, a clear line of sight to at least one of the basestations is required. For lighthouse V2 the sweep rate is approximately 50 Hz.

OW protocol

The one-wire protocol is as the name implies a single wire communication protocol. The one-wire standard operates on a master slave configuration where up to 100 slaves can be connected on the bus [Agnihotri, n.d.] . The one-wire protocol is used on the Crazyflie during startup to identify which expansion boards are connected.

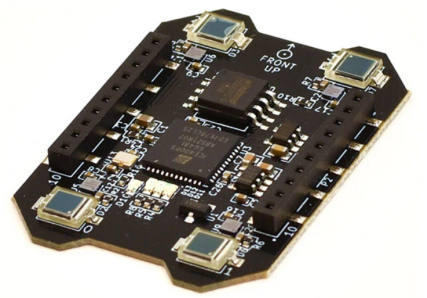


Figure 2.3 Lighthouse positioning deck [Bitcraze, 2023c].

After the connected boards have been identified, the correct drivers are loaded and the appropriate estimator and controller is chosen.

freeRTOS

The Crazyflie runs freeRTOS, which is a free real-time operating system. It is lightweight, fast and offers a multitude of synchronization primitives needed for concurrent programming.

2.4 Control structure

The default control method on the Crazyflie is a set of cascaded PID controllers that map the desired input to one another [Bitcraze, 2023a] as illustrated in Figure 2.4. The innermost controllers are responsible for attitude control and receives measurements from the IMU and gyroscope. On top of this is the velocity controller and position controller which maps a desired position or velocity to a desired roll and pitch angle u_ϕ, u_θ, which is fed into the attitude controllers. Depending on the configuration, the xyz controllers can be bypassed by specifying a control mode in the setpoint structure. The possible control modes include *ModeAbs*, *ModeVelocity* and *ModeDisable*, which as the names imply, bypasses and selects which reference and input to track as illustrated in Figure 2.4. All setpoints are generated by a high-level commander which is then passed on to the control structure. If not specified, the setpoints are in the global reference frame, and thus needs to be rotated to the inertial frame through (2.3) using the current yaw orientation ψ. The z velocity controller additionally encompasses a feedforward term according to (2.4) to improve tracking and counteract gravity.

$$\begin{bmatrix} u_{\phi B} \\ u_{\theta B} \end{bmatrix} = \begin{bmatrix} -\cos(\psi) & -\sin(\psi) \\ \sin(\psi) & -\cos(\psi) \end{bmatrix} \begin{bmatrix} u_{\phi G} \\ u_{\theta G} \end{bmatrix} \tag{2.3}$$

$$u_{thrust} = u_{ff} + u_z \tag{2.4}$$

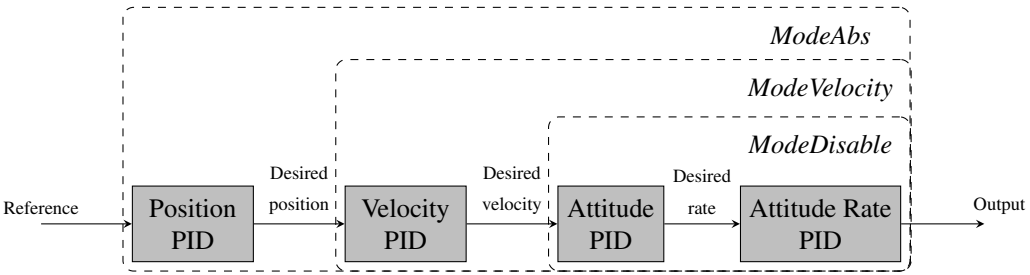


Figure 2.4 Crazyflie cascaded control structure including the different modes available to select in the setpoint structure.

2.5 Supervisor and stabilizer module structure

The stabilizer loop is at the heart of the Crazyflie and coordinates everything from sensor acquisition to control output [Bitcraze, 2023e]. The loop runs at 1 kHz and processes and synchronizes different tasks in the system according to Figure 2.5. Implemented in the firmware is also a supervisor that monitors against faulty behaviour and has full right to override any setpoint generated by the commander. The supervisor includes a state machine, which enables easy error handling and abstracts away key flight modes such as *flying*, *crashed*, *preflight checks* and more. The current implementation overrides the setpoint of the commander and levels out the Crazyflie whenever a setpoint watchdog, updated by external communication, has failed to be reset. The levelout mechanism simply commands roll and pitch to zero leaving the control mode and setpoint in the Z direction to whatever was previously commanded, control is then given back whenever communication is restored.

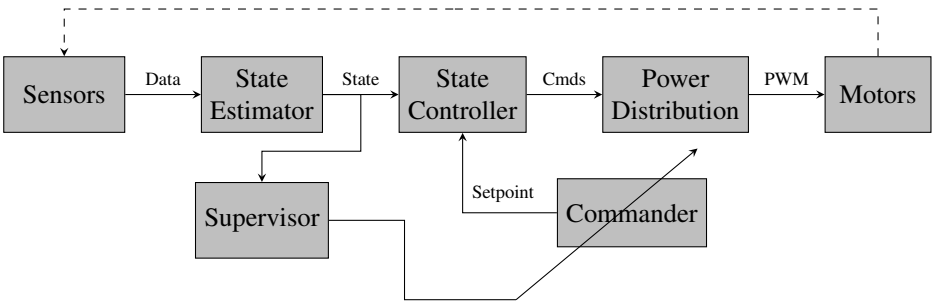


Figure 2.5 Stabilizer loop, adapted from [Bitcraze, 2023e].

2.6 Lowpass filter

A lowpass filter is a filter that only lets signal below a defined cutoff frequency through, while suppressing signal with higher frequencies. A lowpass filter can be implemented analog through special circuits or digitally by converting the analog continuous version to a discrete digital version. A recursive filter is a popular and efficient way to construct a digital filter without having to perform long convolutions. These filters are also generally known as infinite impulse response (IIR) filters. A general N-order recursive filter takes the form of (2.5) [Wickert, 2022]

$$y[n] = -\sum_{k=1}^{N} a_k y[n-k] + \sum_{r=0}^{M} b_r x[n-r] \tag{2.5}$$

where $y[n]$ is the output and $x[n]$ is the n:th input sample and a_k and b_r are known as the recursive coefficients. The relationship between the recursion coefficients and the filter's response is given by the Z-transform [Smith, 1999]. Consider for example the general second order lowpass filter in the continuous s domain

$$H(s) = \frac{1}{s^2 + \frac{s}{Q} + 1} \tag{2.6}$$

$$s = \frac{1}{K}\frac{z-1}{z+1} \tag{2.7}$$

After substituting (2.7), known as the bilinear (or Tustin's) transform [Redmon, 2003] in (2.6). We get following after some manipulation and further substitution

$$H(z) = \frac{K^2 + 2K^2 z^{-1} + K^2 z^{-2}}{(K^2 + \frac{K}{Q} + 1) + 2(K^2 - 1)z^{-1} + (K^2 - \frac{K}{Q} + 1)z^{-2}} \tag{2.8}$$

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.9}$$

Comparing this to a Z transformed version of (2.5), that is (2.9), for $N = 2$ with $H(z) = \frac{Y(z)}{X(z)}$ we can identify the coefficients as

$$a_0 = 1 \qquad a_1 = 2(K^2 - 1) \qquad a_2 = K^2 - \frac{K}{Q} + 1 \tag{2.10}$$

$$b_0 = K^2 \qquad b_1 = 2b_0 \qquad b_2 = b_0 \tag{2.11}$$

where $K = \tan(\pi \frac{F_c}{F_s})$, F_c is the cutoff frequency of the filter and F_s is the sample frequency. The structure of (2.5) is used in the Crazyflie to filter the accelerometer, gyroscope and PID controllers output.

2.7 Kalman filter

On the Crazyflie, an extended Kalman filter developed in the works of [Mueller et al., 2015] [Mueller et al., 2016] is implemented. A Kalman filter is in the linear case an optimal recursive filter that estimates states $\hat{x}_k$ given a system dynamics model and noisy measurements y_k [Simon, 2006]. The extended Kalman filter is a nonlinear extension of the original Kalman filter where the dynamics are linearized around the estimate and the estimate is based on the linearized system [Simon, 2006]. Consider for example the following discrete nonlinear system model

$$\begin{aligned} x_k &= f(x_{k-1}, u_{k-1}) + w_{k-1} \\ y_k &= h(x_k) + v_k \end{aligned} \tag{2.12}$$

where $f(\cdot)$ is the system equation and $h(\cdot)$ the measurement equation, both of which are nonlinear functions and w, v are zero mean independent normal distributed noises with constant covariance $\mathbf{Q}$ and $\mathbf{R}$. Performing a Taylor series expansion around $x_{k-1} = \hat{x}_{k-1}$ and $w_{k-1} = 0$ for the system model and around $x_k = \hat{x}_k$ and $v_k = 0$ for the measurement model results in a linear system of equations and the standard Kalman filter formulation can be used to estimate the states according to Algorithm 1.

Algorithm 1 EKF

EKF predict

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_{k-1}) \tag{2.13}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1}\mathbf{P}_{k-1|k-1}\mathbf{F}_{k-1|k-1}^T + \mathbf{Q} \tag{2.14}$$

EKF correct

$$\varepsilon_k = y_k - h(\hat{x}_{k|k-1}) \tag{2.15}$$

$$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_k\mathbf{H}_k^T + \mathbf{R} \tag{2.16}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}_k^T\mathbf{S}_k^{-1} \tag{2.17}$$

$$\hat{x}_{k|k} = \hat{x}_{k_|k-1} + \mathbf{K}_k\varepsilon_k \tag{2.18}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k|k-1} \tag{2.19}$$

where $\mathbf{F}_k$ and $\mathbf{H}_k$ are the jacobians of the functions f and h at sample time k.

$$\mathbf{F}_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k|k}} \tag{2.20}$$

$$\mathbf{H}_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k|k-1}} \tag{2.21}$$

The EKF on the Crazyflie has a nine dimensional state vector ξ:

$$\xi = (\boldsymbol{x}, \boldsymbol{\rho}, \boldsymbol{\delta}) \tag{2.22}$$

where $\mathbf{x}$ is the position in the inertial frame, $\boldsymbol{\rho} = \mathbf{R}^{-1}\dot{\boldsymbol{x}}$ the velocity in the body frame rotated by using the rotation matrix $\mathbf{R}$ described in Section 3.1, and $\boldsymbol{\delta}$ an attitude error representation [Mueller et al., 2015]. For a detailed explanation and full derivation of the equations, the reader is refereed to the original paper and source-code implementation. When the lighthouse deck is mounted, the EKF estimates position and a heading error through the sweep measurements. The measurement model can be found in [Bitcraze, 2023b]. As different measurements arrive at different time steps and depending on the configuration, different measurements may be available to the system.

The implemented EKF is a scalar updated version different from the formulation presented in Algorithm 1. The scalar implementation additionally removes a matrix inversion in (2.17) as S_k is scalar, which can be beneficial for embedded real-time systems. Additionally, to improve performance and decrease the risk of estimator divergence, a Joseph correction step is integrated that ensures symmetry on the covariance matrix on each correction step. The interested reader is refereed to [Greiff, Marcus, 2017] for the full formulation of the scalar updated Joseph-form EKF with enforced symmetry.

In addition to the EKF, a complementary filter is available on the Crazyflie. A complementary filter estimates roll, pitch and yaw through sensor fusion of the accelerometer and gyroscope [Bitcraze, 2023f]. The complementary filter is efficient and lightweight and is intended to be used for manual control.

3

Modelling

The following section introduces and describes the fundamentals regarding quadcopter modeling and serves to give a deeper understanding as well as a mathematical frame to stand on. First, concepts regarding reference frames and force interaction are introduced, then rigid body dynamics based on Newton-Euler equations is derived and finally, a brief discussion on numerical properties and parameters related to the Crazyflie is presented.

3.1 Attitude angles and reference frames

The attitude of a vehicle describes how it is orientated in one reference frame with respect to another one. There are several ways to represent the attitude of a vehicle with Euler angles being one of the more common approaches. The coordinate system used for the Crazyflie is in the east-north-up (ENU) convention [Bitcraze, 2023g] and the corresponding Euler angles, roll (ϕ), pitch (θ) and yaw (ψ) are illustrated in Figure 3.1.

The measurements of the sensors in the Crazyflie in its body reference frame relative to the global inertial one can be described using Euler angles. By applying three consecutive rotations, the inertial frame, denoted $\dot{}_G$, is transformed to the Crazyflies body frame, denoted $\dot{}_B$. Denoting $c_i = \cos(i)$, $s_i = \sin(i)$, the rotation matrices, in a ZYX configuration, equate to

$$\mathbf{R}_z(\psi) = \begin{bmatrix} c_\psi & s_\psi & 0 \\ -s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}_y(\theta) = \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix}, \quad \mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & s_\phi \\ 0 & -s_\phi & c_\phi \end{bmatrix} \tag{3.1}$$

which leads to the following relation, transforming a vector from the inertial frame (G) to the Crazyflies body frame (B).

$$\mathbf{R}_{GB} = \mathbf{R}_x(\phi)\mathbf{R}_y(\theta)\mathbf{R}_y(\psi) = \begin{bmatrix} c_\theta c_\psi & s_\psi c_\theta & -s_\theta \\ c_\psi s_\theta s_\phi - s_\psi c_\phi & s_\psi s_\theta s_\phi + c_\phi c_\psi & c_\theta s_\phi \\ c_\psi s_\theta c_\phi + s_\phi s_\psi & s_\psi s_\theta c_\phi - c_\psi s_\phi & c_\phi c_\theta \end{bmatrix} \tag{3.2}$$

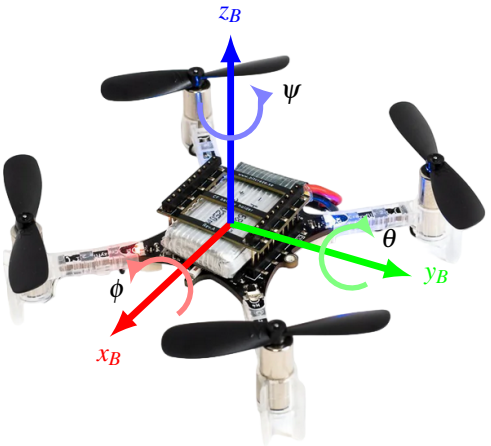


Figure 3.1 Yaw, pitch and roll visualized including the quadcopter body frame.

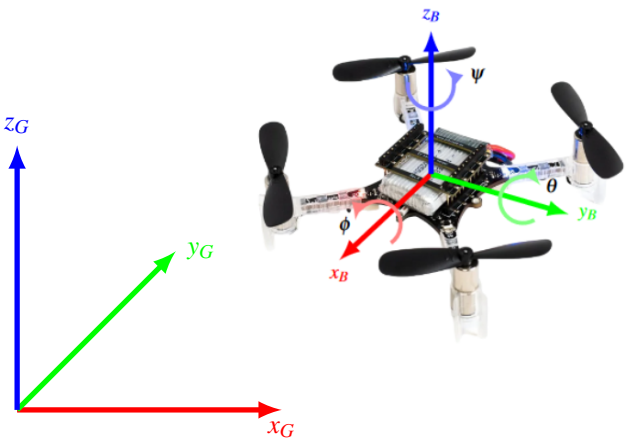


Figure 3.2 Quadcopter and inertial frame.

In the Crazyflie, Euler angles are not utilized to represent reference frames because of the computational overhead of evaluating trigonometric functions and the potential for singularities during aggressive flights. This can be seen in (3.2) for $\theta = (n+1/2)\pi$, where ambiguity arises because of the inability to distinguish between rotations around the roll and yaw axes. This singularity is known as Gimbal lock [Greiff, Marcus, 2017]. Instead, an attitude representation known as the quaternion is used. A quaternion is commonly defined as

$$Q = a + bi + cj + dk \tag{3.3}$$

where $a, b, c, d \in R$ and i, j, k imaginary units. In compact notation using polar coordinates and Euler's identity the definition becomes

$$\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} q_w \\ q_v \end{bmatrix} \tag{3.4}$$

With the quaternion representation and its operators, the mapping between x_G and x_B becomes

$$x_G = R_{BG} x_B = R_{GB}^{-1} x_B = R_{GB}^{T} x_B \qquad \begin{bmatrix} 0 \\ x_G \end{bmatrix} = q_{BG} \otimes \begin{bmatrix} 0 \\ x_B \end{bmatrix} \otimes q_{BG}^{*} \tag{3.5}$$

where q^* denotes the complex conjugate and $\otimes$ the quaternion product and R_{BG}

$$R_{BG} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{3.6}$$

The quaternion representation eliminates Gimbal lock, has better numerical stability and reduces computational load as no trigonometric evaluation is needed. It is, however, less intuitive and a requires a familiarity with quaternion algebra. The interested reader is referred to for example [Greiff, Marcus, 2017] or [Diebe, 2006] for a full derivation and explanation of the relations.

3.2 Forces and moments

The Crazyflie is an X-configuration quadcopter where two opposing propellers rotate in a clockwise rotation (CW) and the other two counter-clockwise (CCW). Each rotating propeller produces a force along the Z_B axis of the quadcopter, this force is given by

$$f_i = k_i \omega_i^2 \tag{3.7}$$

where ω_i is the is the rotation speed of motor i, measured in [rad/s] and k_i is a constant depending on factors such as torque proportionality, surrounding air density, air velocity, area swept by propeller and other unmodelled factors. The total thrust along Z_B is thus given by the sum of each motor thrust.

$$T_B = F = \sum_{i=1}^{4} k_i \omega_i^2 \tag{3.8}$$

Apart from thrust, these rotational speeds also give rise to a torque acting on the opposite direction of the rotation of ω_i. For example, a rotor spinning in the CW direction will give rise to a moment in the CCW direction. The torques or moments

are denoted τ_i and can similarly to (3.7) be modelled as proportional to the rotor speed ω_i^2 scaled by a factor k_m

$$\tau_{Mi} = k_m \omega_i^2 \tag{3.9}$$

How the forces and torques act on the body is captured in Figure 3.3

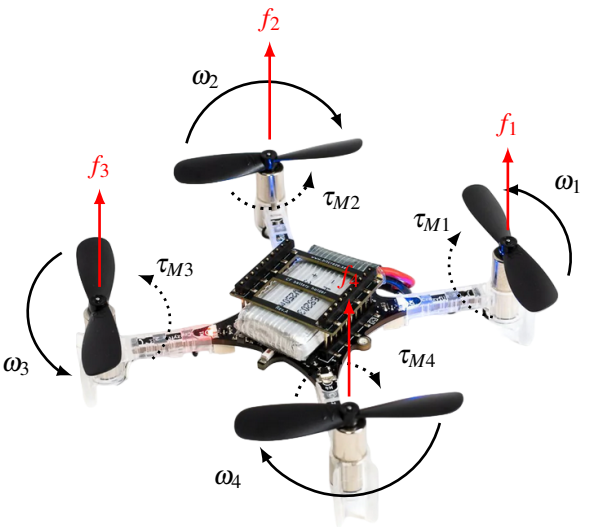


Figure 3.3 Forces and moment acting on the quadcopter.

3.3 Dynamics

Quadcopter modelling has been studied extensively in literature. The dynamic models could be derived either using Newton-Euler or Lagrange dynamics. Here, the dynamics are derived assuming a rigid body using the Newton-Euler equations model. For a full derivation and a more in-depth description, the reader is referred to [Luukkonen, 2011] or [Greiff, Marcus, 2017].

Newton-Euler equations

Following the assumption that the quadcopter is a rigid body, combining the total thrust produced by the force of the propellers in (3.8) with the gravitational force and Newton's law gives

$$m_b \ddot{x} = R e_3 T_B - m_b g \tag{3.10}$$

in the inertial frame for the translational motion of the system, where m_b is the mass of the quadcopter and R a rotation matrix from the body frame to the inertial one,

for example (3.6). The above relation can be rewritten as follows

$$\ddot{x} = \frac{1}{m_b} Re_3 T_B - g \tag{3.11}$$

which can be used to calculate the force required to produce an acceleration in the inertial frame. In the body frame, the angular velocity ω_b of the Crazyflie and angular acceleration $\dot{\omega}_b$ is a function of the external moments τ_b and the quadcopters moment of inertia I_b according to Euler's equation

$$\tau_b = I_b \dot{\omega}_b + \omega_b \times (I_b \omega_b) \tag{3.12}$$

which constitutes the rotational dynamics of the system.

3.4 Numerical parameters of the Crazyflie

Since the Crazyflie is a research tool, many studies of its physical parameters have been conducted. To control the motors on the Crazyflie, pulse width modulation (PWM), denoted D (an unsigned 16 bit integer on the Crazyflie 2.1), is used to set the voltage across the motors and consequently its rotational speed. In the works of [Bitcraze, 2021b] and [Hönig, 2021], several relationships between the Crazyflie 2.1 and its physical parameters were established. These include PMW to thrust, PWM to RPM and thrust to power. Since no feedback about the rotational speed of the individual motors exist, which would for example enable the use of (3.7), these relations act as a good open loop alternative whenever information about unmeasured quantities are needed.

$$T_g = 1.65 \times 10^{-9} D^2 + 9.44 \times 10^{-5} D \tag{3.13}$$

In Table 3.1, a summary of physical properties used throughout the thesis is presented. The thrust coefficient k_f was determined by averaging a hover PWM value and utilizing the PWM to RPM relation (3.14) together with (3.7). How well (3.13), which establishes the relationship between PWM to thrust in grams, T_g, tracks real life dynamics is illustrated in Figure 3.4, where a comparison between (3.13) which has been used to calculate the quadcopter's accelerations through (3.11), and the accelerometer output, which has been rotated to the world frame through (3.6) is given.

$$RPM = 0.34 D^2 + 3302.63 D \tag{3.14}$$

Table 3.1 Parameters of the Crazyflie quadcopter

Symbol	Description	Value	Unit
m_b	mass of the Crazyflie	34	[g]
r_p	Propeller radius	22	[mm]
k_i	Thrust coefficient	3.38	$[\mathrm{Ns}^2/\mathrm{rad}^2 \times 10^{-7}]$

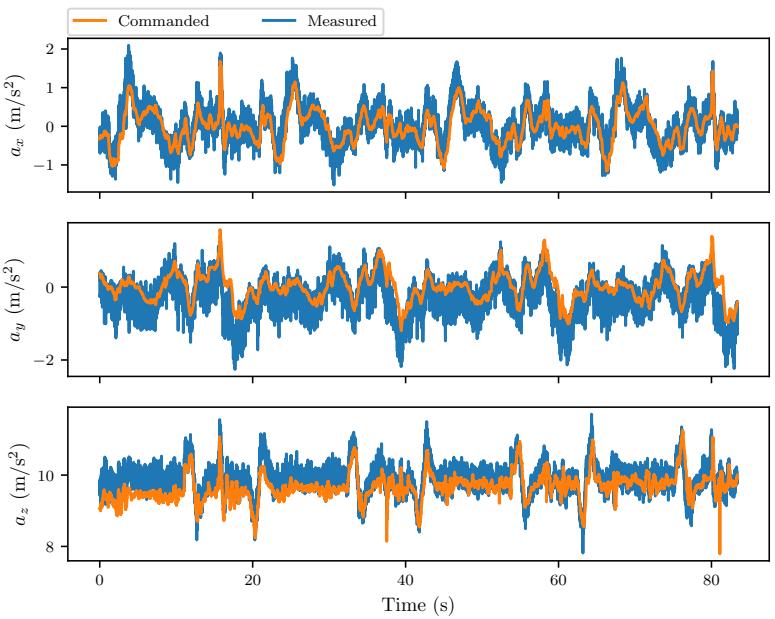


Figure 3.4 Comparison between commanded and measured acceleration rotated to inertial frame.

4

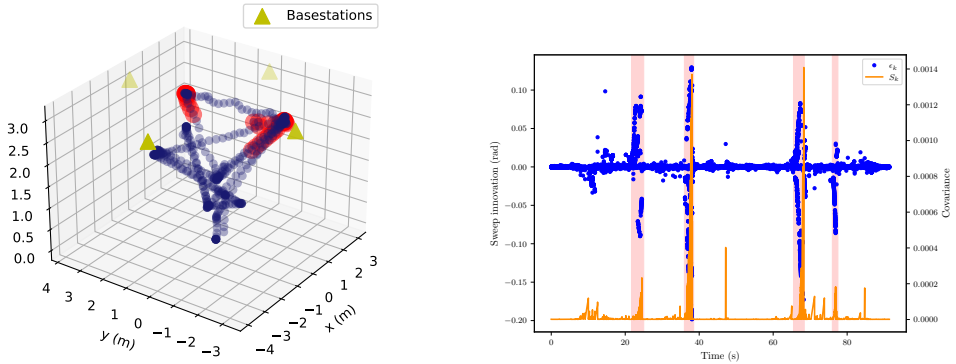
Method

4.1 Residual generation

Residual generation is as mentioned in Section 2.1 the first step in model-based fault detection. It is the base on which the detection algorithm relies, and thus it is of key importance that anomalous behaviour is captured in them. Since there is already an EKF implemented on the Crazyflie, it is naturally chosen as the source of residual generation. In principle, any output of the estimator that can be symptomatic of a fault can be used as a residual. For example, elevated state covariance or states that have exceeded a known physical property, like position or velocity. The residual, identified as the innovation in (2.15), is however the most typical source of a residual. The innovation is then typically also transformed to a *distance measure*, denoted s_t, which together with a *stopping rule* is used to determine if a change to the no fault hypothesis has happened [Gustafsson, 2000]. The choice of a *distance measure* is dependent of what type of change one is trying to detect. If a detection of change in the mean is of interest, the residual itself can be used, as in [Nilsson, 2020]. Other possible distance measures include a change in variance, change in correlation or change in sign correlation.

An example of what a disturbance look like for the lighthouse system and what is of interest to detect can be seen in Figure 4.1. This is a typical example of where the Lighthouse system struggles to receive good measurements from the basestations. The position of the Crazyflie during time frames of large sweep measurement errors are highlighted in red in Figure 4.1a, the corresponding timeframes and error spread is visualized in Figure 4.1b. This phenomenon occurs, for this particular setup, at heights above 2.5 m and close to the edges of the flying area. Different setups and configurations exhibit different behavior in various positions. However, the lighthouse system commonly struggles when the line of sight is obstructed, the drone is flying too high with respect to the base stations, the calibration is off, or due to external disturbances such as reflections.

Normalized innovation squared. For this, a *distance measure* called the normalized innovation squared is used [Dingler, 2022]. The normalized innovation



(a) **(b)**

Figure 4.1 Position coordinates visualized along with the corresponding sweep innovation and covariance. The Crazyflies position in the highlighted time windows in (**b**) is highlighted with red in (**a**).

squared, η_k, NIS for short, is defined as

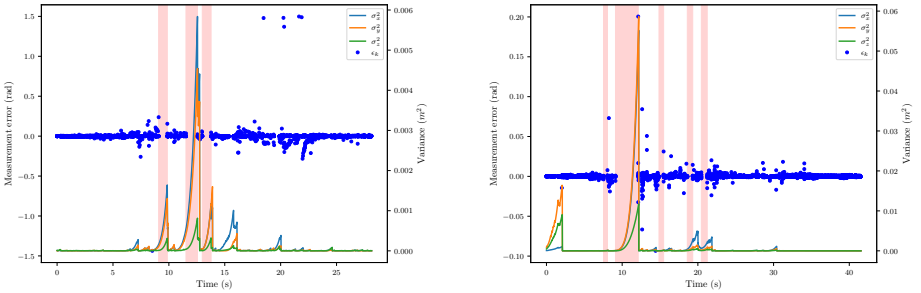
$$\eta_k = \varepsilon_k^T S_k^{-1} \varepsilon_k \tag{4.1}$$

or in the scalar case

$$\eta_k = \frac{\varepsilon_k^2}{s_k} \tag{4.2}$$

where ε_k is identified as the innovation in (2.15) and s_k is the innovation covariance (2.16). The normalized innovation squared essentially checks if the Kalman filter is consistent and can easily detect filter divergence. This is easily seen in the formulation since if the innovation is large relative to how certain the filter is, the metric will be large, indicating that the measurement is not in good agreement with the filter. On the contrary, if the filter is unsure of its estimate, a large innovation will not inflate the metric. This property is specifically advantageous and appropriate for the lighthouse system, as the measurements are prone to sporadic outages, because of line of sight obstruction, where only the *predict* step in the Kalman filter formulation, Algorithm 1, is ran. During the *predict* step, the system dynamics as well as the covariance matrix is pushed forward, which increases the uncertainty of the filter.

After an outage, the innovation will often be large because of the inability of the underlying model to accurately capture real life dynamics for extended periods. In these scenarios the normalized innovation squared will put less emphasis on the large error until the filter converges again, which typically requires a few samples. This is illustrated in Figure 4.2 where the sweep innovation is showcased together with the position variance of the filter and outages of different lengths are highlighted.



(a) **(b)**

Figure 4.2 Sweep measurement innovation and position state variance. Outages longer than 0.3 s (**a**) and 0.5 s (**b**) are highlighted.

4.2 Residual evaluation

As mentioned in Section 2.1, residual evaluation is the second step in the general model-based fault detection scheme, Figure 2.1, and plays a significant role when it comes to performance of the fault detection scheme. In the ideal scenario, the residual generator produces residuals which are different from zero only when a fault is present, however because of unmodelled behaviour and process noise this is normally not the case. Residual evaluation in model-based fault detection is achieved by applying an evaluation function together with a simple threshold. This evaluation function must in an appropriate fashion deal with the noise in the residuals. The evaluation function could be as simple as a test on the instantaneous or average of the *distance measure* or be based on a statistical test such as described in Section-2.1. Most importantly, the evaluation function must consider the trade-off between fast and reliable detection.

CUSUM test. In this approach, a *CUmulative SUM* (CUSUM) is used [Granjon, 2013]. The cumulative sum is a commonly applied method to detect any shift in the mean of a time series and is frequently used in fault detection. It fulfils many of the demands required to be a viable option in an embedded real-time system and is thus a suitable option. It is lightweight, simple, and easy to tune to either prioritize fast detection or reliability. The original cusum formulation of [Page, 1954] is illustrated in Algorithm 2. In the formulation only a shift in the positive direction can be detected, however, the formulation can easily be extended to include detection in the negative direction as well. This is commonly referred to as the two-sided cusum test, but is not used since the normalized innovation squared is strictly non-negative. In the formulation, g_t is an auxiliary test statistic that together with a threshold (or *stopping rule*) is used to raise an alarm.

The main principle in a cumulative sum is to sum up the test statistic, g_t, with its input, s_t, and raise an alarm if the metric exceeds a threshold. Given a white

Algorithm 2 CUSUM

Input: s_t

$$g_t = g_{t-1} + s_t - v$$
$$g_t = 0 \text{ if } g_t < 0$$
$$g_t = 0 \text{ if } g_t > h > 0 \text{ and alarm}$$

Design parameters: drift v and threshold h

noise input, the metric will drift much similar to that of a random walk. To prevent this behaviour, a drift term v is subtracted at each evaluation but capped at zero to prevent a drift in the negative direction. Several other variants of this original formulation exists, which has usually been developed to incorporate some other desirable feature. An example of this is the cusum recursive least squares formulation [Gustafsson, 2000], Algorithm 3. This formulation combines a smoothing filter to the input of the test statistic by applying a forgetting factor λ. The assumption is that during normal conditions, the input, y_t, undergoes small changes, which is caught by the forgetting factor, but occasionally undergoes abrupt changes, which is expected behaviour in our case as can be seen for example in Figure 4.1.

Algorithm 3 CUSUM RLS

Input: y_t

$$\hat{\theta}_t = \lambda \hat{\theta}_{t-1} + (1-\lambda) y_t$$
$$s_t = y_t - \hat{\theta}_t$$
$$g_t = \max(g_{t-1} + s_t - v, 0)$$
$$\text{Alarm if } g_t > h$$

After an alarm, reset $g_t = 0$ and $\hat{\theta}_t = y_t$
Design parameters: drift v, threshold h and forgetting factor λ

For the CUSUM RLS, the idea behind setting $\hat{\theta}_t = y_t$ after an alarm is that the algorithm instantly forgets all previous information, while at the same time avoiding bias and a transient [Gustafsson, 2000]. Worth noting is also that setting $\lambda = 1$ reverts the algorithm back to the original cusum test according to Algorithm 2.

Change detection action

As recommended in [Gustafsson, 2000], an appropriate action whenever an alarm has been detected is to reset the CUSUM and momentarily increase the process noise covariance matrix, $\bar{Q} = \alpha Q$ in the *predict* step of the kalman filter formulation, Algorithm 1. This can be viewed as forcing the filter into the right direction under the assumption that it is the model and not the measurement that is wrong.

An example of this would be whenever the Flowdeck experiences a sudden altitude change, as it is the relative and not absolute height that is measured. However, for example with the lighthouse or any of the positioning decks any long-worthy elevated NIS level is in this context, best to be treated as the opposite, that is, that bad measurements have been repeatedly fed into the EKF. A corresponding suitable action would then involve aborting the mission and trigger a descent. A more elaborate discussion on change action is presented in Section 4.4.

For this scenario, the developed strategy to distinguish a true positive from a false one, is based on empirically knowledge of the residuals. For example, the residuals are prone to outliers often many magnitudes of order greater than the nominal case. A sufficiently high threshold would of course compensate for this at the expense of increasing the time to detection for other fault scenarios. But, as a fast time to detection is of great importance during flight to prevent a crash and under the assumption that outliers do not occur frequently, the approach is to require several violations in a given timespan to consider it a true positive. This is commonly refereed to as a *run test* and is often used to to reduce false alarm rate and increase robustness [Gustafsson, 2000]. An alternative would have been to simply filter out any NIS measurements above a certain threshold. But this, for obvious reasons, would make the fault detection insensitive to large instantaneous faults.

Additionaly, due to the EKF not being an optimal state estimator, and the system dynamics of a quadcopter being highly nonlinear, the residuals are expected to be elevated during aggressive flight maneuvers. This is exemplified in Figure 4.3 that illustrates how the sweep and heading residuals are affected during rapid acceleration and deceleration. As can be seen, the NIS is increased during high velocity, and so is the spread as can be seen in the plot on the third row, which shows a 0.5 s windowed standard deviation of the innovation including the process standard deviation R that the filter has been tuned for.

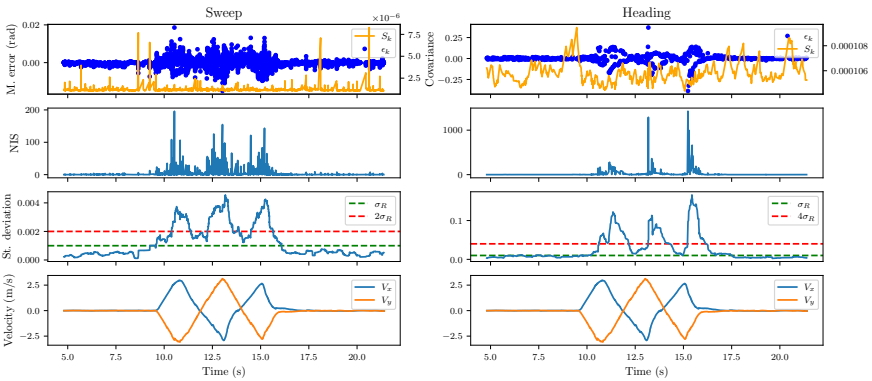


Figure 4.3 Illustration of how sweep and heading measurement innovation, covariance and NIS is affected during complex maneuvers.

To compensate for this and allow for fast detectability for all flight maneuvers, a linear adaptive threshold based on the current velocity magnitude $\|\mathbf{v}|$, is applied

$$h = k\|\mathbf{v}\| + m \tag{4.3}$$

where h is the threshold in Algorithm 3 and k and m is the slope and y-intercept respectively. Since the NIS is not perfectly correlated to the velocity, but rather the aggressiveness of the flight, the velocity magnitude is lowpassed filtered through (2.5) to smoothen and capture the low frequency flight characteristics. The filter is designed with a cutoff at 0.3 Hz and a sample frequency of 100 Hz coming from the predict step in the Kalman filter. A summary of the approach discussed so far is illustrated in Figure 4.4.

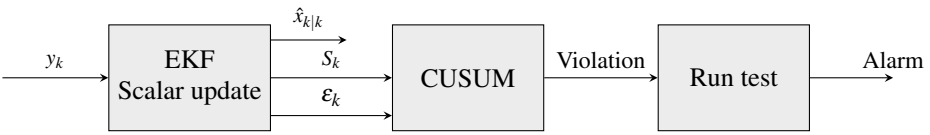


Figure 4.4 Change detection and run test using CUSUM on NIS interaction with the EKF.

Outlier filter

Due to reflections and noise, the lighthouse system is as previously mentioned subject to sporadic outliers. To account for this, all sweep measurements are passed through an outlier filter. The current outlier filter implementation thresholds the height error, that is h in Figure 4.5, which is calculated through (4.4) and (4.5) with a fixed threshold T.

$$r = \sqrt{(x_p - x_0)^2 + (y_p - y_0)^2} \tag{4.4}$$

$$h = r \cdot \tan(\theta_\varepsilon) < T \tag{4.5}$$

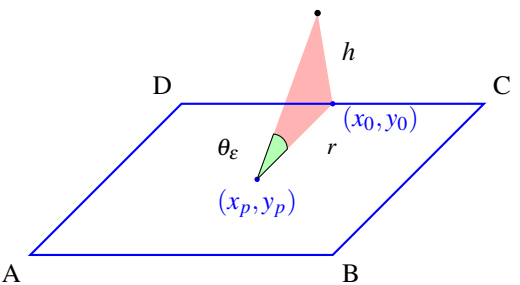


Figure 4.5 Current outlier filter strategy for the sweep measurements.

The Mahalanobis distance is a test statistic commonly used for outlier detection and is defined as

$$M_k^2 = (z_k - \bar{z}_k)^T P_k^{-1} (z_k - \bar{z}_k) \tag{4.6}$$

where z_k is a point, $\bar{z}_k$ the mean of a gaussian distribution and P is a covariance matrix [Jiang and Zhang, 2018]. It is often used in multivariate statistic to find multivariate outliers. The previously chosen residual generator, the normalized innovation squared, has many similarities to the Mahalanobis distance, especially in combination with the EKF. Both the Mahalanobis distance and the normalized squared are χ^2 distributed with $\dim(z_k) = n_z$ degrees of freedom [Dingler, 2022][Jiang and Zhang, 2018]. In the scalar case, this corresponds to a χ_1^2 distribution. An illustration of how the sweep NIS measurements are distributed can be seen in Figure 4.7a, where samples were collected between random points meant to be representative of a varied trajectory and common use case illustrated in Figure 4.6.

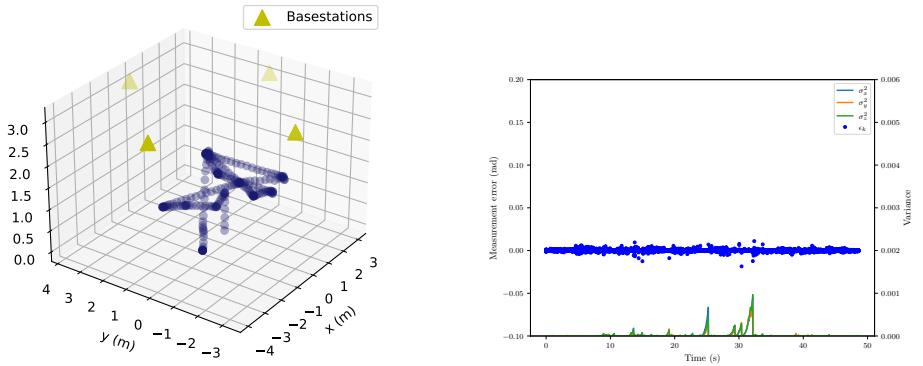
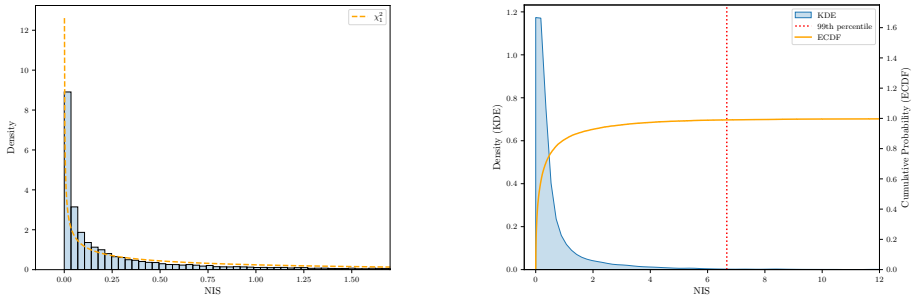


Figure 4.6 Path, error and position variance during flight with good line of sight. Axis scaled to match that of Figure 4.2 for comparison.

As the normalized innovation squared metric provides some interesting characteristics as described in the previous section, it is to be compared to the current outlier implementation. Illustrated in Figure 4.7a is the normalized innovation squared from sweep measurements. As anticipated, it follows a χ_1^2 distribution well. This is, however, a representation of the behaviour during normal conditions and as discussed in the previous section and illustrated in Figure 4.1, this is not the case for all trajectories and positions. Therefore, sweep samples were collected from three runs where the Crazyflie was tracking along the border of the flying area. These measurements were then post processed by removing samples deemed outliers, Figure 4.8.

The distribution of the collected samples was then analyzed using a kernel density estimation. A kernel density estimation (or KDE) smooths the observations with a Gaussian kernel, which produces a continuous density estimate [Waskom,



(a) **(b)**

Figure 4.7 Predicted and empirically derived statistical distributions of the NIS measurements. Histogram, NIS measurements (**a**) and kernel density estimation (KDE), empirical cumulative distribution function (ECDF) and 99:th percentile (**b**)

n.d.] Along the estimated density function, the ECDF was calculated and the corresponding 99:th, 98:th, 98.5:th and 97:th percentiles, which are thought to be used as thresholds, were calculated and are highlighted in Figure 4.9.

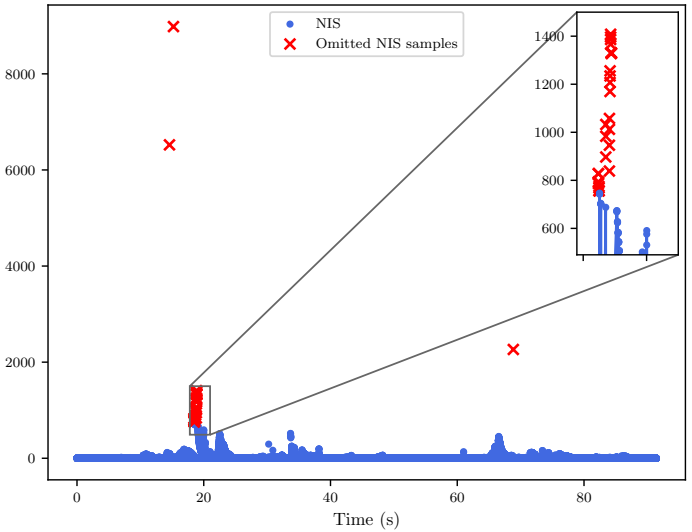


Figure 4.8 NIS samples from three flights for distribution analysis. Samples deemed outliers are highlighted in red.

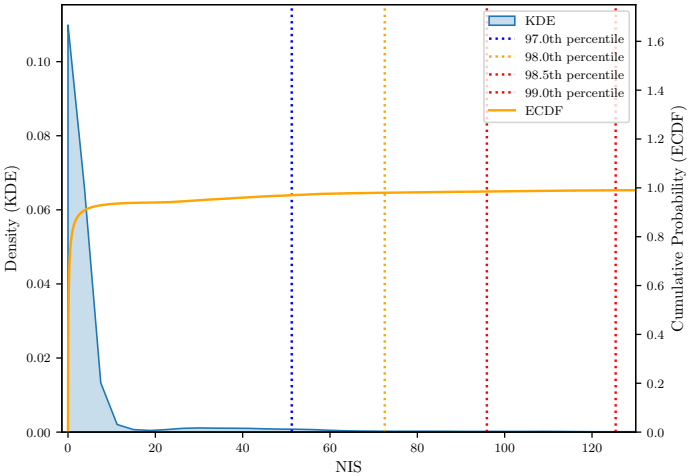


Figure 4.9 KDE, ECDF & Percentiles from NIS measurements in Figure 4.8

4.3 Emergency control strategies

In this section, different explored methods for bringing the Crazyflie to a landed state during an emergency situation is described. Since the Crazyflie is mainly a research and teaching tool, users often deploy their own control strategies and target trajectories, passive fault tolerant control is therefore ruled out as a viable option. In addition, in the case of a fault, it is not of critical importance to continue to be able to control the UAV to a certain destination or objective. In the presence of an error, the UAV should preferably not cause any harm to itself or its environment, and to its greatest ability neutralize itself. Therefore, an active control strategy where complete control is taken over from the user is developed. Emphasis has been put on the scenario where all external measurements have been lost and common across all strategies developed here is the choice to split the strategy into two distinct phases, similar to that of [Mueller and D'Andrea, 2012]. As a first action whenever a fault has been detected, the first phase of the strategy tries to get the Crazyflie to a stable hover. Only leveling out, that is, setting ϕ, θ to zero using the complementary filter, will make the Crazyflie continue its path as any directional velocity is carried along with it. It is therefore not sufficient if a stable hover with zero velocity along all axes is desired. In the second phase of the developed strategy, a descent is initialized where the strategy tries to get the Crazyflie to a landed state in a safe manner. These phases along with its transitions have been incorporated into the existing state machine of the Crazyflie and the intricacies and design choices are further elaborated on in the following sections.

Acceleration based PID control. As a first strategy, acceleration based PID control was theorized to be a viable option. The idea is to control the quadcopter by making use of its onboard accelerometer. During an emergency situation, depending on what error has occurred, external or internal, it can be argued that it would be wise to not rely on any external measurements or state estimations and only make use of the resources onboard. The accelerometer is one of the few onboard sensors, rendering it available at all times, no matter the situation, which could be valuable if all other states are lost or deemed unreliable. The signal from the accelerometer is, however, very noisy and contains high frequency components. The signals from the accelerometer are therefore filtered through the structure described in (2.5) with a cutoff of 1.3Hz and a sample frequency of 1000 Hz. This heavy filtering smoothens out any measurement spikes at the cost of being rather slow.

An overview of the control layout is illustrated in Figure 4.10 and the three PID controllers are implemented through the discrete PID equations (2.2). The levelout and descent control logic is as follows. Whenever a fault has been detected, the latest state is saved and the following is evaluated. In the initial transition to the levelout state, the maximum velocity along any of the axes is determined. This velocity then acts as the base in which the other velocities must adhere to. From empirical testing, the maximum achievable acceleration was determined to be 3 m/s^2. Whichever

velocity along any axis is greater thus determines the maximum acceleration during levelout. This acceleration is from here on now denoted a_{max}. The levelout timeout, that is the duration in which the subsequent accelerations should be applied, is then calculated according to (4.8). From this, the acceleration setpoints, including directions are calculated, (4.9). The principle idea in the 2-dimensional case is illustrated in Figure 4.11. To increase performance of the controller, an additional feedforward term to the x-y tracking is added as a static gain of 0.2. Additionally, a feedforward term as in the velocity controller, (2.4), is added as a base to the Z PID controller to increase performance and counteract the effects of gravity. All setpoints are calculated in the inertial frame and subsequently have to be rotated to the body frame using (2.3).

$$v_{max} = \max(v_x, v_y, v_z) \tag{4.7}$$

$$t_{levelout} = \left| \frac{v_{max}}{a_{max}} \right| \tag{4.8}$$

$$a = \frac{v}{-t_{levelout}} \tag{4.9}$$

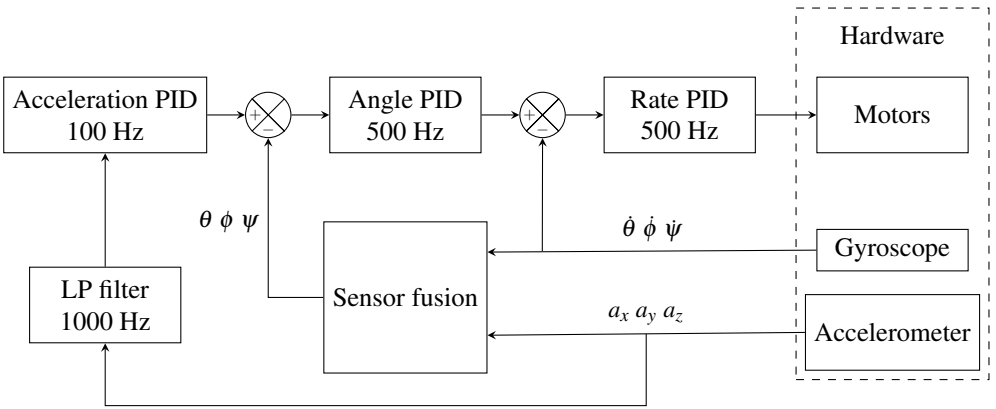


Figure 4.10 Cascaded acceleration PID control structure.

Once the levelout timeout is triggered, the state machine transitions to the descent state. During the transition, the descent duration is calculated using the snapshot saved from the previous transition. In the initialization of the descent state the Crazyflie is assumed to have reached zero speed along all its axes. A negative acceleration of 1 m/s^2 is then applied for 1 second to, in an ideal world, make the Crazyflie descend with a velocity of 1 m/s. The distances traveled during this deceleration period is calculated through the simple equation of motion, (4.10), where, as previously mentioned, v_0 is assumed to be zero.

$$s = v_0 t + \frac{1}{2} a t^2 \tag{4.10}$$

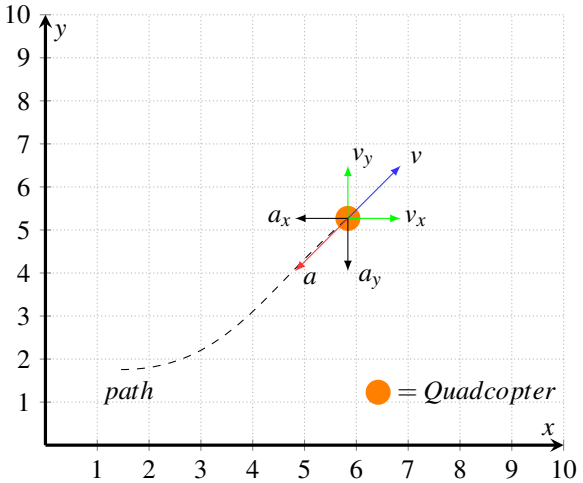


Figure 4.11 Principle sketch illustrating how the acceleration components are applied to counteract velocity and bring the quadcopter to a standstill.

This distance is then subtracted from the state snapshot and the time to descend the remaining height is calculated by $t = s/v$.

Strategy divison. The strategy of using the accelerometer is divided into two different strategies where the levelout phases are both identical. The descent phase is, however, divided into two different ones. Since it is assumed that zero velocity have been reached along all axis at the end of the first phase, using *modeDisable* and commanding roll and pitch to be zero would not cause the Crazyflie to drift away. Therefore in the first variant, the control setpoint is switched to keep θ, ϕ equal to zero using the complementary filter and in the second variant, all accelerations setpoints are instead set to zero.

Kalman velocity & position predict. As a second strategy, simply continuing to control the Crazyflie based on the states estimated by the Kalman filter in the predict step, (2.13)-(2.14) is explored. The filter will diverge from the true states but given a short enough timeframe the estimates might be good enough to be able to make a landing. The strategy is thus to levelout using the predicted states by the Kalman filter, and commanding zero velocity along all three axes. As a simple rule, the levelout timeout, that is the time the Crazyflie is given to reach zero velocity before transitioning to the descent state, is simply calculated by a linear function of the velocity magnitude (4.11), capped at 2 seconds.

$$t_{levelout} = \max(\frac{1}{2}\|\mathbf{v}\| + 0.3, 2) \tag{4.11}$$

The descent timeout is once again calculated after transitioning by taking the altitude state snapshot divided by the maximum allowed PID Z velocity. Once again, the Crazyflie is assumed to have reached zero velocity along all its axes at the same spot as where the fault was detected.

Kalman velocity levelout & roll, pitch hold with reduced thrust. Given a short enough levelout phase and quick detection from measurement outage, it can, as previously argued, be valid to assume that the Kalman filter would not diverge to much from the true states. The first phase of this strategy therefore uses the same levelout technique as previously mentioned. In the second phase with the assumption that the velocity along all axes are zero, the estimator is switched to the complementary filter and roll and pitch is commanded to zero. Descent is initialized by setting u_z in (2.4) to zero and the feedforward term to a value which causes the Crazyflie to descend. The descent timeout is then set to a fixed value.

4.4 Fault action and design

As a complete system design, all developed safety features are gathered in a separate module. The module runs at a fixed frequency of 20Hz or whenever a scalar update or velocity measurement has been received. It itself, communicates with the supervisor via a queue, where all relevant setpoint modes and state transitions have been calculated. The module is designed to supervise, monitor and react to the following errors:

- Run test error
- Scalar update frequency error
- OW error
- User defined error

Run test error. As discussed in Section 4.2, the module monitors against repeated CUSUM violations through a run test. All parameters and thresholds are stored in the deck's drivers and are loaded at startup. The innovations and innovation covariance enter the module through a queue directly from the EKF scalar update. The latest velocity is shared with the task at each Kalman predict step and utilizes the same queue.

Scalar update frequency error. Drift and abrupt sensor fault is expected to be captured by the previous method, measurement losses of a sensor will, however, not be naturally captured by the fault detection as is. These faults typically arise from bad solders, a bad pin connection, low battery level or because the line of sight from the receiving sensors is obstructed. Therefore a simple tick monitor, which supervises the period of the latest update from every scalar is implemented. Whenever a scalar update arrives, the monitor is updated with the current system tick. As the module runs periodically, any outage is detected by comparing the current system tick with the latest update. The period of which a scalar frequency error is defined

as was chosen as a combination of the period of what the Kalman predict step was still able to accurately capture the dynamics and from the longest recorded measurement outage in which the Crazyflie regained measurements and successfully regained control. This was recorded as approximately 3 s and is highlighted in red between $t \approx 9$ s and $t \approx 12$ s in Figure 4.2. As this was recorded during a hover at a spot with a bad line of sight, the maximum allowed measurement blackout period was reduced to account for potential velocities in real world use. The final period was set to 1.1 s.

OW error. As all modules adopt the one-wire protocol, they have a dedicated pin to allow the Crazyflie to know which modules are connected to it at startup. The module checks that the connected modules still equal the number found at system startup. The idea is to discover if the pins have been physically dislodged, which is then indicative of a potential more fatal error. This situation can occur when the battery swells or if the Crazyflie has crashed.

User-defined error. The module additionally checks an optional user-defined function for each connected board. The idea is to let the user define how an error or erroneous situation would be defined, and let the fault detection module handle it in a generic way. For example, users might have developed their own breakout modules and want to create and define their own fault conditions. This is of particular interest for modules that do not feed or are involved with the EKF. As deck drivers are usually implemented as periodic tasks, the easiest mechanism would be to let the task feed a watchdog which is then continuously checked for a timeout, but the criterion could also vary depending on the module. For example could measurements that are not part of the state estimation be thresholded or continuously checked. This could, for example, be if gas, temperature or live video feed is measured by a board.

System interaction

All mechanisms previously mentioned and their interaction are captured in Figure-4.12.

Fault action. To keep user customizability high, an action for each type of error can be specified. For example could one user simply want to freefall whenever a fault has been detected, whereas another might want to try and descend with available resources. For this there are several default actions available. The actions are ordered in level of severity so that if multiple boards with different error actions fail, the action with the highest severity level gets executed. The defined default actions available include:

- No Action
- Alert
- Levelout
- Levelout & (Descent or Transition)
- Descent
- Freefall

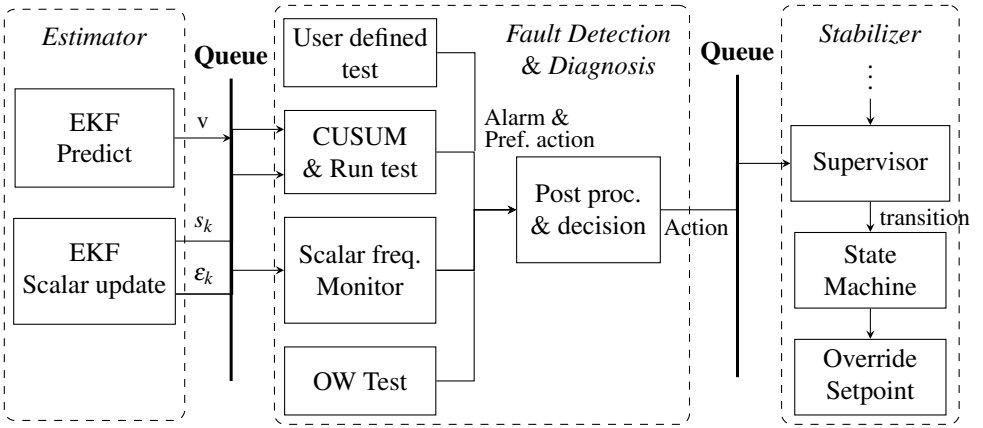


Figure 4.12 Illustration of fault detection and diagnosis task interaction with the EKF and complete test coverage with internal communication and decision making. Complete action and transition information sent to the supervisor via a queue.

If *No action* is specified, the detected error is ignored whereas an *alert* gives a visual LED fault pattern. A *freefall* triggers a system reset that will make the Crazyflie fall out of the sky and a *descent* tries to land the Crazyflie based on the fault type. A *levelout* only initializes the first phase in the emergency control strategy and tries to keep it at a hover. Control is given back to the commander after a *no error* is reported by the module but no earlier than 500 ms. The delay is applied to make the Crazyflie not transition rapidly between setpoints, potentially making it unstable.

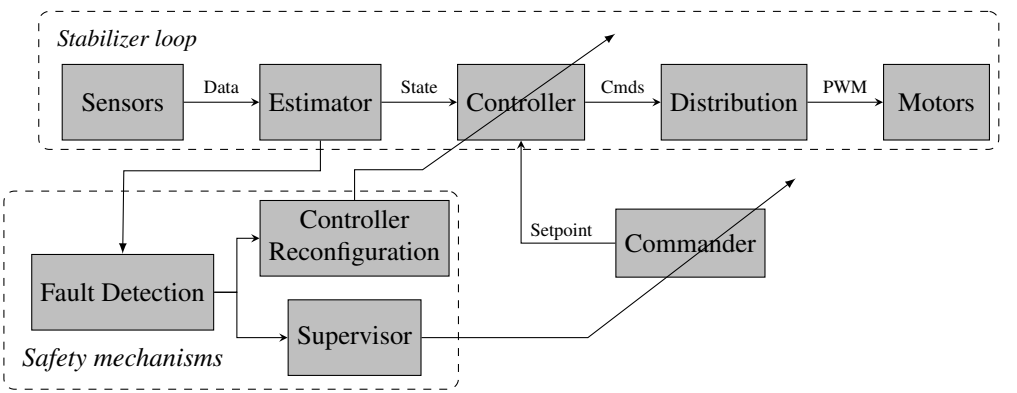


Figure 4.13 Fault detection module layout and interaction with the stabilizer, commander and supervisor. Crossed arrow illustrates that the crossing block has full right to override the output of the crossed block.

A *levelout & (Descent or Transition)* action initializes the first phase in the emergency control strategy and transitions either back to the *flying* state, if the module reports a *no error*, or to the descent phase if the error has not disappeared within a time period. The *levelout & (Descent or Transition)* time period is longer than the levelout timeout calculated in the *Descent* action, which is designed to bring the Crazyflie to a landed state as fast as possible. The decision and execution flow is all calculated in the fault detection module and is passed to the supervisor that handles the interaction with the state machine. The transitions are added as an extension of the onboard state machine and is illustrated in Figure 4.14. The added states are highlighted in red and the existing transitions with blue. If a tumble is detected at any point during the diagnosis or recovery phase, the Crazyflie transitions to the freefall state to avoid any potential dangerous situation. The system interaction with the stabilizer loop and commander is illustrated in Figure 4.13.

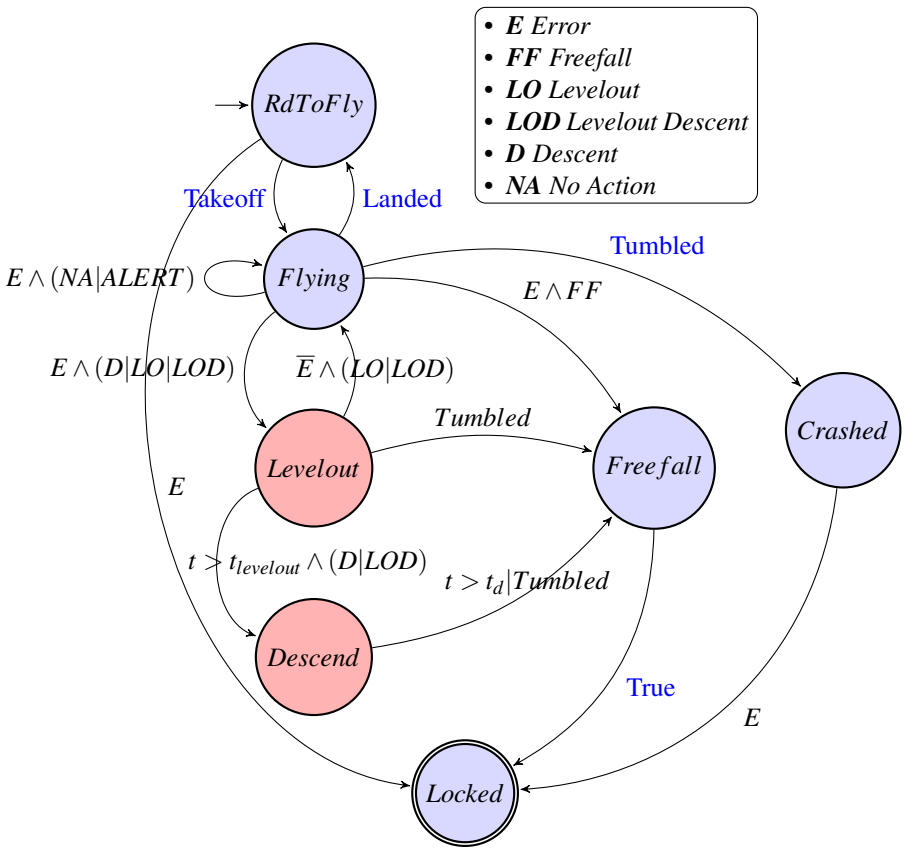


Figure 4.14 Subpart of the onboard state machine illustrating the states and transitions added in this thesis.

4.5 Propeller unbalance estimator

In this section, a vibration based propeller unbalance estimator developed in the works of [Ghalamchi et al., 2020] is described. The estimator is thought to provide a reliable health monitoring system to the Crazyflie and could act as an autonomous service indicator if the Crazyflie is flying unsupervised for extended periods of time. The method makes use of the accelerometer output and through correlation with the commanded thrust, the magnitude of an unbalance in any of the propellers can be detected. Presented below is a brief overview of how the estimator works and for a full explanation, the reader is referred to the original paper.

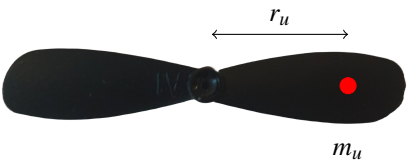


Figure 4.15 Propeller unbalance visualized in red.

The method presented assumes that an unbalance of mass $m_{u,i}$ at a distance $r_{u,i}$ from the center of rotation on propeller i is present as illustrated in Figure 4.15. This unbalance then causes a radial force on the shaft which propagates to the body of the quadcopter. This force can be expressed in the body-fixed coordinate system as

$$f_{u,i} = m_{u,i} r_{u,i} \omega_i^2 \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \\ 0 \end{bmatrix} = \frac{m_{u,i} r_{u,i}}{k_f} f_i \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \\ 0 \end{bmatrix} \tag{4.12}$$

where ω_i is the angular speed of propeller i, θ_i the corresponding angle of the unbalance with respect to y_b and (3.7) has been used in the second relation. The accelerometer, which is fixed to the quadcopter frame, measures the acceleration in the body frame and the measured acceleration can be related to the forces acting on the vehicle according to

$$\mathbf{a} = \frac{1}{m_b}\left(\sum_i (f_{u,i} + f_i e_{z_b}) + f_d\right) + v_a \tag{4.13}$$

where v_a captures the accelerometers additive noise, f_d are external disturbances and m_b is the quadcopter's total mass. Isolating the accelerometer ouput in the plane of the propeller gives

$$a_x = \sum_i \left(\frac{m_{u,i} r_{u,i}}{m_b k_{f,i}} f_i \cos\theta_i\right) + \frac{1}{m_b} f_{d,x} + v_{a,x} \tag{4.14}$$

$$a_y = \sum_i \left(\frac{m_{u,i} r_{u,i}}{m_b k_{f,i}} f_i \sin\theta_i\right) + \frac{1}{m_b} f_{d,y} + v_{a,y} \tag{4.15}$$

For compactness, the additive disturbance and mass unbalance ratio can be grouped together according to

$$d_x = \frac{1}{m_b} f_{d,x} + v_{a,x} \tag{4.16}$$

$$\Delta_i = \frac{m_{u,i} r_{u,i}}{m_b K_{f,i}} \tag{4.17}$$

$$a_x = \sum_i \Delta_i f_i \cos\theta_i + d_x \tag{4.18}$$

where the relation for a_y follows similarly. The goal of the estimator is to estimate the unbalance $m_{u,i} r_{u,i}$ without exact knowledge on the angle θ_i or the angular speed ω_i. The estimator does this by looking at the signal power in the accelerometer output, which will spike during maneuvers requiring roll or pitch since the unbalance is expected to cause the propeller to generate less thrust than its nominal counterpart. The scalar measurement at time k is thus defined as

$$y(k) = a_x(k)^2 + a_y(k)^2 \tag{4.19}$$

where the expected value is defined as

$$E[y(k)] = \sum_i \Delta_i^2 f_i(k)^2 + E[d_x^2] + E[d_y^2] \tag{4.20}$$

when utilizing the compact definitions of (4.16), (4.17) and (4.18). The unbalances are then estimated through an $(N_p + 1)$-state EKF, where N_p is the number of propellers on the quadcopter. The unbalances are encoded in a exponential relationship to ensure that the quantities stays positive. The first N_p states x_i relate to the unbalances as

$$e^{x_i} = \Delta_0^{-1} \frac{m_{u,i} r_{u,i}}{m_b K_f} \tag{4.21}$$

and the last to the additive noise according to

$$e^{x_{N_p+1}} = \sigma_{d,0}^{-1} \sqrt{E[d_x^2]} \tag{4.22}$$

where Δ_o is an initial guess for all unbalances and $\sigma_{d,0}$ an initial guess for the standard deviation of the additive noise. The measurement model is

$$y(k) = \sum_{i=1}^{N_p} f_i(k)^2 e^{2x_i} \Delta_0^2 + 2e^{2x_{N_p+1}} \sigma_{d,0} \tag{4.23}$$

and its jacobian defined as in (2.21) becomes

$$H_i = 2 f_i(k)^2 e^{2\hat{x}_i(k)} \Delta_0^2 \text{ for } i \in 1, \ldots, N_p \tag{4.24}$$

$$H_{N_p+1} = 4 e^{2\hat{x}_{N_p+1}(k)} \sigma_{d,0}^2 \tag{4.25}$$

The estimator then follows the standard EKF formulation presented in Algorithm 1, where the commanded force is calculated for each motor by utilizing the PWM to force relation found in (3.13) and the system dynamics are modeled as being driven by zero-mean white independent noise. The forward propagation of the covariance matrix in the predict step thus takes the form according to

$$P_p(k) = P(k) + \text{diag}(q_\Delta^2, \ldots, q_\Delta^2, q_d^2)\Delta t \tag{4.26}$$

5

Results

5.1 Fault detection

To assess the performance of the chosen fault detection strategy, experiments in two different scenarios were carried out. As shown in Figure 4.3, the NIS natural increases during complex maneuvers and to increase sensitivity, a velocity variable threshold was developed. Therefore, the experiments involve tracking a trajectory with a default speed profile and a maximum velocity profile. The profiles along with its velocities during the trajectory are illustrated in Figure 5.1 where the lowpassed filtered velocity that acts as the gain in the linear function threshold is illustrated together with the true velocity. When the Crazyflie crosses $y = 1.7$ m in the trajectory,

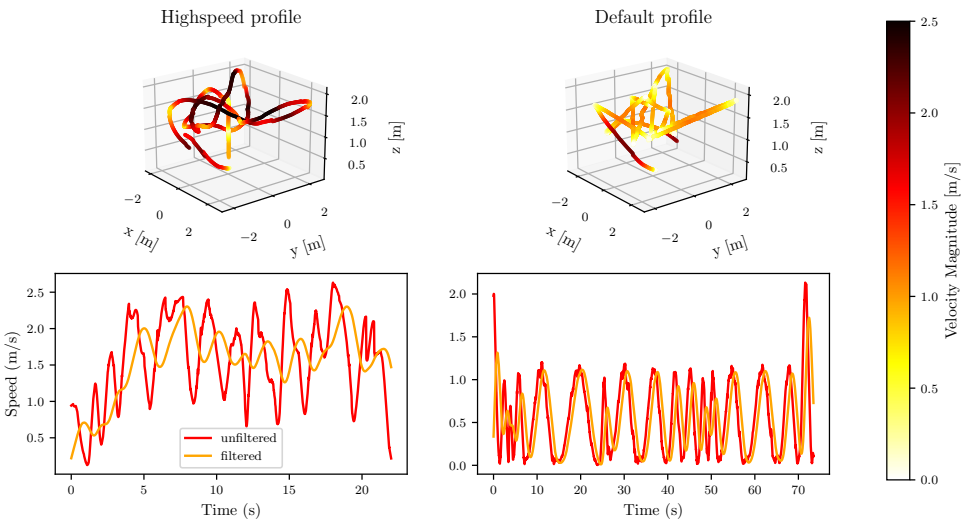


Figure 5.1 Experiment trajectory, velocity profiles and velocities.

a zero mean white gaussian noise is injected in the angle measurement to simulate a disturbance. The noise was injected outside the outlier filter and before the positioning and heading estimation. The injected noise level corresponded to $4, 6$ and 8 times the standard deviation of what the sweep measurement model is tuned for. The experiment was carried out five times for each noise level and velocity profile. A summary is found in Table 5.1, where the average of the experiments is found. Illustrated in Figure 5.2 and Figure 5.3 is what the sweep and heading CUSUM looks like for a common run for each noise level. Worth noting is that the same noise was injected for all repeated disturbance levels.

The average incoming velocity, that is the velocity at disturbance injection, for the default and high velocity profiles were 0.7 and 2.4 m/s, respectively. The run test for the sweep measurement required two consecutive violations within a timespan of 150 ms to be considered a true positive and three violations within a timespan of 250 ms for the heading measurement, owing to the lower frequency and higher magnitude outliers. As performance metrics, the average time to detection and the average number of violations for the sweep measurement before a run test was triggered is included. For the heading CUSUM, only the average number of violations is included since this did not trigger a run test on any of the experiments.

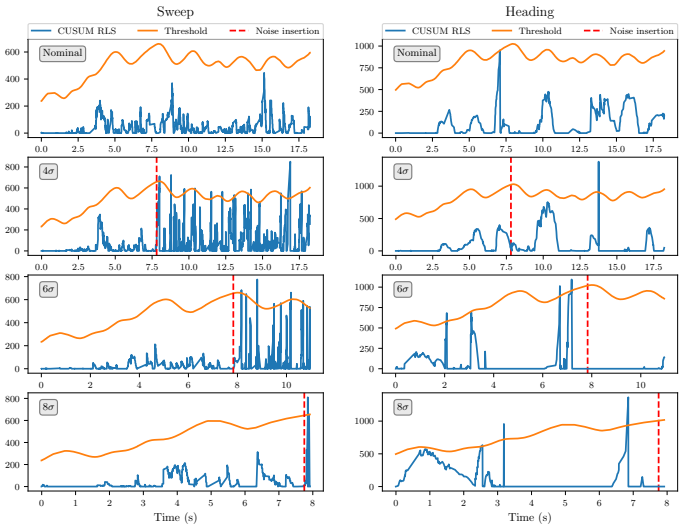


Figure 5.2 Showcasing what a common Cusum RLS looks like for the different disturbance levels during the high velocity profile. Left and right column showcases the sweep and heading residuals respectively. Increased noise level in the measurement decreases time to detection for the sweep Cusum while the heading measurement model is unaffected.

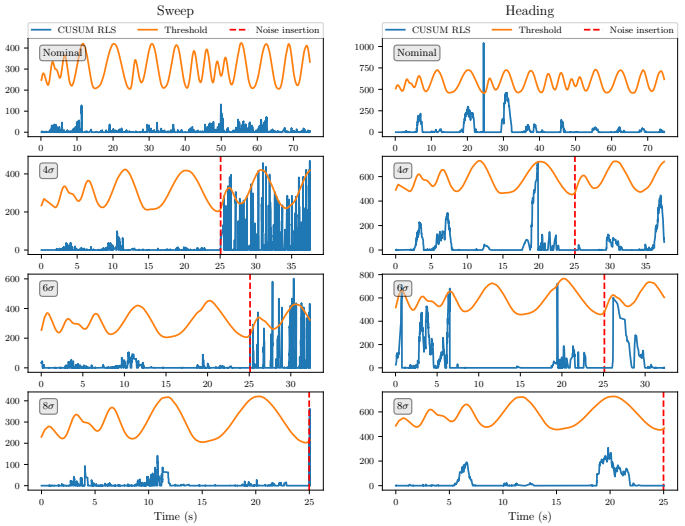


Figure 5.3 Showcasing what a common Cusum RLS looks like for the different disturbance levels during the low velocity profile. Increased noise level in the measurement decreases time to detection for the sweep Cusum while the heading measurement model is unaffected by the injected noise and subject to false alarms sporadically.

Table 5.1 Fault detection summary across five runs of each disturbance level

Noise scenario	Velocity profile	Avg. N violations to detection Sweep	Avg. N violations Heading	Run test triggers [x/5]	Avg. Time to detection [s]
Nominal	Default	-	2.2	0	-
	Highspeed	-	3.0	0	-
$\sigma_n = 4\sigma_R$	Default	6.8	2.0	5	4.80
	Highspeed	8.0	1.2	2	6.10
$\sigma_n = 6\sigma_R$	Default	4.8	3.5	5	1.73
	Highspeed	3.4	2.0	5	0.82
$\sigma_n = 8\sigma_R$	Default	3.8	1.0	5	0.82
	Highspeed	2.8	2.0	5	0.39

5.2 Emergency strategies

To benchmark the performance of the different strategies discussed in Section 4.3, the Crazyflie was commanded to fly in a circle with a radius of 1.4 m at an altitude of 1.8 m for three different velocities. At time t_0, all measurements to the Kalman filter are rejected, the safety module then detects the measurement outage and initializes the emergency control strategy.

Ground truth. As ground truth, the crossing beam method is used. The crossing beam method is an alternative method for positioning and directly calculates the position of the Crazyflie using the intersection between two or more basestations. In the investigative work of [Taffanel et al., 2021] the crossing beam method was concluded to be a viable option to be used as ground truth and the performance is on par with the EKF during nominal conditions. The crossing beam method does, however, need direct line of sight from at least two basestations, and thus for aggressive maneuvers, the incoming measurement frequency is often reduced. Even during nominal conditions the measurements come in, although frequently, sporadically and at different periods. For this reason, the positioning data from the crossing beam method is interpolated in post processing to be able to be compared to the Kalman output at the same time stamps. A monotone cubic interpolation technique was used, which is constructed to be only once differentiable, in order to preserve the local shape, avoid oscillating and "overshooting" [Scipy, 2024]. For velocity "truth", the interpolated positions were simply integrated. This, of course, is a very rough estimate and should be taken with a large grain of salt, but is nonetheless interesting when compared to the output of the Kalman filter.

Experimental setup. For each emergency control strategy, the same experiment was repeated five times. As performance metrics, the average Kalman filtered and interpolated velocities are calculated as well as the average distance from measurement outage detection to final landed position. The latter performance metric is visualized in red in Figure 5.4. Additionally, all events, states and phases including a ground projection are color coordinated for illustrative purposes. Besides the compiled 3D plots, a plot highlighting how the Kalman filter, crossing beam and interpolated position and velocity states diverge and relate to one another during a common representative case is provided, where the levelout state as well as the descent phase is highlighted in red and orange, respectively. For the acceleration based strategies, a common response for each initial velocity is also illustrated. Finally, a summary of the performance metrics is presented in Table 5.3 as well as the final acceleration PID parameters in Table 5.2.

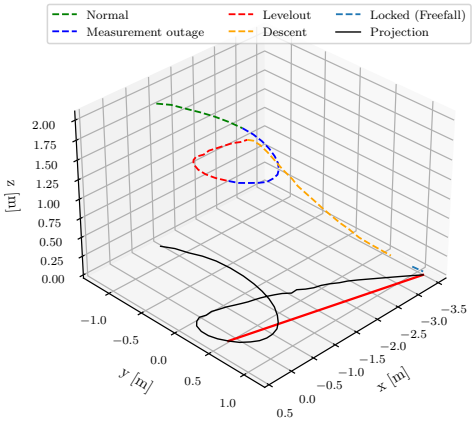


Figure 5.4 Illustration of performance metric used, highlighted in red, including color coordination of the different events and states throughout the experiments.

Table 5.2 Final PID controller parameters used in the acceleration based emergency control strategy described in Section 4.3.

Controller	K_p	K_i	K_d	Output cut-off freq. (Hz)
x	15.0	0.04	1.0	15.0
y	15.0	0.04	1.0	15.0
z	12.0	0.01	1.0	15.0

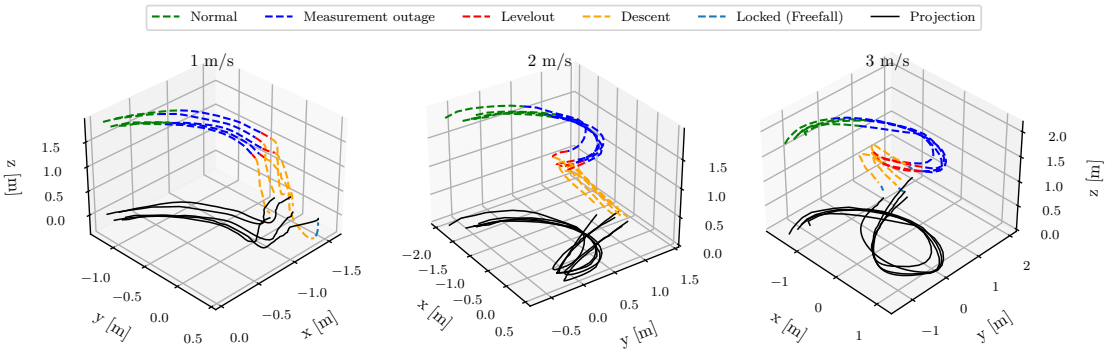


Figure 5.5 3D visualisation of the experiments using acceleration levelout and roll, pitch hold with acceleration descent.

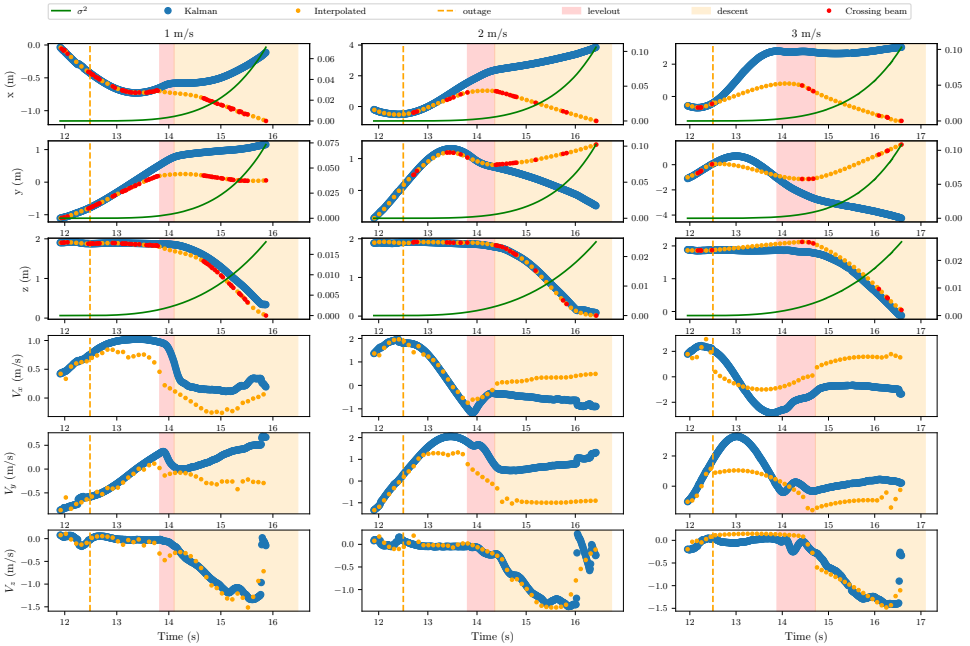


Figure 5.6 Illustration of how Kalman filtered and ground truth position and velocity diverge during outage for acceleration levelout and pitch, roll hold with acceleration descent. Each column represents the states for a given initial velocity.

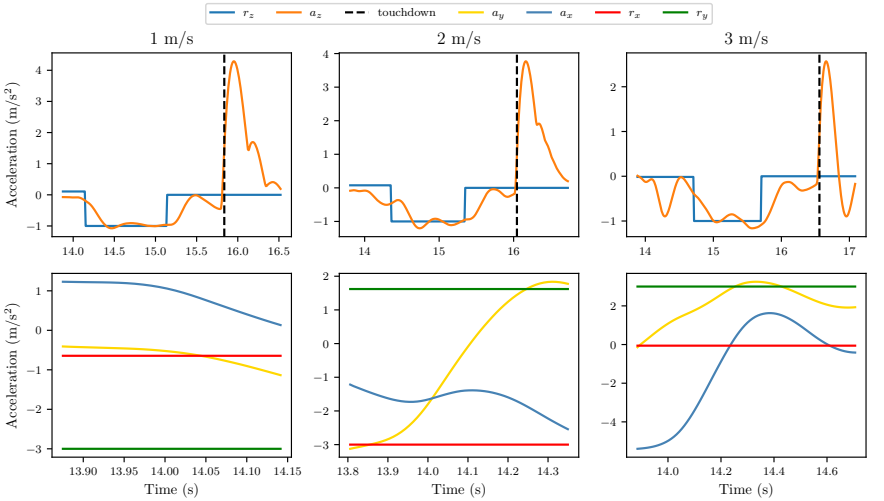


Figure 5.7 An illustration of what a common setpoint response looks like using the acceleration levelout and roll pitch with acceleration descent where each column illustrates a response for a given initial velocity. Scenario corresponding to the second row in Table 5.3.

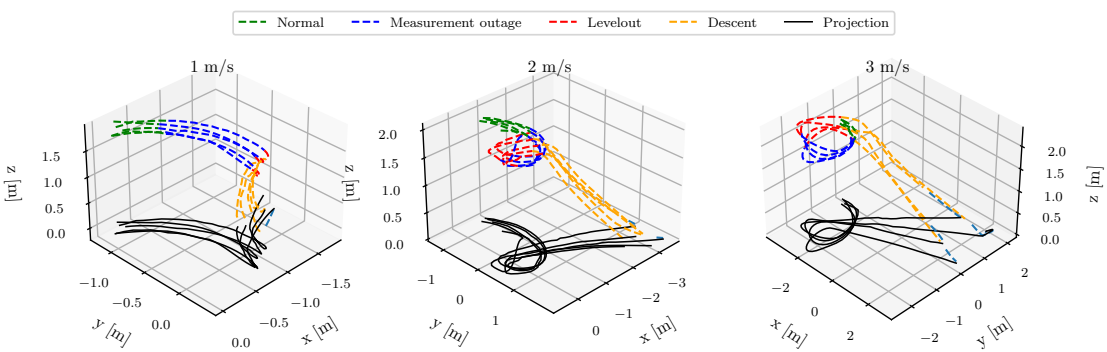


Figure 5.8 3D visualisation of the experiments using the Kalman filter estimates during levelout and descent.

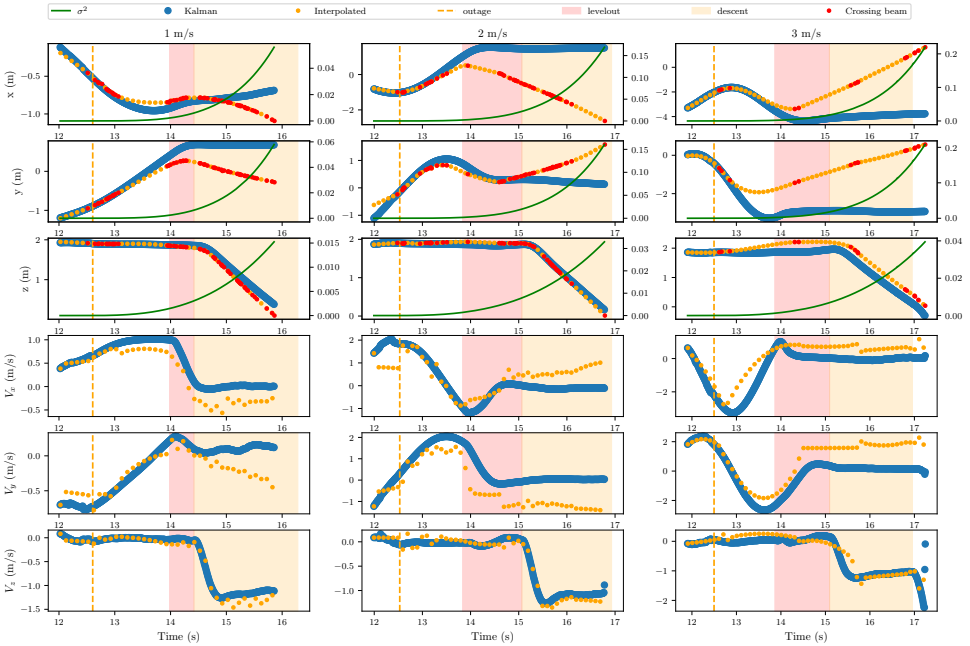


Figure 5.9 Illustration of how Kalman and ground truth position and velocity diverge during outage using the Kalman filter to descend. Each column represents the states for a given initial velocity. Controllers accurately track to zero Kalman velocity as expected.

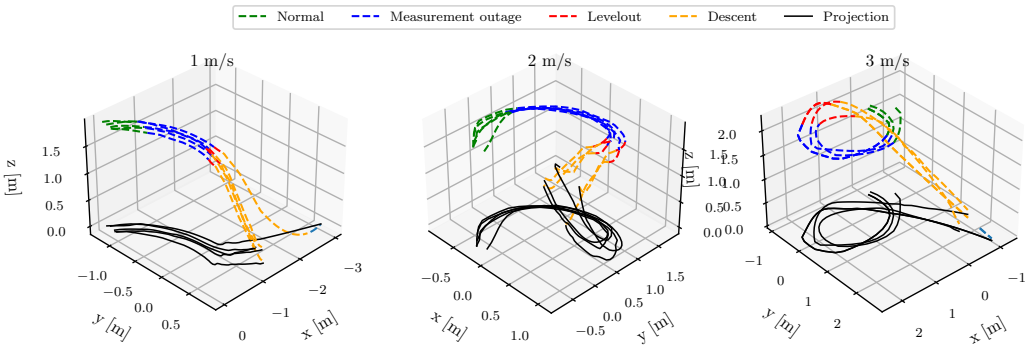


Figure 5.10 3D visualisation of the experiments using acceleration levelout and descent.

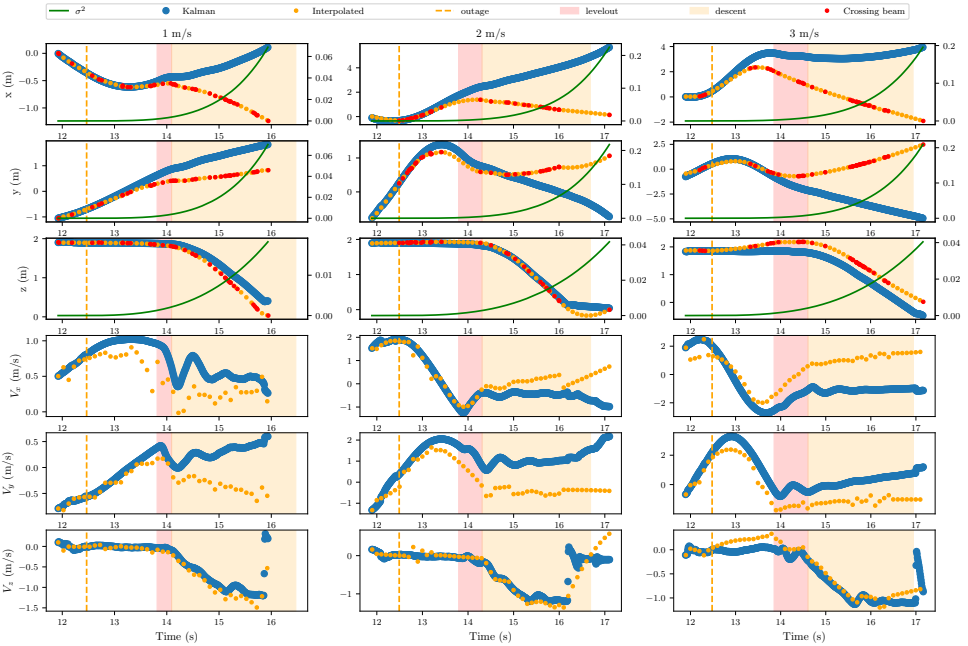


Figure 5.11 Illustration of how Kalman and ground truth position and velocity diverge during outage for acceleration levelout and descent. Each column represents the states for a given initial velocity. Slight improved performance of the "truth", interpolated velocity at the end of the levelout phase compared to using the Kalman estimates.

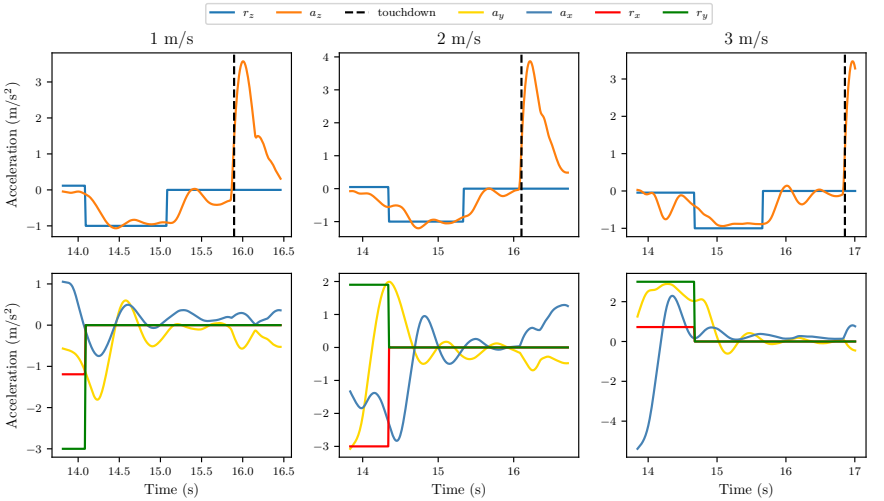


Figure 5.12 An illustration of what a common setpoint response looks like using the acceleration levelout and descent where each column illustrates a response for a given initial velocity. Scenario corresponding to the third row in Table 5.3.

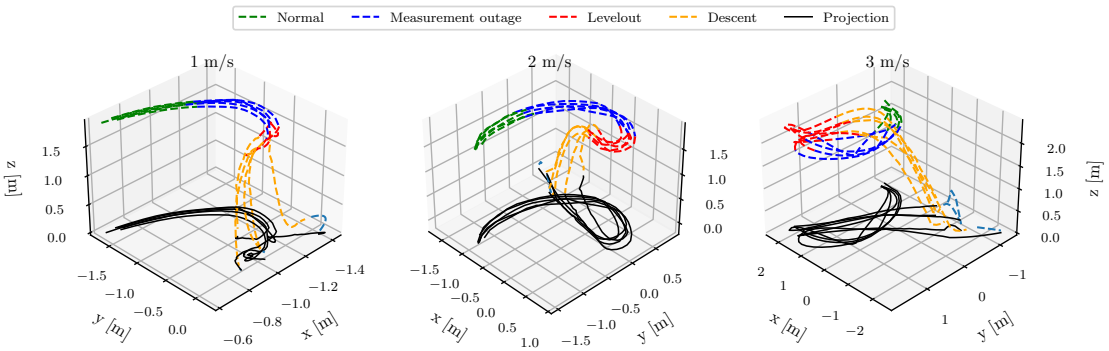


Figure 5.13 3D visualisation of the experiments using velocity levelout and roll pitch hold with PWM descent.

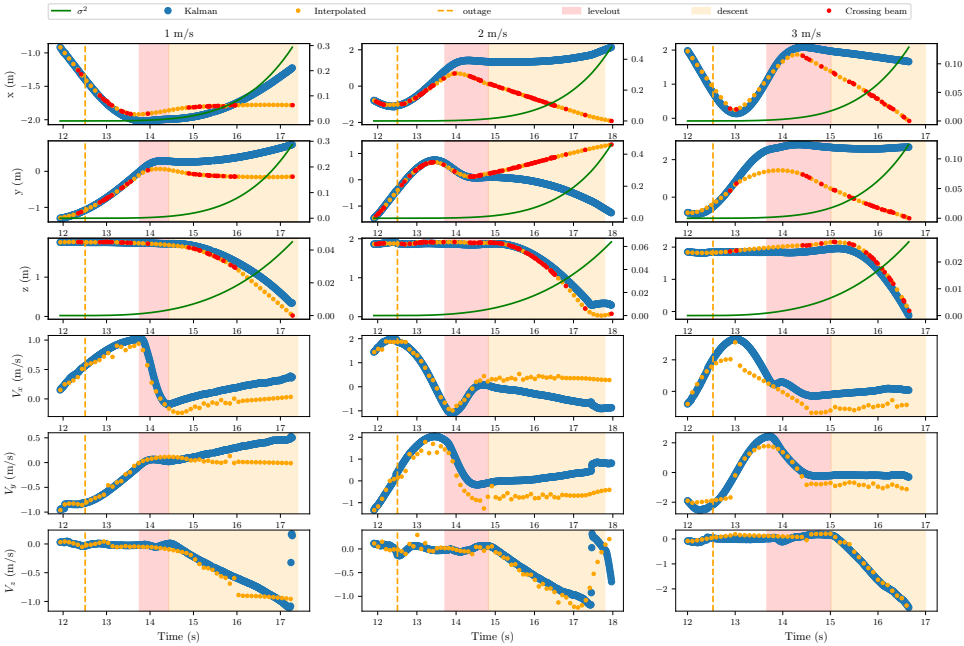


Figure 5.14 Illustration of how Kalman and ground truth position and velocity diverge during outage for velocity levelout and roll pitch hold with PWM descent. Each column represents the states for a given initial velocity.

Table 5.3 Landing strategy summary. Each strategy presented in Section 4.3 with the average Kalman and interpolated velocity, $\bar{v}_d$ for comparison as well as the average ground distance performance metric illustrated in Figure 5.4.

Strategy	Init. cond. [m/s]	$\bar{v}_d$ Kalman [m/s]	$\bar{v}_d$ Interp. [m/s]	Avg. ground distance from detection to crash [m]
x, y, z v. levelout,	1.0	0.14	0.39	0.82
x, y v. descent	2.0	0.09	1.33	3.58
z pos. descent	3.0	0.38	1.85	5.88
x, y, z acc. levelout	1.0	0.64	0.41	0.56
$\phi, \theta = 0$ descent	2.0	0.83	0.48	1.47
z acc. descent	3.0	1.10	0.86	2.58
x, y, z acc. levelout	1.0	0.62	0.46	1.03
x, y acc. $= 0$	2.0	1.05	0.57	1.37
z acc. descent	3.0	1.27	1.41	3.82
x, y, z v. levelout	1.0	0.11	0.19	0.31
$\phi, \theta = 0$ descent	2.0	0.13	0.86	1.73
z pwm. descent	3.0	0.34	1.49	3.50

5.3 Outlier filter

To asses the NIS outlier filter, four runs at different threshold levels corresponding to different percentiles were carried out. The trajectory passed several identified points where the line of sight to the basestations were poor and were meant to be representative of a normal use case. The filter ran online on the Crazyflie and samples classified as outliers were rejected from entering the Kalman filter. For comparison, the current outlier filter ran concurrently and classified outliers on the same samples. Measurements classified as outliers with the current implementation were allowed to enter the Kalman filter, except, of course, if they were classified as outliers by both filters. Figures 5.15-5.18 illustrates and compares the performance of the two strategies when running along the same trajectory. A summary of the results is presented in Table 5.4.

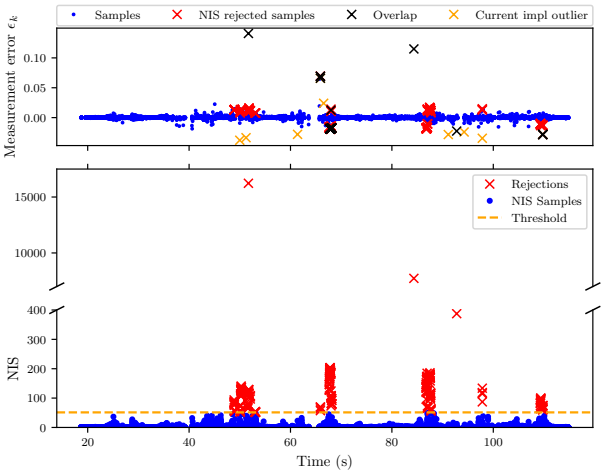


Figure 5.15 Rejection comparison for sweep samples between NIS and the current outlier filter, (4.5), at the 97:th percentile.

Table 5.4 NIS outlier filter summary, presenting the percentiles along with the scalar threshold, number of rejected samples with the respective outlier filter strategy, rejections as a percentage of the total samples and rejection overlaps.

Percentile	Threshold	No. Rejected NIS	% of tot. samples	Overlaps	No. Rejected Curr.
97th	51.3	238	0.993	31	45
98th	72.5	141	0.572	10	21
98.5th	96.0	33	0.137	2	7
99th	125.0	4	0.017	1	7

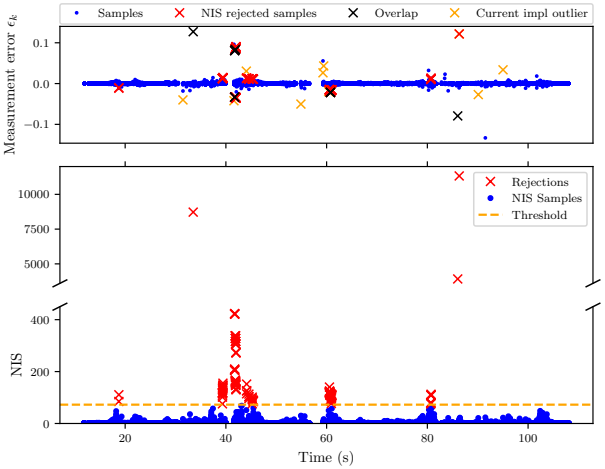


Figure 5.16 Rejection comparison for sweep samples between NIS and the current outlier filter, (4.5), at the 98:th percentile.

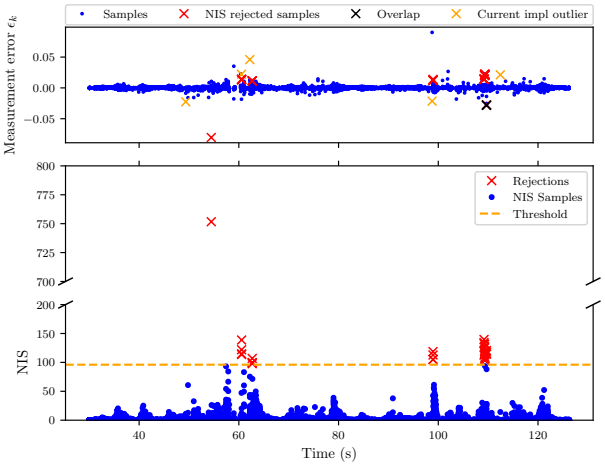


Figure 5.17 Rejection comparison for sweep samples between NIS and the current outlier filter, (4.5), at the 98.5:th percentile.

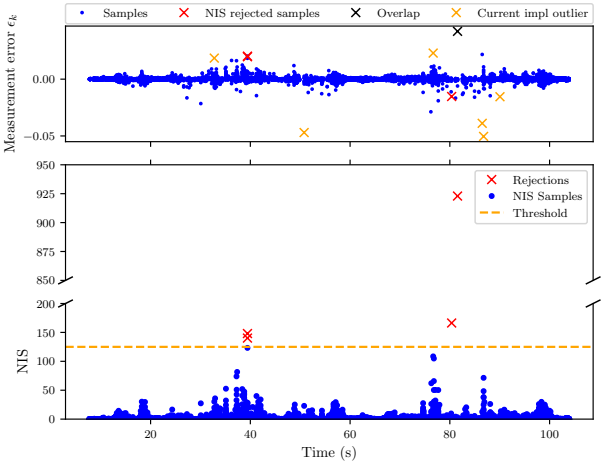
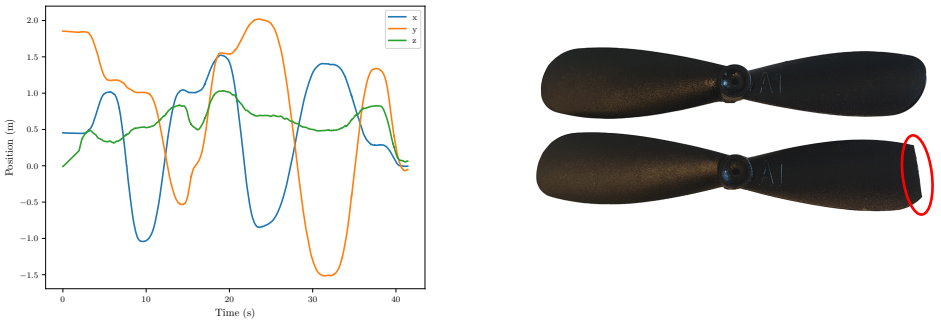


Figure 5.18 Rejection comparison for sweep samples between NIS and the current outlier filter, (4.5), at the 99:th percentile.

5.4 Propeller unbalance estimator

The estimator is demonstrated by tracking random points illustrated in Figure 5.19a for three different scenarios. In the first, all propellers are in nominal condition, whereas in the two latter, a section was cut off at the tip as illustrated in Figure 5.19b. The tip damage corresponded to an unbalance of approximately 15-20 mg. Two runs were then carried out where the damaged propeller was swapped in between the motors generating the forces f_2 and f_4 in Figure 3.3. An illustration of the three runs is presented in Figure 5.20 where the unbalances have decoded through (4.21) and the additive noise through (4.5).



(a) **(b)**

Figure 5.19 Points tracked (**a**) and damage to propeller (**b**)

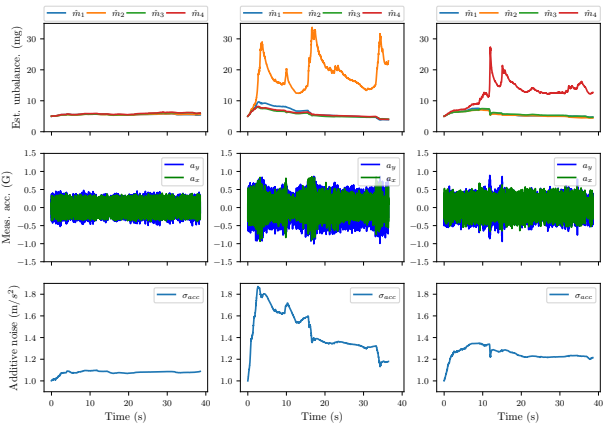


Figure 5.20 A summary of the states from the three runs and scenarios including accelerometer output. Top row illustrates the estimated unbalances decoded through (4.21) and the bottom row the estimated additive noise decoded through (4.5).

6

Discussion

6.1 Fault detection and trigger

The developed fault detection system is overall robust and works as intended. As can be seen in Figure 5.2, Figure 5.3 and Table 5.1, the mechanism does not trigger or cause any violation in the nominal case whereas several violations and eventually a run test is triggered for all disturbance levels. For the lower noise injection level, $4\sigma_R$, the rather slow average time to detection might seem like an odd result but is by design. For this noise level, the Crazyflie, although wobbly, is still able to carry out flight. Even though a disturbance is present, causing a descent in this scenario can be considered unnecessary, especially if the disturbance is temporary.

For the higher noise levels the time to detection significantly decreases. At these noise levels, a crash is from empirical testing almost always imminent if no fault detection system is active. The choice of the normalized innovation squared as the distance measure is sound and the hypothesized benefits are clear. An interesting finding is how the injected noise does not affect the heading estimation. For the high-speed trajectories, the average time to detection decreases compared to its low-speed counterpart for all disturbance levels. The fault detection mechanism developed is overall lightweight, easy to tune, maintain and is robust. If the system is desired to be less sensitive, increasing the run test and time window to three violations and 200 ms was from empirical testing deemed a good choice. Increased sensitivity can easily be achieved by simply increasing the time window in between run tests, as even during lower disturbances, violations are frequent as seen in either Figure 5.2 or Figure 5.3.

Fault action. Although not thoroughly investigated, the descent strategy used included switching to the complementary filter, holding pitch and roll while descending using a fixed PWM value. This approach proved to be an effective strategy although the Crazyflie tended to drift significantly while descending. An alternative would have been to use the acceleration based PID controller but since the disturbances would often cause the Crazyflie to wobble significantly, a more safe approach was to use the former.

6.2 Outlier filter

Comparing the previous outlier filter with the one developed in this thesis is a difficult task. At a first glance, the current outlier implementation more eagerly rejects measurements that are "visual" outliers, that is measurements with a large measurement error. This is of course not surprising as this is exactly what it is, a fixed threshold test on an height error. The performance of the NIS outlier filter seems to coincide with the current outlier implementation at a threshold level of 95.0, which, according to the collected data set represents the 98.5:th percentile. An obvious difference in the prioritization between the two is when the Kalman filter has repeatedly been feed with measurements with a low absolute error level. The filter covariance then becomes drastically lower, meaning the filter is more and more sure of its estimation. Receiving, not necessarily bad measurement, although with a slightly larger absolute error, thereafter, causes a lot of rejections at a low threshold, this is natural with the definition of the metric and is by design. This can for example be seen at approximately after $t = 84$ s in Figure 5.15. Another difference that highlights the advantage of using NIS is seen in Figure 5.16 at approximately $t = 60$ s where after an outage, the current implementation would have rejected the samples, whereas the NIS does not by definition of the metric.

Conclusion. The only conclusive assessment is to state that the two outlier filter are "different". Since they operate on different metrics, different prioritisation is given to different types of misalignment between model and measurement. The hypothesized benefits of the NIS outlier filter are hard to verify, but the result and performance seem to be at least on par to that of the current implementation, as the rejection overlaps are high when tuned to a suitable threshold. In the experiments a fixed threshold was used but as in the CUSUM test case, a velocity dependent threshold could be used to improve performance.

Recommendation. As a concluding remark and recommendation for the outlier filter using the Mahalonobis distance, it would not be wise to use it on the sweep measurements, as there is a high risk of filtering out too many measurements during noisy periods, which would severely if not completely render the CUSUM test inoperable. Using it on the heading measurement does, however, make sense as spurious outliers from empirical observation usually vary in the range 10-100 times that of the nominal condition and no outlier filter exists. Proving a performance increase is, however, difficult but the approach should prove useful.

6.3 Emergency strategies

Acceleration

The acceleration profile used to descend is not the fastest, it is although perhaps the easiest and safest one. One could think of a multitude of other profiles. The profile

which would bring the Crazyflie down to the ground with zero velocity would be the trapezoidal rule. One could for example also just keep on accelerating down to ground by applying a constant negative acceleration. The landing trigger could also have been extended to include a threshold check on the accelerometer readings in the positive Z-axis in addition to the calculated theoretical timeout. The success of this "bump" check would depend on the impact velocity, but as can be seen in Figures 5.7 and 5.12 the accelerometer reading at touchdown is significant. As a further discussion point, the assumption that levelout was achieved instantaneously when calculating the descent timeout does not seem to impact performance negatively. Overall the performance of the strategy seems promising and its major downside lies in its ability to accurately track setpoints. The measurements were heavily filtered, and the filter is rather slow, the controller also experiences oscillations and has a hard time settling on a steady state when switching between setpoints as seen in Figure 5.7. This was, however, a conscious performance trade off between rise time and overshoot and as meeting the desired accelerations as quick as possible was prioritised, this behaviour was expected and tolerable. Both strategies that only relied on the accelerometer and gyroscope performed significantly better than just using the Kalman filter for the two higher velocities when looking at the average interpolated velocity and distance metric. Switching to the complementary filter in the second phase performs better than continuing to control on acceleration in the x-y direction. This is reasonable for two reasons: Firstly, if the assumption that the Crazyflie has reached zero velocity after the first phase is true, then commanding a roll and pitch hold for the second will not cause any major drift. Secondly, since the controller gains in the x-y direction are quite aggressive, the overshoots will cause unintentional velocities when switching setpoint in the descent phase.

Kalman filter based control

Continuing to control the Crazyflie by using the predicted states from the Kalman filter in the case of an outage gives the worst performance for the higher velocities, while being on par with the other approaches for lower velocity. As expected and can be seen in Table 5.3, the average Kalman filtered velocity is close to zero at the end of levelout. Looking at the interpolated velocities reveals another story though, but should be taken with a large grain of salt because of the integration method chosen. What is interesting is the ability of the estimated position and velocity to accurately track the true values during steady state operation, that is when no outage is detected and tracking the circle. Looking for example at an average for each velocity in Figure-5.9 (or any other of the common situations) reveals that until the detection system initialises the descent strategy the crossing beam method and the Kalman predictions align well. At the start of the first phase the positions clearly start to diverge, and for the second phase predictions are nowhere close to truth. The Kalman filter is essentially a double integrator with the accelerometer and gyroscope and during rapid maneuvers, such as with the levelout and descend phases,

the noise and drift of the accelerometer integrations clearly adds up and severely degrades the ability to descend while holding zero velocity in x-y. The Crazyflie is despite this divergence steadily descending and is not at risk of tumbling, only smashing into its environment.

Reduced thrust. Reducing the thrust to bring the vehicle to a landed state works well overall. It is perhaps the easiest strategy and a very fault tolerant method as it does not depend on any external measurements. The main flaw of this method is if the chosen base thrust unexpectedly generates an upward thrust instead. This can occur if the vehicle has been modified in any way, for example if the propellers have been swapped out, motors changed, altered or if a more powerful battery is mounted. The current state of charge of the battery also has an impact on the thrust produced. An easy way to combat this is to simply choose a base thrust value way below what upwards thrust any heavily modified Crazyflie would be able to achieve. This is hard to predict as there are many Crazyflies out on the consumer market and any such modification is hard to predict. Another downside would be the implications a severely reduced base thrust would have on any other non-modified version. A too low base thrust would cause every other Crazyflie to simply fall out of the sky or land with a potentially harmful velocity. The implemented descent timeout was a fixed value and an approach would have to try to predict the descent time required depending on the last altitude measurement or to stop based on a "bump check" on the Z accelerometer reading as seen in either Figure 5.7 or Figure 5.12 at touchdown.

Fault action and tests performed

Actions. The available fault actions and tests performed are all interesting options but their usefulness differ and can be subject to modification. When the user selects an action for a specific fault, consideration needs to be taken in regards of the capabilities. Choosing the levelout option for example in the case of an outage would cause the vehicle to hover until the measurements reappear again. If this is not the case, the Crazyflie will keep hovering only using the predict step which is not able to keep it at zero velocity indefinitely. The drift would then most probably make it hit its environment. Generally, it could be argued that if for whatever reason the desired reference is needed to be overridden, the "mission" is simply over and trying to re-track to the plan defeats its purpose. This is especially true for a small indoor research quadcopter as the Crazyflie. This is where the *Descent* action shines. The *levelout & (Descent or Transition)* action would be suitable in an autonomous situation with a long-term objective where sporadic erroneous behaviour is acceptable, as for example in a demo scenario. From experience, the most important thing in case of a measurement outage or other fault is to get to ground as fast as possible if it is the desired behaviour. There is really no need of a levelout state and the stabilization could be done while descending like what was done when performing the

noise injection experiments. The *alert* option currently only shows the LED fault pattern but could be extended to also send a status code to the user through the GUI.

Tests. The usefulness of the OW-test is arguably low. If the OW were to fail because of a connection loss while flying, something else has probably failed before it. It could, however, be useful after a crash where the UAV looks intact but the pins have been disconnected. Entering the locked state and preventing a takeoff would then be reasonable and useful to prevent a more dangerous situation. The success of the user defined test heavily depends on how the user defines it and thread safe mechanisms need to be considered. The module simply calls a *connected* function for each board connected, which needs to be defined by a user.

6.4 Propeller unbalance estimator

The unbalance estimator works as intended and no change to tuning parameters used in the original paper was needed. The estimator can accurately predict the unbalance to the correct order of magnitude and point out which propeller is problematic. Any effects of gravity or bias in the accelerometer was filtered out by a second order high-pass filter with a cutoff at 1 Hz. The high-pass filter was constructed by the structure in (2.5) and utilizing the fact that high-pass and low-pass filters are complements to each other. Different from the original implementation is the exclusion of a time delay from commanded thrust to output, which does not seem to negatively impact performance. It is easy to see the excitations that trigger the unbalance estimation and it is not impossible some information is lost in this implementation as the estimator runs at a fixed rate of 500Hz, while the accelerometer sends measurements at roughly 1 kHz. Increasing the run period of the estimator or running an accumulator and taking the average of the accelerometer output and commanded thrust might further increase performance, but given the convincing results this was not explored. The implementation of the estimator is model dependent as no measurement of the angular speed is available. In this approach the relation between commanded thrust and PWM is utilized and as there are many different propeller and motor variants out on the market, the approach is not suitable as a general service mechanism for the Crazyflie.

7

Conclusions

Research questions. Following the same methods presented, the developed CUSUM mechanism is most likely directly adaptable to the loco positioning system as well, but not to any modules that measure relative distance such as the Z-ranger or Multiranger deck. The measurement outage system is, however, directly applicable to all decks in the Crazyflie eco-system and so is the OW and user-defined test. If the CUSUM mechanism were to be used in a configuration where multiple decks measure different (or the same) states, no conclusion could be made to where the error originates from. If fault isolation between boards is desired, the *generalized observer sceheme* (GOS) or the *dedicated observer scheme* (DOS) [Chen, 1995] could prove useful as demonstrated in [Nilsson, 2020]. The main drawback would be the added complexity and computational overhead as several observers are needed.

The acceleration based PID control strategy proved to be an effective strategy during an outage. It does, however, introduce added complexity, both in the firmware structure and in the levelout and descent state compared to the other strategies. When dealing with increased noise faults, the Crazyflie experiences severe wobble which leads to the conclusion that switching to *modeDisable*, controlling using only onboard sensors is the most suitable action.

The developed module, strategies and methods are applicable to any future version of the Crazyflie. The CUSUM mechanism is lighthouse specific and the strategies utilized already have platform dependent configurations applicable.

Concluding results. Following the findings in the results section the fuzzy logic presented in Algorithm 4, see below, is implemented and suggested as strategies for the levelout and descent state when handling different fault scenarios. The variables s_l and s_d are the levelout and descent setpoint structures as described in Section 2.4 and the modes select the appropriate controller according to Figure 2.4. Holding roll and pitch during both the levelout and descent phase while commanding a lower PWM frequency is the preferred option when a run test has triggered the action as reasoned in Section 6.1. If an OW or user-defined error triggers the action, all states should be available and thus commanding a descent using velocity is pre-

Algorithm 4 Levelout and descent setpoint logic

Input: actionToExecute, errCode
Ensure: actionToExecute = levelout **or** descent

$s_l \leftarrow \emptyset$
$s_d \leftarrow \emptyset$
if $errCode =$ Run test **then**
  $s_{l.mode.\theta,\phi} \leftarrow modeAbs$
  $s_{l.\theta,\phi} \leftarrow 0$
else
  $s_{l.mode.x,y,z} \leftarrow modeVelocity$
  $s_{l.x,y,z} \leftarrow 0$
end if
$s_{d.mode.z} \leftarrow modeAbs$
$s_{d.z} \leftarrow 0$
if $errCode =$ Run test **then**
  $s_d \leftarrow s_l$
  $s_{d.mode.z} \leftarrow modeRaw$
  $s_{d.z} \leftarrow baseThrust$
else if $errCode =$ Scalar freq err **then**
  $s_{d.mode.\theta,\phi} \leftarrow modeAbs$
  $s_{d.\theta,\phi} \leftarrow 0$
else
  $s_{d.mode.x,y} \leftarrow modeVelocity$
  $s_{d.x,y} \leftarrow 0$
end if

ferred since this would result in a totally controlled landing. As an action whenever an outage is detected, controlling on velocity is the chosen strategy. Although the acceleration based control strategy performs better according to the conducted experiments for the higher velocities using velocity to levelout and holding roll and pitch with *modeAbs* in the Z-direction is the chosen strategy for two reasons. First, states could only be temporarily obscured and thus might reappear and enter the EKF, which would in that case cause a totally controlled levelout and descent if chosen. Secondly, the EKF is good at tracking Z position as seen in any of the common comparisons between the EKF and crossing beam method. But as seen in Table 5.3, holding roll and pitch to zero gives better performance if measurements do not reappear, which is the usual case.

Final remarks. Currently a fixed expected frequency rate is used to supervise if all measurement rates are nominal. This could easily be extended to give individual measurements different acceptable outage periods. For example if aggressive flight is to be carried out or if the Crazyflie's positioning system is expected to be obscured for extended periods, a higher outage period might be acceptable for a specific mea-

surement, but not for any other. What further needs to be taken into consideration is what to do when flying in a swarm. Perhaps commanding all other functional UAVs to stay put, leave space or just let the failing UAV fall out of the sky are all valid options that need to be considered. The experiments were also conducted in a rather spacious but nonetheless confined space. Further consideration or awareness from the user in terms of the capabilities of the fault actions with regards to the available flight space needs to be taken.

Bibliography

Agnihotri, N. (n.d.). *What is the 1-wire protocol?* Accessed: 2024-05-09. URL: https://www.engineersgarage.com/what-is-the-1-wire-protocol/.

Åström, K. and T. Hägglund (1995). *PID Controllers: Theory, Design, and Tuning*. English. ISA - The Instrumentation, Systems and Automation Society. ISBN: 1-55617-516-7.

Bitcraze (2021a). *Datasheet Crazyflie 2.1 - Rev 3*. Accessed: 2024-06-01. URL: https://www.bitcraze.io/documentation/hardware/crazyflie_2_1/crazyflie_2_1-datasheet.pdf.

Bitcraze (2021b). *PWM to thrust*. Accessed: 2024-05-10. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/pwm-to-thrust/.

Bitcraze (2023a). *Controllers in the Crazyflie*. Accessed: 2024-06-01. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/.

Bitcraze (2023b). *Lighthouse Kalman measurement model*. Accessed: 2024-05-25. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/lighthouse/kalman_measurement_model/.

Bitcraze (2023c). *Lighthouse positioning deck*. Accessed: 2024-05-20. URL: https://store.bitcraze.io/collections/decks/products/lighthouse-positioning-deck.

Bitcraze (2023d). *Lighthouse positioning system*. Accessed: 2024-05-10. URL: https://www.bitcraze.io/documentation/system/positioning/ligthouse-positioning-system/.

Bitcraze (2023e). *Stabilizer module*. Accessed: 2024-05-09. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/.

Bitcraze (2023f). *State estimation*. Accessed: 2024-06-01. URL: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/state_estimators/#complementary-filter.

Bitcraze (2023g). *The coordinate system of the Crazyflie 2.x*. Accessed: 2024-05-20. URL: https://www.bitcraze.io/documentation/system/platform/cf2-coordinate-system/.

Chen, J. (1995). *Robust residual generation for model based fault diagnosis of dynamic systems*. PhD thesis. University of York.

Diebe, J. (2006). *Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors*. Tech. rep. Stanford, California 94301–9010.

Dingler, S. (2022). *Normalized innovation squared (NIS)*. Accessed: 2024-05-20. URL: https://kalman-filter.com/normalized-innovation-squared/.

Ducard, G. J. J. (2007). *Fault-Tolerant Flight Control and Guidance Systems for a Small Unmanned Aerial Vehicle*. No. 17505. PhD thesis. ETH.

Frisk, E. (2001). *Residual Generation for Fault Diagnosis*. PhD thesis. Linköping University.

Ghalamchi, B., Z. Jia, and M. W. Mueller (2020). "Real-time vibration-based propeller fault diagnosis for multicopters". *IEEE/ASME Transactions on Mechatronics* **25**:1, pp. 395–405. DOI: 10.1109/TMECH.2019.2947250.

Granjon, P. (2013). "The CuSum algorithm - a small review".

Greiff, Marcus (2017). *Modelling and Control of the Crazyflie Quadrotor for Aggressive and Autonomous Flight by Optical Flow Driven State Estimation*. eng. Student Paper.

Gustafsson, F. (2000). *Adaptive Filtering and Change Detection*. John Wiley & Sons Lt.

Hägglund, T. (2021). *Reglerteknik AK, föreläsningar*.

Han, X. (2021). "Observer based fault diagnosis and fault tolerant control of nonlinear systems". *Automatic. INSA de Toulouse,* pp. 13–21.

Hönig, W. (2021). *Crazyflie-system-id*. Accessed: 2024-05-10. URL: https://github.com/IMRCLab/crazyflie-system-id/tree/main.

Jiang, C. and S.-B. Zhang (2018). "A novel adaptively-robust strategy based on the mahalanobis distance for gps/ins integrated navigation systems". *Sensors* **18**:3. ISSN: 1424-8220. DOI: 10.3390/s18030695.

Khan, A. Q. (2010). *Observer-based Fault Detection in Nonlinear Systems*. PhD thesis. Von der Fakultät fur Ingenieurwissenschaften der Universität Duisburg-Essen.

Luukkonen, T. (2011). *Modelling and control of quadcopter*. Tech. rep. Mat-2.4108. School of Science, Espoo.

Mueller, M. and R. D'Andrea (2012). "Critical subsystem failure mitigation in an indoor UAV testbed". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 780–785. ISBN: 978-1-4673-1737-5. DOI: 10.1109/IROS.2012.6385910.

Mueller, M., M. Hehn, and R. D'Andrea (2016). "Covariance correction step for kalman filtering with an attitude". *Journal of Guidance, Control, and Dynamics* **40**, pp. 1–7. DOI: 10.2514/1.G000848.

Mueller, M. W., M. Hamer, and R. D'Andrea (2015). "Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1730–1736. DOI: 10.1109/ICRA.2015.7139421.

Nilsson, A. (2020). *Sensor fusion and fault diagnostics in non-linear dynamical systems*. MA thesis. Uppsala university.

Page, E. S. (1954). "Continuous inspection schemes". *Biometrika*.

Redmon, N. (2003). *The bilinear z transform*. URL: http://www.earlevel.com/main/2003/03/02/the-bilinear-z-transform/ (visited on 2024-05-07).

Scipy (2024). *1-d interpolation*. Accessed: 2024-05-11. URL: https://docs.scipy.org/doc/scipy/tutorial/interpolate/1D.html#cubic-splines.

Simon, D. (2006). *Optimal State Estimation Kalman, H_∞ and Nonlinear Approaches*. Wiley, Hoboken, NJ, USA.

Smith, S. W. (1999). *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing.

Taffanel, A., B. Rousselot, J. Danielsson, K. Mcguire, K. Richardsson, M. Eliasson, T. Antonsson, and W. Hoenig (2021). *Lighthouse Positioning System: Dataset, Accuracy, and Precision for UAV Research*.

Waskom, M. (n.d.). *Visualizing distributions of data*. Accessed: 2024-03-27. URL: https://seaborn.pydata.org/tutorial/distributions.html#tutorial-kde.

Wickert, M. (2022). *Lecture notes in Real-Time DSP. Ch 7.* Accessed: 2024-05-10. URL: https://ece.uccs.edu/~mwickert/ece5655/lecture_notes/ARM/ece5655_chap7.pdf.

Wittenmark, B., K.-E. Årzén, and K. Åström (2002). *Computer Control: An Overview*. English. IFAC Professional Brief. International Federation of Automatic Control.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
August 2024

Document Number
TFRT-6257

Author(s)
Björn Fredlund

Supervisor
Tobias Antonsson, Bitcraze AB
Kimberly McGuire, Bitcraze AB
Zheng Jia, Dept. of Automatic Control, Lund University, Sweden
Björn Olofsson, Dept. of Automatic Control, Lund University, Sweden
Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Design, Implementation and Evaluation of a Fault Detection and Emergency Control Strategy for a Quadcopter System

Abstract

In this thesis, a fault detection and diagnosis system is developed for the Crazyflie 2.1 unmanned aerial vehicle. Different types of faults are investigated and suitable methods and tests are implemented. The system is mainly developed for the lighthouse positioning, but the tests and methods are generic and suitable for other parts of the Crazyflie eco-system. For the faults investigated, different methods of bringing the quadcopter to a landed state are explored. To keep user customizability high, different default actions are available to specify for each type of fault. The different actions are implemented as an extension of the onboard state machine and will cause the Crazyflie to respond to a fault in a certain way. The results indicate promising results for the developed control strategy but, because of simplicity, already implemented strategies are preferred. Overall, the implementation is simple, flexible and robust. Additionally, a propeller unbalance estimator developed in previous work was implemented and evaluated. The estimator is able to identify the damaged propeller and estimate the unbalance within the correct order of magnitude. It was, however, deemed too model dependent and thus not suitable as a general service indicator across different Crazyflie setups and configurations.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-69

Recipient's notes

Security classification