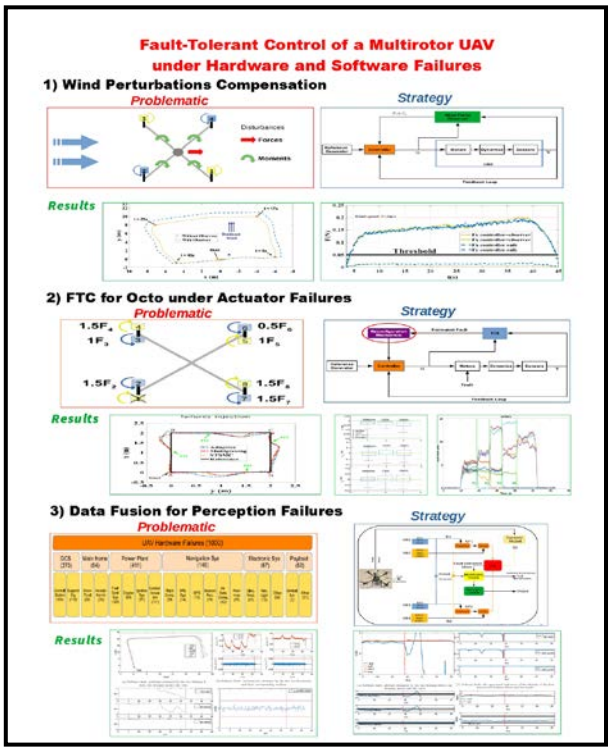
Fault-tolerant control of a multirotor unmanned aerial vehicle under hardware and software failures

Hussein Hamadi

Par **Hussein HAMADI**

Fault-tolerant control of a multirotor unmanned aerial vehicle under hardware and software failures

Thèse présentée en cotutelle pour l'obtention du grade de Docteur de l'UTC



Soutenue le 15 juillet 2020
Spécialité : Sciences et Technologies de l'Information et des Systèmes
: Unité de recherche Heudyasic (UMR-7253)

D2555

Fault-Tolerant Control of a multirotor Unmanned Aerial Vehicle Under hardware and software Failures

Hussein HAMADI

Thèse soutenue le 15 Juillet 2020 devant le jury composé de :

Rapporteurs:

Didier THEILLIOL	*Jean-Marc Thiriet*
Professeur des universités	Professeur des universités
Univ de Lorraine	Grenoble Alpes

Examinateurs:

Abdelhamid Chriette	*Maan Elbadaoui*	*Reine Talj*
Maître de conférences	Professeur des universités	Chargée de recherche cnrs
Ecole Centrale De Nantes	Univ De Lille	Univ de Technologie de Compiègne

Directeurs de Thèse:

Isabelle FANTONI	*Clovis FRANCIS*
Directrice de Recherche CNRS	Professeur des universités
Univ de Technologie de Compiègne	Univ Libanaise

Encadrants:

Benjamin LUSSIER	*Hassan SHRAIM*
Enseignant Chercheur	Enseignant Chercheur
Univ de Technologie de Compiègne	Univ Libanaise

Université de Technologie de Compiègne

Laboratoire Heudiasyc UMR CNRS 7253

15 - 07- 2020

Contents

Introduction

Unmanned aerial vehicles (UAVs) can be classified into fixed-wing aircraft, helicopters, and multirotors. Before 2010, fixed-wing aircraft and helicopters had overwhelming dominance in the field of both aerial photography and model aircraft sports. Nowadays, multirotors UAVs have consolidated their dominance in the market of small UAVs, due to their ease-of-use, and the development of performant sensors and onboard computers. In the domain of civil applications, the use of multirotor UAVs is indeed considered for a large number of delicate or expensive missions such as the exploration of an unknown environment, the surveillance or the intervention in potentially dangerous zones, the evaluation of damage and monitoring of forest fires, high voltage power lines, road traffic, overflight of mountainous and inaccessible areas, etc...

In fact, the development of multirotors is becoming more and more popular for many reasons, namely: the advancement in related technologies, preferential policy support, the promotion of open source autopilots...

Multirotor UAVs exist in several configurations including tricopter, quadrotor, hexarotor and octorotor. A quadrotor is a multirotor UAV with four independently controlled actuators or motors, where an octorotor is equipped with eight motors arranged in a coaxial or a star-shaped configuration. As a conventional quadrotor, an octorotor is equipped with multiple sensors (IMU, GPS, Magnetometer, LIDAR, optical flow...) and controlled by varying the speeds of its eight motors. The increased number of actuators in an octorotor presents many advantages in terms of payload and inbuilt redundancy. This hardware redundancy is useful to ensure safe flights in case of motor faults and failures.

Motivation

During their operations, multirotor UAVs are subject to different technical and operational constraints. Thus, it is important to increase their technical reliability to ensure a safe flight in critical areas despite faults in the system and external disturbances.

Fault Tolerant Control strategies can be used to increase reliability and safety in multirotors help to maintain the system's stability even when a partial or a complete fault occurs.

Due to a lack of motor redundancy in quadrotors, the complete loss of a rotor or the complete failure of a motor results in a vehicle that is not fully controllable. Hence one solution is to consider multirotors with additional motors like hexarotors and octorotors. Also, sensor and software faults as well as external perturbations of wind forces are important factors to be considered since they can lead to the system's instability.

The main motivation of designing a fault-tolerant control system is to try to avoid the consequences of all sort of faults (hardware, software and external perturbations) if possible, or at least, to minimize their negative effect on the system's performance.

Thesis Contribution

The main contributions of this thesis are:

- Development and validation of a wind estimation strategy using only position and acceleration information based on a super-twisting sliding mode observer. This observer is used in a robust wind tolerant controller that is also developed and validated during the thesis.
- Proposition of a new self tuning fault tolerant control mechanism to deal with motor failures in a coaxial octorotor and the experimental validation of this new approach.
- Development and validation of a fault-tolerant data fusion technique based on [19] to cope with hardware (GPS, IMU and Lidar) and software faults in the UAV's perception system.
- Comparative study of self tuning, adaptive and multiplexing Fault-Tolerant Control strategies for successive failures in a coaxial octorotor UAV.
- Experimental validation of the proposed strategies using several platforms developed within the framework of this thesis: assembly, programming and implementation of several multirotors platforms (quadrotors, hexarotors, octorotors) using different autopilots (pixhawk, Raspberry pi 3 + Navio2, cube).

Thesis Outline

The thesis is organized in 6 chapters:

- **Chapter 1**: This chapter provides a state of the art on fault-tolerant control and wind perturbations concerning multirotor UAVs and fault tolerance for multi-sensor data fusion.
- **Chapter 2**: This chapter presents the dynamic model of the quadrotor and the coaxial octorotor which are used to implement and validate our work. Complete and simplified models are presented and some model parameters are identified using real data sets.
- **Chapter 3**: This chapter presents a baseline controller based on the sliding mode super twisting control. This controller is implemented on the quadrotor and octorotor platforms that are used to validate our contributions.
- **Chapter 4**: This chapter presents the first major contribution of this thesis, which is the compensation of external wind perturbations. We develop and implement an observer-based super twisting controller, which is robust to wind perturbations.

Outdoor experiments are realized to validate the observer and the controller performances obtained by simulations.

- **Chapter 5**: As the second major contribution of this thesis, we propose a new FTC strategy based on a self tuning sliding mode control (STSMC) where the gains of the control law are adjusted depending on the detected errors to maintain the stability and controllability of the system. We compare this method through extensive real indoor experiments with two other FTC methods: a multiplexing method using error detection and system recovery that applies precompiled mixing strategy to detected faults, and an adaptive method that uses a robust controller and fault masking to passively tolerate faults.

- **Chapter 6**: As the third major contribution of this thesis, we present the design and implementation of a sensor fusion scheme that tolerates hardware faults in the sensors and software faults in the data fusion. Based on [19], it aims to tolerate both hardware sensors faults (GPS jamming, IMU lock or freezing, magnetometer sensitivity to high power magnetic fields...) and software faults (faults in the Kalman filter, bad gains value....).

Finally, the thesis concludes with a summary about the obtained results and an outlook about potential improvements of our work.

Concepts and State of the Art

Contents

This chapter presents a state of the art concerning two aspects for unmanned aerial vehicles (UAVs): external wind disturbances recognition and resilience, and fault-tolerant control against hardware (particularly concerning actuators and sensors) and software faults. First, basic concepts and principles of dependability are presented: threats, means and attributes, focusing particularly on one mean, fault tolerance. Second, we introduce the notion of fault tolerance applied to automatic control. Third, the concept of robustness in control is discussed, notably its similarity and links to fault tolerance. Fourth, we present a state of the art regarding Fault Tolerant Control (FTC) in UAVs. Fifth, we present a state of the art regarding external wind perturbations on the dynamics of the UAV. Finally, we present fault tolerance techniques for data fusion mechanisms to deal with sensors and software faults in perception.

1.1 Dependability

In recent years, autonomous systems have been integrated in different domains such as space aircrafts, space exploration, intelligent vehicles, etc. In such applications, a system failure may have catastrophic consequences regarding human lives or the economical impact. Thus arises a need to increase the dependability of such systems. From a dependability point of view, UAVs are primarily concerned with the safety, reliability and security attributes, which will be presented in this section.

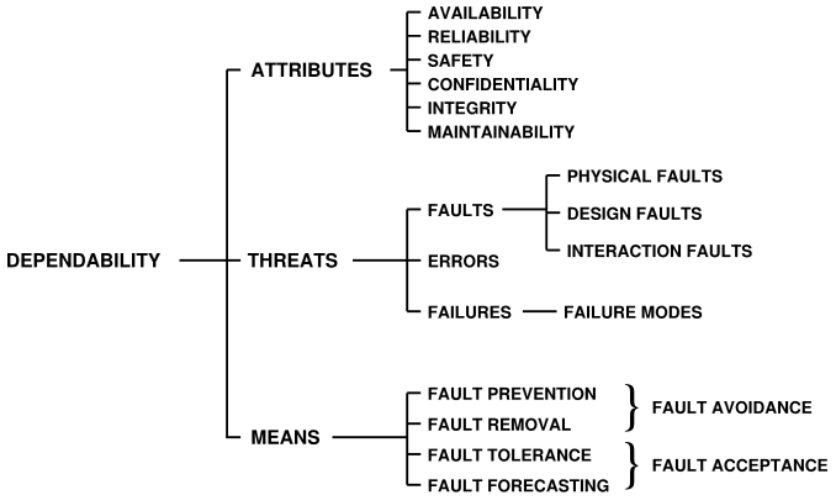


Figure 1.1 – Dependability tree [14]

1.1.1 General principles of dependability

In this section, all the presented definitions are taken from [14].
In software engineering, the dependability of a computing system is its ability to provide services that can justifiably be trusted within a time-period. Correct service is considered delivered when the system behaves as it is intended to. Dependability can be broken down amongst three aspects:

- Its attributes - how to characterize the dependability of a system.
- Its threats - what can negatively affect the dependability of the system.
- Its means - how to increase the dependability of a system.

Fig. 1.1 shows the dependability tree that represents these three aspects.

Attributes

The attributes are what characterizes a system's dependability. These can be assessed to determine its overall dependability using qualitative or quantitative measures. [14] defined the following dependability attributes:

- **Availability**: readiness for correct service.
- **Reliability**: continuity of correct service.
- **Safety**: absence of catastrophic consequences on the user(s) and the environment.
- **Confidentiality**: absence of unauthorized disclosure of information.
- **Integrity**: absence of improper system alteration.
- **Maintainability**: ability for a process to undergo modifications and repairs.

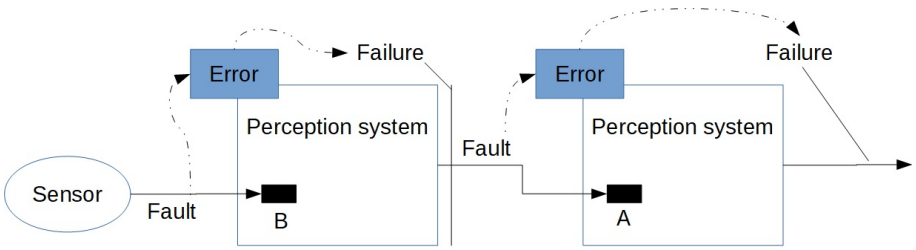


Figure 1.2 – Example of error propagation in a perception system

Moreover, the confidentiality and integrity attributes can be considered together as another attribute: security.

Here, we focus particularly on the reliability and safety attributes. However, these concerns can be contradictory, since in some critical situations safety concerns could require to safely stop the UAV while reliability concerns could require to continue the mission.

Threats

By definition, threats are what can affect a system and diminish its dependability. The threats are decomposed in three elements, namely faults, errors and failures.

- Failure: A failure is an event that occurs when the delivered service deviates from the correct service.
- Error: An error is the incorrect part of the system state that can cause a subsequent failure. An error can cause other errors in the system, until this propagation ultimately causes a failure by making the system deviates from its correct service. Some errors may stop propagating before becoming a failure; for example, a threshold condition on a variable may stop an error on this variable from propagating if the erroneous value is close enough to the correct value to still be under the threshold. An example of error propagation is given in Figure 1.2 adopted from [18].
- Fault: A fault is the supposed or adjudged cause of an error. It may manifest as incorrect lines of code, weariness in components, bad design, etc. A fault is called active when it produces an error, and latent when it has not been activated yet. A latent fault is hidden from regular means of detection, and can only be identified once it has caused an error or after a detailed analysis.

According to their origin, nature and creation phase, faults are classified into several partially overlapping classes, such as:

- *development faults* include all fault occurring during development while *operational faults* include all fault occurring after development when the system is used.
- *hardware faults* include all fault that affect hardware while software faults include all fault that affect the *software components* of a system.

- *external faults* include all faults which are external to the system (such as weather, electromagnetic rays, a nail on the road, etc.) while *internal faults* include all faults internal to the system (bad design, incorrect lines of code, frail circuitry, etc.).
- *permanent faults* include all faults whose presence is continuous in time while transient faults are faults that are present only for a bounded time.

In this manuscript, we primarily consider hardware faults on actuators (both internal and external occurring during development and operation), and internal faults on perception (both hardware and software occurring during development and operation).

Means

In order to increase the dependability of a system, four means exists:

- **Fault prevention**: it deals with preventing the occurrence or introduction of faults in the system, through good design process and good practices
- **Fault removal**: it is done during development through verification and validation, usually dynamically with testing or statically with formal methods.
- **Fault forecasting**: it is the ability to predict potential faults so that they can be removed or their effects can be avoided. It is usually achieved through statistical analysis and mostly considers hardware faults.
- **Fault tolerance**: it deals with developing mechanisms that can allow the system to still deliver correct service in the presence of faults through error detection and system recovery. To be achieved, it requires some form of redundancy in the system.

1.2 Fault Tolerance

In this thesis, we particularly focus on Fault Tolerance (FT), the mean of dependability that seeks to maintain a system's correct service despite the presence of faults.

1.2.1 Principle of Fault Tolerance

Fault Tolerance is usually carried out either:

- Actively though error detection and system recovery
- Passively through fault masking.

Active fault tolerance requires a fault detection and identification (FDI) process. This FDI detects and localizes the faults occurring in the system. The main advantage of this method is that since we are monitoring the system health, we can know if the system is getting close to its limits (particularly in term of redundancy), and recovery can be executed if the system is deemed not safe enough (such as an emergency landing or an order to return to base). The main drawback of AFTCs is that the fault is not tolerated

during the time needed to detect and isolate it, possibly leading to a system's failure if the detection takes too long.

In contrast to active fault tolerance, passive fault tolerance strategies consist in masking the faults, using for example a robust controller able to tolerate some specific faults (like model uncertainties or actuators failures) or redundant components associated with a voter. Oppositely to active fault tolerance, such strategies can avoid the time delay needed to detect the error, but if error detection is not implemented at all it could lead eventually to a failure when the system reaches its fault-tolerance limits.

1.2.2 Error detection

Error detection is a prior step before the implementation of system recovery. It aims to detect erroneous states in of the system before they cause a system failure. There are three main methods of error detection:

- The Duplication/Comparison method compares the results provided by at least two redundant units providing the same service and independent to the faults to be tolerated. Typically, physical faults are detected by redundancy of hardware components such as sensors in perception systems, and software faults by N-version programming such as the diversification of data fusion algorithms.
- The Temporal Watchdog method detects temporal errors in the system, for example by checking that its response time does not exceed a maximum value (time out).
- The Likelihood Checks method seeks to detect errors in outliers of the system's state. A typical example of likelihood control for a perception sensor is to check whether its output belongs to a range of possible values.

1.2.3 System recovery

System recovery seeks to transform the detected erroneous state of the system into an error-free state. It is performed by two complementary methods:

- **Error handling** allows substituting an erroneous state by an error-free state. This substitution can itself take three forms: the rollback that brings the system back to a correct state that occurred before the occurrence of the error; the rollforward seeking a new state from which the system can operate (possibly in degraded mode); and the compensation where the erroneous state has enough redundancy to allow its transformation into a correct state.
    - ◇ The rollback consists in returning to a correct previous state. This requires capturing and periodically backing up the state of the system, and creating a set of possible checkpoints, from which a consistent state can be restored. The advantage of this technique is that it is independent of the application. However, there are several challenges to it: the checkpoints themselves must be error-free, and they are expensive in terms of backup size. We must add to that the difficulty of making consistent backups, and the over-cost of time

required for their establishment. Finally, this technique is usually difficult to implement for embedded systems like cars, planes or UAVs, as these systems function in open environments that can not be turned back to a previously saved state.

- ◇ The rollforward requires finding a new correct state from which the system can operate, usually in a degraded mode. This technique remains specific and dependent on the application. The search for a new acceptable state often consists of a re-initialization of the system and the acquisition of a new execution context with the environment.
- ◇ The compensation requires the presence of redundancies in the system allowing it to provide a correct service despite the errors that may affect it. This compensation comes in two forms: detection and compensation if it is consecutive to an error detection, or fault masking if it is applied systematically without detection of errors.

- **Fault handling** is not necessary to avoid failure, but aim to prevent a new activation of the fault causing the error. It is possible either to bring a restorative solution, or to deactivate the faulty components. This method is composed of four successive phases:
  - ◇ **Diagnosis** identifies and records the causes of errors in terms of both location and type.
  - ◇ **Isolation** performs physical or logical exclusion of the faulty components from further participation in service delivery.
  - ◇ **Reconfiguration** either switches in spare components or reassigns tasks among healthy components.
  - ◇ **Reinitialization** checks and records the new configuration and updates system tables.

1.2.4 Fault Masking

As mentioned before, error compensation can be done through detection and compensation if it is consecutive to an error detection, or can be systematic through fault masking. Fault masking is usually done using duplication/comparison techniques and voters: computation is carried out by three or more identical or similar components whose outputs are voted. Recently in UAVs robust controllers are also implemented that allow to tolerate some software (imprecisions in the model) and hardware (actuators failures) faults. These techniques also belong in fault masking.

1.2.5 Examples of redundancy

The implementation of FT requires redundancies in the system. This redundancy can take several forms:

1. Hardware Redundancy: addition of replicated modules, or use of extra circuits for fault detection.

2. Information Redundancy: addition of extra information to data, allowing error detection and correction, such as error-detecting codes, error-correcting codes (ECC), and self-checking circuits.

3. Software Redundancy: N-Version Programming, Transactions, ...

4. Time redundancy: performing the same operation multiple times such as multiple executions of a program or multiple copies of data transmitted.

1.3 Fault and failure in automatic control systems and their classification

We presented in section 1.2 the definitions of faults, errors and failures as threats for the dependability community. The control community uses other definitions for faults and failures, that we will present and discuss in this section. We still also present the classification of representative faults on sensors and actuators proposed by this community.

1.3.1 Faults

In control systems, a fault constitutes an unexpected change in a system parameter from the acceptable/ normal condition, which can degrade system performance. It is a fact that a fault can disturb the normal operation of a system from the desired one, but may be tolerable. This definition appears to be distinct and more clear than that from the dependability community, thus it will be adopted in this thesis. According to their location, faults in automatic control systems are classified as follows:

- *Actuator faults*: correspond to an abnormal actuator behavior.
- *Sensor faults*: correspond to significant errors in the sensors readings.
- *Plant faults*: correspond to changes in the dynamical input/output properties of the system.

Note that compared to the dependability community, both actuator and sensor faults are hardware faults, while plant faults are mostly software faults. Faults can be also classified with respect to the way they are modeled as:

- *Additive faults*: faults that can be treated as external inputs or biases. They usually affect the sensors of the system (GPS, IMU ...).
- *Multiplicative faults*: faults that are represented by a multiplication with the state and input vectors. Both sensors and actuators are vulnerable to such kind of faults.

In this thesis, in case of actuators faults, we considered multiplicative faults which are also called losses in control effectiveness (Figure 1.3(d)), and are modeled as follows:

$$u_f^i = (l_i)u_n^i \tag{1.1}$$

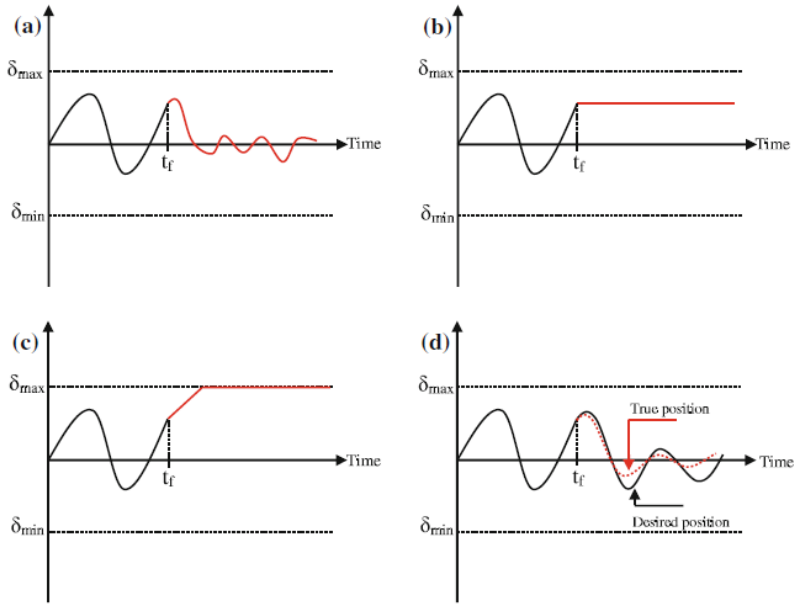


Figure 1.3 – Types of actuator faults and failures (adapted from [12]). a Float failure. b Lock in place failure. c Hard over failure. d Loss of effectiveness

where i indicates which actuator is affected by the fault, u_f is the actual erroneous control output, u_n is the nominal control output and l_i is the multiplicative factor representing the loss of effectiveness of the actuator. $l_i = 1$ and $l_i = 0$ means respectively that the i-th actuator is completely healthy or failing.

In case of sensor faults, we considered both additive and multiplicative faults.

1.3.2 Failures

In the control domain, a failure is considered as an intolerable malfunction, and describes the inability of a system or a component to accomplish correctly its function. A system failure, resulting from one or more faults, should be avoided by means of a fault-tolerant controller. This definition is very similar to the one used in the dependability community, as a system's function is part of the service that it offers.

According to [12], actuators failures are divided in three categories, as seen in Figure 1.3:

- *float failure*: This occurs when the control surface moves freely providing random outputs.
- *lock in place failure*: This occurs when an actuator becomes stuck and immovable.
- *hard over failure*: This occurs when the actuator moves to its maximum output due to an electrical or software fault.
- *Loss of effectiveness*: This type of failures is similar to the multiplicative faults presented in 1.3.1.

Note that, as previously stated, these actuators failures are also considered in the dependability community as system's errors until they possibly propagate and ultimately become system's failures as they negatively impact the system's correct service.

1.3.3 Failure modes identification

A risk analysis process starts by identifying the potential failure modes of a given system. For a multirotor UAV, some failure modes are hardware-related while others are software-related, and some of them can be organized by subsystem as follows ([119]):

1. Altitude Sensor Failure Modes: Loss of valid return from the altitude sensor can cause the vehicle to become unstable regarding motions on the vertical axis which introduces the potential for loss-of-control (LOC). This failure mode can result from multiple causes: flying above the maximum range of the sensor, flying over an obstacle, improper filtering, vibration from airframe, etc.

2. IMU Failure Modes: A failure of the IMU leads to a worse/wrong, or even no attitude information, which could make the control impossible.

3. Motor Failure Modes: A motor failure is considered as a catastrophic failure. For some multirotor configurations, such as quadrotor or a specific configuration of hexarotor, loss of control is unavoidable after one motor failure. This failure could occur after a progressive degradation of the motor caused by prolonged use or particles in the motor housing or a sudden contact with part of the environment (tree branches, debris falling from some height, etc.). This could also be caused by Electronic Speed Controller (ESC) overheating due to an overdraw on current or a hot environment.

4. Navigation and Stabilization Algorithms Failure Modes: A navigation algorithm failure mode causes an erroneous localization of the system, either due to wrong sensors information or some software faults in the navigation process. A stabilization algorithm failure mode leads to an unstabilizable system and can occur when a linear control law is combined with fast dynamical flight (nonlinear effects), or in case of bad control gains when the system properties change due to a payload for example.

5. Ground Station Communications Failure Modes: Communication losses cause the loss of visibility of the system. They can be due to signal interference, router problems, going out of range or latency and delay in data transmission.

6. Battery failure Modes: A battery failure cause the loss of the system's power. It can be caused by battery damage, short circuit, overheating due to current exceeding or failure of its electric connections due to vibrations and forces during flight.

In this thesis, we particularly focus on faults and failures occurring on the actuators, the sensors and the computational processes (cases number 1 to 4). In our opinion, these faults represents a major risk for UAVs due to the impact they have on the stability of the vehicle combined with the high likelihood of occurrence during the life of a multirotor.

1.4 Robustness

The concept of robustness is found in many different contexts. In the field of robotics, robustness is seen as a response to the significant variability of the execution context that a robot may be confronted with. This variability is due to the open environment in which the robot evolves, which may present obstacles or variations in occurrence of unforeseen events, and uncertainties of observation and action, which may result in the system's inability to accurately observe its own state and that of the environment, and to predict the precise consequences of its actions. However, there is no unique definition of this term in the literature, and it is not easy to distinguish between the notion of robustness in the field of robotics, and the notion of fault tolerance in the field of dependability.

In our opinion, a possible link between robustness and fault tolerance is that both aim to increase the ability of the system to deliver correct behavior, but the fault tolerance target a specific subset of the adverse situations facing an autonomous system: the faults. From this point of view, fault tolerance therefore appears as a subset of robustness.

However, it seemed to be helpful to differentiate between the notion of robustness and the notion of fault tolerance according to the origin of the adverse situations: either internal or external to the autonomous system. Thus, we retain the two following definitions from [100]:

- Robustness is the delivery of a correct service despite adverse situations due to uncertainties about the system environment (such as unexpected obstacle, external perturbations of wind on the dynamics of the UAV, etc.)
- Fault tolerance is the delivery of a correct service despite faults or failures affecting the various resources of the system (such as loss of effectiveness of an actuators, sensor failure, or software fault, etc.)

Robustness in robotic systems (UAVs, intelligent vehicles...) is typically achieved either by functional redundancy, aimed at compensating the limitations of hardware components and software algorithms (such as a combined use of camera, laser sensor and bumper to detect obstacles, or complementary localization algorithms), or by using uncertainties management, aimed at compensating environment uncertainties for control and observation, such as robust control laws (sliding mode, adaptive laws..) which automatically compensate for system imperfections and uncertainties or Kalman filters which can filter out the inconsistency in the sensors readings by evaluating its variances with respect to time.

1.5 Fault-Tolerant Control in control

In this section, we will discuss the different methods used in control to deal with faults and failures in dynamical systems. Such process is called in the control community fault tolerant control (FTC). Note that in the control community, most of the research papers address the problem of actuators faults, whereas only few works are dedicated to study other kind of faults in the system, namely sensors and software faults. The motivation for the early research in the field of fault tolerant control was in the area of flight control

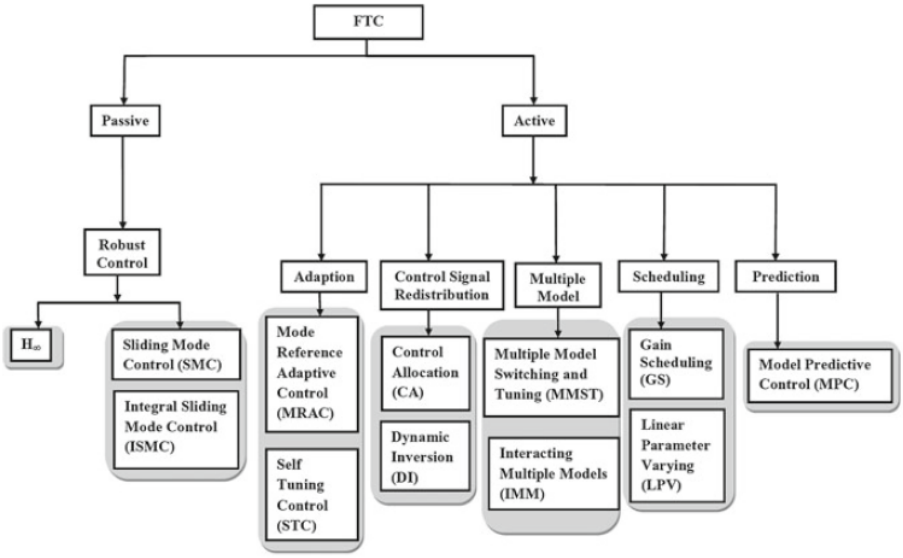


Figure 1.4 – Classification of FTC methods (adapted from [149])

systems to improve the reliability and safety of aircrafts. A fault tolerant control system has the capability to maintain some level of acceptable performance or degrade gracefully after the occurrence of a fault. Usually to design such a controller, the system should have redundant control actuators, which can be efficiently used and exploited to achieve fault tolerance. In the case of a failure in certain actuators, the control effort can be distributed to healthy actuators to maintain the desired performance or at least some level of acceptable performance. As previously stated in section 1.2. redundancy is absolutely necessary for fault tolerance. This redundancy can be achieved by the direct replication of the hardware (actuator/sensor) or it can be in the form of dissimilar hardware having similar functionality.

Depending on the way the problem is tackled, FTC systems can be classified as passive fault tolerant control (PFTC) systems or active fault tolerant control (AFTC) systems. A block diagram representing the classification of FTC methods is shown in Fig. 1.4.

1.5.1 Passive Fault Tolerant Control Systems

In PFTC systems the controller is of a fixed structure and is designed off-line [167]. Due to the fact that PFTC systems do not require error detection, PFTC methods are computationally more attractive. In passive fault tolerant control schemes, the idea is to design the controller using robust control techniques and considering the system's faults are uncertainties. In practice, passive fault-tolerant control (PFTC) strategies consists mostly in masking the faults by using a robust and reliable controller able to deal with all expected fault without the need to detect and identify them. The main advantage of using PFTCs is that it can avoid the time delay needed to detect the error since it is not required. However, in this case, we usually do not monitor the system's health, and this could lead eventually to a failure when the system reaches its fault-tolerance limits.

Moreover, adaptation to failing situations from robustness is generally not as efficient to recover from a particular failure as a specifically designed recovery mechanism.

The $\mathscr{H}_\infty$ methodology is a well-known technique in the field of robust control, and can take into account performance and stability requirements. The idea behind the $\mathscr{H}_\infty$ control methodology is to design a controller which can provide stabilizing properties and minimize the effects of uncertainties or disturbances on certain outputs of interest. When designing a robust controller, the worst case performance specifications are taken into account, which may lead to a requirement to sacrifice the nominal performance of the system. Since faults usually occur rarely in the system to sacrifice the nominal performance to obtain robustness against a certain class of faults may not be appropriate.

Another robust control methodology is the Sliding Mode Control (SMC). This approach will be discussed in depth in this thesis. SMC schemes have inherent robustness properties against matched uncertainties (i.e. uncertainties which have effects on the input channels). The basic concept is to first design a sliding surface, and then to specify a controller to induce and maintain a sliding motion on the sliding surface. Due to its inherent robustness against matched uncertainties, SMC schemes have the capability to directly deal with actuator faults, which can be effectively modeled as matched uncertainties. However, this requires for the system to have sufficient actuators redundancy for the fault to be tolerated.

1.5.2 Active Fault Tolerant Control Systems

Active fault tolerant control (AFTC) systems on the other hand rely on error detection from a fault detection and identification (FDI) unit scheme to perform a system recovery. Specifically AFTC systems react to faults/failures by reconfiguring control actions so that stability and acceptable performance of the system can be maintained. In certain circumstances, degraded performance may have to be accepted. A typical AFTC system is represented in Figure 1.5. The structure of an AFTC system is usually more complex compared to PFTC systems, but can deal with a wider class of faults. The error handling in AFTCs is done using rollforward and error compensation techniques presented in Section 1.2.3. It is difficult to apply rollback techniques to dynamical systems since it is usually difficult to influence the environment to return to a previous state. The reconfiguration mechanism changes the parameters or structure of the controller, (usually) based on the fault information provided by the FDI unit. In the literature, AFTC methods are further sub-classified as projection-based methods, and online control redesign methods. In projection based methods, one of the pre-computed controllers from a set, which have already been designed off-line for a specific fault scenario, is selected, depending on the fault information provided by the FDI scheme. In online control redesign methods, depending on the fault information provided by the FDI scheme, the new controller is synthesized online. Online control redesign methods are also categorized as reconfigurable control or restructurable control. In reconfigurable control, the controller parameters are computed online depending on the fault information provided by the FDI unit, whereas in restructurable control both the structure and controller parameters are computed online.

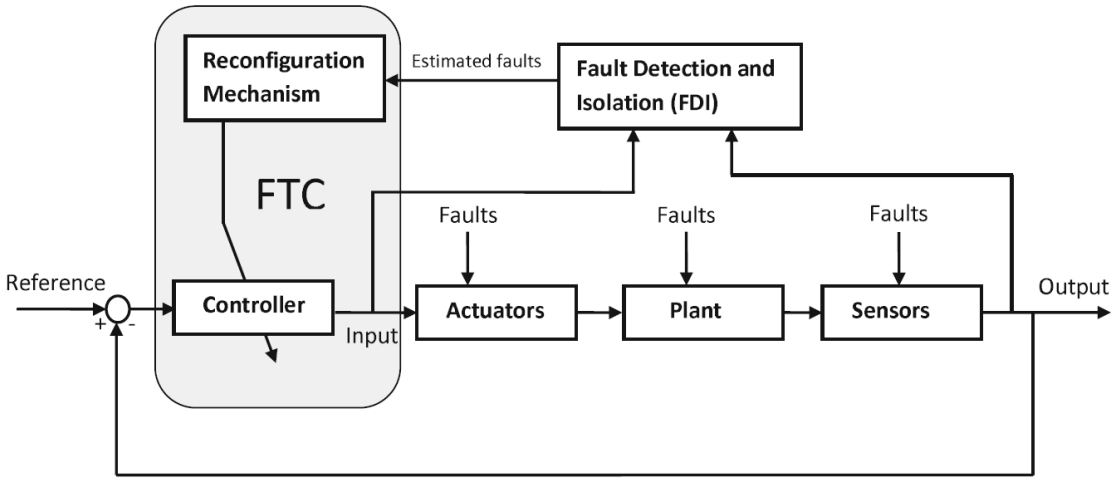


Figure 1.5 – Main structure of AFTC systems (adapted from [149])

Control allocation

In the literature, control allocation is used as part of active fault tolerant control techniques, as a way to redistribute control signals amongst healthy actuators to compensate the failing ones.

In order to redistribute the control signals in the case of faults or failures, diversified configurations targeting either the nominal mode or specific faults are used to transform the virtual control input $\nu(t)$ into the real control input $u(t)$. The structure of the Control Allocation (CA) scheme is shown in Figure 1.6, which shows that the CA element is not part of the virtual control law $\nu(t)$. The virtual control effort $\nu(t)$ produced by the controller is directly translated into actuator deflections by the CA module.

The hardware redundancy provides opportunities that can be exploited when designing fault tolerant controllers. The advantage of the CA method is that the underlying control law can be designed separately in order to produce the desired control effort and the CA distributes this virtual effort among the available actuators to achieve the required system performance.

1.5.3 State of the Art on Active and Passive Fault Tolerant Control

Many different approaches have been considered for FTC in different dynamic systems (UAV, intelligent vehicle, etc...). In PFTCs, the main focus is on using robust controllers such as sliding mode [36, 109, 16, 148, 102, 40], and $\mathscr{H}_\infty$ [95, 101, 83] control laws presented previously in section 1.5.1. Also, as a good example of PFTC, adaptive sliding mode controllers are used in several works to mask the faults by adapting the controller gains with respect to the system health [23, 91, 92, 56].

In contrast to PFTCs which rely in the literature mostly on robust controllers, many techniques are considered as the basis of AFCTs: (1) Control Allocation (CA) [73, 151, 125, 135, 136] is used, as explained previously, to handle actuator faults or failures without the need to modify the underlying control law. (2) Linearization-

Design approaches	*references*
Adaptive control	[23, 91, 92, 56]
Control allocation	[73, 151, 125, 135, 136]
Sliding mode control	[36, 109, 16, 148, 102, 40]
Dynamic inversion	[99, 97]
Gain scheduling	[126, 114, 20]
$\mathscr{H}_\infty$ robust control	[95, 101, 83]
Multiple model	[144, 76, 27]
Linear parameter varying	[103, 121, 104]
Model predictive control	[75, 142]
Model reference adaptive control	[127, 34]
Multiple model switching and tuning	[28, 60]
Interacting multiple models	[165, 166]

Table 1.1 – Examples of existing control design methodologies used in FTC

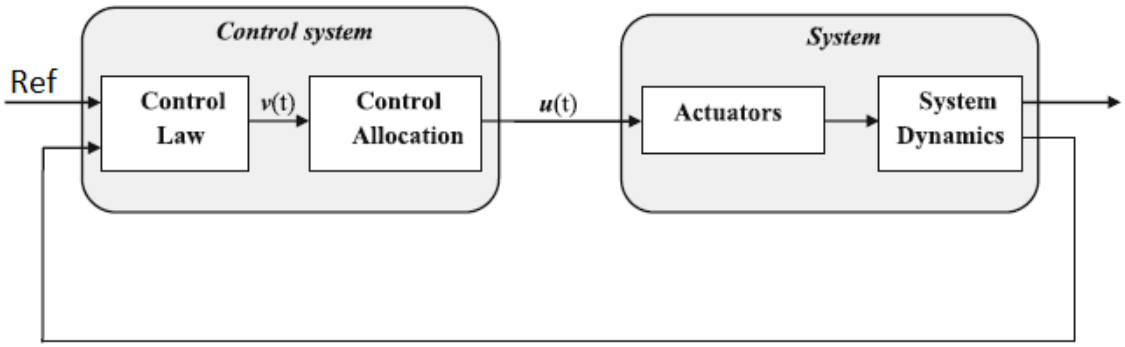


Figure 1.6 – Control allocation scheme (adapted from [65])

based methods, such as Nonlinear Dynamic Inversion (NDI), are nonlinear control methods which can handle nonlinearities and faults in the model [99, 97]. (3) The gain scheduled controller [126, 114, 20] is an approach designed to select several operating points, covering the range of the plant's working conditions. (4) In multiple models [144, 76, 27], the goal is to detect, isolate and estimate an accurate state of the system in presence of faults/failures around an operating point. (5) Linear parameter varying (LPV) [103, 121, 104] techniques can be considered as an updated version of gain scheduling methods, by taking into account local stability and performance assurances of the system. (6) Model predictive control (MPC) and Model reference adaptive control [75, 142, 127, 34] are techniques where actuators and sensors faults can be represented through modifications of the constraints in the MPC problem definition. (7) The multiple

model switching and tuning (MMST) [28, 60] technique is applied by setting up parallel identification models and corresponding controllers and proposing a suitable strategy for switching among the controllers. (8) Interacting multiple model (IMM) [165, 166] is a technique where the occurrence or recovery of a failure in a dynamic system has been explicitly modeled as a finite-state Markov chain with known transition probabilities.

A summary of these techniques is proposed in Table 1.1. This is not an exhaustive list, as indeed other techniques exist in the literature.

1.6 Fault Tolerance Applied to Unmanned Aerial Vehicles

Nowadays, several application fields rely on the use of drones or UAVs. As they are small, light, and maneuverable, drones have many uses in both military and civilian domains, such as fire detection, inspection of power lines, flying surveillance, aerial photography, roof and solar panels inspection, 3d mapping, etc. In order to accomplish these missions safely, UAVs must be equipped with an onboard computer or autopilot, different sensors (GPS, IMU, Magnetometer, barometer...) and actuators to ensure the full functionality of the drone.

As previously explained in section 1.2, a motor's failure failure will first manifest as an error in the system until it has undesired consequences on the system's behavior (which will usually happen quickly). Fault-tolerant control (FTC) strategies for UAV have recently received significant attention in research fields due to the increasing awareness about the risks resulting from components failures, and the need for reliable and safe systems in critical applications [167].

1.6.1 State of the art of FTC for quadrotor UAVs

Among different types of multirotor UAVs, the quadrotor has been widely used by the academic communities due to its simple design and dynamics. This allows to easily develop and test several control algorithms [13]. A lot of recent works have focused on fault detection and identification (FDI) and fault-tolerant control for quadrotors UAVs.

A complete loss of a motor/propeller of the quadrotor leads to the inability to fully control the system's attitude [85]. Due to this difficulty, there are few works considering the case of complete rotor loss. We can find solutions to this problem mostly in [115], [98], [118]. The most commun solution is to sacrifice the controllability of the yaw angle to still be able to command the UAV's position and altitude by making it continuously rotate around the Z axis.

In the literature, different strategies were developed considering the FTC problem against partial actuators failures in a quadrotor. In [35], a reconfigurable FTC is proposed, based on a trajectory re-planning scheme and an online decision making strategy using differential flatness. This strategy consists in synthesizing a reconfigurable feedback control with a modified reference trajectory once an actuator fault has been diagnosed by a fault detection and diagnosis scheme, which uses a parameter estimation based unscented Kalman filter. In [123], the authors proposed a nonlinear adaptive feedback linearization strategy where the solution takes into account the management of the control

authority by incorporating the post-fault model of the actuator, which guarantees an acceptable performance in presence of certain types of faults in the actuators. Other proposed strategies addressing partial actuator failures can be found in [15], [154] and [140], where the authors proposed robust and adaptive control laws based on sliding mode and backstepping theories to compensate the loss of effectiveness in actuators.

1.6.2 State of the art on fault-tolerant control for UAVs through actuators redundancy

In order to maintain complete control on the UAV in the case of one or more motor failures, to our knowledge, the only solution is to consider multirotors with redundant actuators, i.e., hexarotors [133] and octorotors [130]. The main advantage of such configurations is the possibility of tolerating multiple failures without losing complete controllability. It thus allows the UAV to keep a stable flight and possibly continue its mission.

The added hardware redundancy in coaxial octorotors, though allowing more dependability in the system, complexifies the definition of the control law. The actuators redundancy is considered as an important factor in any FTC strategy implementation, since it allows to maintain complete controllability of the system not only with partial faults, but also with up to four total motors failures. However, an effective motor control allocation of the remaining healthy motors is required to achieve acceptable performance [168].

In [87], a time delay fault tolerant control (TDC) is proposed to maintain attitude stabilization after a fault on one or more rotors by treating the fault as a disturbance. Another work [163] suggests a fault-tolerant control strategy using cooperation between a radial based function neural network, fuzzy logic control and sliding mode control (SMC) technique in presence of actuator faults, to alleviate the chattering and to maintain good tracking of the system. In [11], two FTC schemes using linear parameter varying system representation, with a combination of SMC theory and control allocation, are developed and tested in the presence of uncertainties, as well as faults and failures. In the first scheme, the knowledge of the rotor effectiveness is required in order to apply an online control allocation methodology, and to redistribute the control signal to working motors. In the second scheme, this knowledge is not necessary. In [164], a neural network, an interval type-2 fuzzy logic control approach and a sliding mode control technique are used to design a controller, named as a fault tolerant neural network interval type-2 fuzzy sliding mode controller. This control scheme has many advantages, since it allows avoiding difficult modeling, attenuating the chattering effect of the SMC, reducing the number of rules for the fuzzy controller, and guaranteeing the stability and the robustness of the system. Note that in these different studies is that only simulations are presented to validate the effectiveness of the proposed methods.

Another fault tolerant control scheme based on nonlinear model predictive control for a Y6 coaxial tricopter is proposed in [108]. In this study, a cascaded closed-loop control methodology is proposed which incorporates a reconfigurable low-level controller. The effectiveness of the presented fault-tolerant scheme is validated by following an 8-shaped trajectory with a complete loss of one rotor. In [113], the authors showed that a passive FTC controller based on a second order sliding mode control outperforms an active FTC

controller designed using a pseudo-inverse dynamic control re-allocation and a first order sliding mode control.

A global active FTC of underactuated UAV with redundant actuators (hexarotors and octorotors) is proposed in [162] where the entire architecture contains a baseline controller (adopted from [161]), and a fault detection and isolation scheme based on a robust parameter identification approach. The FDI identifies the generated thrust and torques by the actuators in normal and faulty situations and a controller reconstruction module then calculates a feasible solution to the control allocation problem in faulty cases. In [22], a fault-tolerant control for an octorotor UAV is proposed based on a combination of Backstepping and Dynamic Surface Control. In this method, the uncertainties and parameters variations caused by faults and failures are estimated and then taken into account in the control allocation algorithm, based on a Moore-Penrose pseudoinverse [74], which distributes the control efforts among the actuators and minimizes the energy consumption. Another FTC strategy based on the reconfiguration of the remaining rotors was proposed in [39], to handle the disturbance torque and the large yaw rate which occurs after the failure of a one rotor in a hexarotor UAV. The main issue in all these different studies is that only simulations are presented to validate the effectiveness of the proposed methods.

In contrast, a few studies have addressed the problem of fault-tolerant control for actuators faults in redundant UAVs providing experimental validations for the proposed strategies, such as in [116], where the authors proposed a fault detection and a fault-tolerant control FTC scheme which can handle up to two actuators failure for an hexarotor. This FTC scheme uses a nonlinear Thau observer to estimate the states of the UAV and to detect the actuators failures. It also uses a sliding mode and disturbance observer to stabilize the UAV despite the existence of disturbances. However, the authors cited that for the frames that have more than six motors (like octorotor), the fault detection and fault-tolerant control schemes are more complex, which is out of scope of their research and will be investigated in further studies.

In [152], an offline control mixing fault-tolerant control strategy is detailed, which consists in computing a set of explicit laws, where each one is dedicated to a fault and obtained by solving an optimization problem considering this particular fault. Fault detection and isolation must be associated with the FTC scheme to match a corresponding control mixing law. From this detection, a lookup table containing all possible faults and failures combinations is used to select the control law to apply for the system recovery. In [129], an online ASMCA scheme is proposed as a passive robustness approach to adapt to motors failures. The stability of the overall system is verified using the Lyapunov theory. However, the convergence time may vary depending on the gain tuning process and the initial condition of the system. These two strategies represent two typical examples of PFTC (ASMCA) and AFTC (control mixing). They are used later in our work on actuator fault tolerance in Chapter 6, where we present a comparative study between the behavior of these methods and our proposed solution (self tuning sliding mode STSMC). A classification of the existing work on quadrotors and octorotors is presented in Table 1.2.

ref	Types of faults	Types of FDD	UAV
[141], [37], [111], [1], [3]	Loss of effectiveness	Observer-based	Quadrotor
[70]	Loss of effectiveness	Moving Horizon	Quadrotor
[159], [112], [7], [93], [94]	Complete actuator failure	Observer-based	Quadrotor
[8], [10], [9]	Loss of effectiveness	Observer-based	Octorotor
[132], [53], [105], [110]	Complete actuator failure	Observer-based	Octorotor

Table 1.2 – Classification of works on quadrotor and octorotor UAVs

1.7 Wind perturbations on the dynamics of a multirotor UAV

The objective of controlling a drone is to stabilize its orientation and position in the space, successfully track a desired path, and ensure automatic take-off / landing. In order to obtain solutions at the controller level, it is necessary to have a representative model of the uncontrollable effects on the drone [139].

The influence of weather conditions must thus be taken into account in order to find a solution for the control problem of an outdoor multirotor drone. These conditions can be summarized thereafter:

- Changes in temperature which can affect the sensor readings.
- Changes in air density which can partially affect the friction forces acting on the propeller.
- Changes in movements, directions, and speed of the wind which can be considered as external perturbations on the system dynamics.

1.7.1 Wind model

In practice, [139] explains that the influence of temperature, changes in air density is not very important at the operational flight altitude of the multirotors, thus it is sufficient to study only the effect of wind conditions on the system dynamics.

The same paper says that the analysis of the aerodynamic wind effect allows the separation of a constant component $\mathbf{V}_s$ (systematic) that defines the constant wind speed and a variable component $\mathbf{V}_\mu$ (turbulent) that defines the gusts. Thus, the complete wind speed could be considered as a vector defined in the space at the point r by the following formula:

$$\mathbf{V}(r) = \mathbf{V}_s(r) + \mathbf{V}_\mu(r) \tag{1.2}$$

The wind force affecting the UAV can be determined given its speed and direction. The value of this force can be determined using the equation (1.3).

$$F_v = S_e A \mathbf{V}^2(r) \tag{1.3}$$

where F_v is the magnitude of the wind force, S_e is the effective area that is subjected to this force, $A = 0.16$ is the conversion ratio of velocity (m/s) to pressure (N/m^2) and $\mathbf{V}(r)$ is the total wind speed [139]. The influence of the wind force on the UAV is determined according to formula (4.4) by the wind speed and by considering the orientation of the UAV and the direction of the force.

The forces F_x and F_y, which represent the components of the total wind force F_v in the x and y direction respectively, can be calculated given the direction of the wind attacking the UAV using the equation (1.4).

$$\begin{aligned} F_x &= S_{ex} A \mathbf{V}^2(r) \cos(\delta) \\ F_y &= S_{ey} A \mathbf{V}^2(r) \sin(\delta) \end{aligned} \tag{1.4}$$

where S_{ex} and S_{ey} are the effective areas that are subjected to the forces F_x and F_y respectively, and δ represents the wind direction relative to the UAV.

1.7.2 State of the art of resilience to wind perturbations

Since in outdoors environment the drone is exposed to adverse atmospheric conditions, reliable and robust control algorithms are required. However, there exist to our knowledge few works where wind effects on these systems are quantified. In [120], the authors developed two methods to estimate the wind speed and its direction. In a direct approach, wind information are estimated using a wind sensor (anemometer) mounted on top of the vehicle. In an indirect approach, the attitude data from the drone are used in the estimation. However, the wind effect is considered as only due to the air frame drag which does not reflect real flight conditions. The problem of wind estimation was also addressed in [155], where an extended state observer is developed to identify wind disturbances, but only indoor experiments where conducted to validate this method. A linear observer for the estimation of wind disturbances is presented in [106]. Real-time outdoor experiments where conducted using a GPS receiver to test the proposed method in real flight conditions. Nevertheless, the authors explain that it can only be used near the equilibrium point, i.e. hovering, since it is based on a linear model of the vehicle.

Indeed, the design of a robust control law against disturbances due to wind is a difficult problem [6], especially when tracking aggressive trajectories. Thus two approaches are oftenly used: first the implicit modeling of the forces of wind on the drone and its integration in the expression of the control law, and second the use of a robust controller to tolerate wind effects and other uncertainties (such as model uncertainties).

Among the first approach, Sydney et al. [143] carried out an estimation process of the wind effect on a quadrotor and used it to design a control law capable of countering its disturbances. The authors in [145] investigated the wind effect on a quadrotor model, and showed that it can be viewed as an oblique flow approaching the rotors based on the blade-element momentum theory. Huan et al. [67] took into account the aerodynamic effect by proposing a law of aggressive command for a quadrotor using the blade flapping approach. In [88], an active rejection of wind disturbance using an approximate feedback linearization of the model dynamics is proposed and validated. However, such a controller based on feedback linearization is known to be very sensitive to sensor noise, since up to three-order derivatives of the states are included in the inputs. The main issue in all these

different studies is that only simulations are presented to validate the effectiveness of the proposed methods.

In contrast, other studies have addressed the problem of wind perturbations on the UAVs providing experimental validations for the proposed strategies, such as in [51] where three types of disturbances including payloads, rotors failures, and the wind generated by a fan are chosen separately to verify the effectiveness of the controller in each situation.

Among works of the second approach, [79] and [117], propose robust control laws with adaptive gains to ensure the stability of the system. In [57], the authors address flight control in presence of wind-gust disturbances during trajectory tracking with a 6DOF nonlinear dynamics model separated into two subsystems (dynamic and kinematic). A hierarchical controller is used to stabilize the under-actuated subsystems using the adaptive control techniques to deal with slow and fast varying wind conditions. At the same time, a backstepping technique is used to stabilize the inner loop heading dynamics.

The authors of [153] developed an adaptive robust controller based on the sliding mode algorithm to deal with payload variation and wind gust disturbances. To quantify the wind impact, they implemented a propeller momentum drag and wind gust model including forces and moment disturbances.

To our knowledge, there exists only few works which address the problem of wind perturbations and propose experimental. We think that most researchers working on UAVs assume that robust controllers are sufficient to deal with such problem. However, from our experiments in chapter 4, we found that only the complex robust controllers (such as second-order sliding mode controllers) are able to compensate most of the external wind perturbations. To solve this problem, we propose as an alternative solution an active disturbane rejection technique through online estimation of wind perturbations.

1.8 Fault tolerance for multi-sensor data fusion

We introduce here the field of data fusion, which plays a fundamental role for multi-sensor perception systems, and we particularly focus on the Kalman filter technique that allows to fuse data from different sensors and reduce noise. We then present a state of the art for fault tolerant techniques in perception.

Data Fusion [26] consists in joining or merging information that stems from several sources and exploiting that information in various tasks such as answering questions, making decisions, estimating numerical values, etc. These sources can be physical sensors observing the actual situation and providing different information on the possible events. Other definitions of the data fusion concept can be found in [29]. Multi-sensor data fusion combines several sensors measurements to form a better and easier to use representation of the observed environment: the world model. It seeks to take advantage of all available information on a given problem to counter each sources imperfections. Generally three major theoretical frameworks are used to implement a data fusion mechanisms: the probability theory, the possibility theory [55, 54], and the belief function theory [48]. In our work we focus on the probabilistic framework, particularly the Kalman filter method [78].

Data fusion has a long history in robotic field. It has been a significant focus during

the 80-90s in military applications [63, 62], and remains relevant in this area. Fusion methods have been adapted and developed for robotics applications (such as autonomous navigation, target tracking, and localization). In [160] a belief function theory data fusion technique is used and adapted to the robot localization problem using ultrasound images. As navigation is fundamental for mobile robots, Kalman filters have been used in that function for a long time [41, 59]. As the original Kalman filter can only be applied to linear systems, an extended Kalman filter (EKF) has been proposed for non linear systems. This EKF has been successfully implemented for robot position estimation in [47], and [82].

1.8.1 Data fusion using Kalman filter

The Kalman filter assumes that the system states and the measurements of the sensors can be described by a linear dynamic system. This dynamic system is divided into two parts:

- The linear model which describes the evolution of the system states over time.
- The measurement model which describes how the measurements are related to the states.

Thus, the Kalman filter assumes that the system can be represented by linear equations. When the system is nonlinear, we can use linearization techniques to transform the problem into a linear problem, using the Extended Kalman filter (EKF). A good introduction to the topic can be found in [157]. A more detailed description of the concept, derivation and properties is given in [58].

1.8.1.1 Kalman filter for linear systems

The Kalman filter, also known as linear quadratic estimator, is an optimal estimator for discrete linear system of the form:

$$x_{k+1} = A_k x_k + w_k \tag{1.5}$$

where $x_{k+1} \in \mathbb{R}^n$, $x_k \in \mathbb{R}^n$ represent the system states respectively at the instants $k+1$ and k, $A_k \in \mathbb{R}^{n \times n}$ is the transition matrix between $k+1$ and k and $w_k \in \mathbb{R}^n$ is the noise vector in the states model.

On the other hand, the observation model which describes the measurement of the sensors is describe by:

$$z_k = H_k x_k + v_k \tag{1.6}$$

with $z_k \in \mathbb{R}^p$, $H_k \in \mathbb{R}^{p \times n}$ and $v_k \in \mathbb{R}^p$ are respectively the measurement vector at instant k, the observation matrix and the noise vector in the observation model.

The process noise w_k and the measurement noise v_k are assumed to be independent random variables with Gaussian probability density functions and zero mean value. The normal probability distributions p are as follows:

$$p(w) \backsim N(0, Q), Q = diag(\sigma_{w1}^2, \sigma_{w2}^2, ...,) \tag{1.7}$$

$$p(v) \backsim N(0, R), Q = diag(\sigma_{v1}^2, \sigma_{v2}^2, ...,) \tag{1.8}$$

with σ^2 being the variance of the corresponding noise distribution.

Kalman filter process

The first step is to determine the predicted state $\hat{x}_k^-$ at time k, using the previous corrected state $\hat{x}_{k-1}$, and the associated error covariance matrix P_k^- using:

$$\begin{aligned} \hat{x}_k^- &= A_k \hat{x}_{k-1} \\ P_k^- &= A_k P_{k-1}^+ A_k^T + Q_k \end{aligned} \tag{1.9}$$

The second step (correction step) is to correct the predicted states and the covariance matrix using the measurements from the sensors z_k when they are available. The predicted state can be corrected using the innovation (or residual) $\hat{S}_k$ and the computed Kalman gain k_k. Thus, the corrected state $\hat{x}_k$ and its associated covariance matrix P_k^+ can be estimated using the following equations:

$$\begin{aligned} \hat{S}_k &= z_k - H_k \hat{x}_k^- \\ \hat{S}_k &= H_k P_k^- H_k^T + R_k \\ k_k &= P_k^- H_k^T \hat{S}_k^{-1} \\ \hat{x}_k &= \hat{x}^- + k_k \hat{S}_k \\ P_k^+ &= P_k^- k_k H_k P_k^- \end{aligned} \tag{1.10}$$

The innovation $\hat{S}_k$ represents the difference between the predicted states and the real measurement from the sensors.

1.8.1.2 Extended Kalman filter EKF for nonlinear systems

For the estimation of a nonlinear system, several extended versions of the Kalman filter exist. A widely used approach is to linearize the system dynamics in every step around the a priori estimation x_k^-, and proceed as for a linear system. This approach is known as the EKF. The fundamental imperfection of the EKF, as pointed out in [96], is that the distributions of the various random variables are no longer a normal distribution, after undergoing their respective non-linear transformations. Thus, the optimality of the estimation is only approximated by linearization. The stochastic system equations from (1.9) and (1.6) are now generalized to the nonlinear case as:

$$x_{k+1} = f(x_k, w_k) \tag{1.11}$$

again with the state vector $x_k \in \mathbb{R}^n$, and the observation model is given by:

$$z_k = h(x_k, v_k) \tag{1.12}$$

with the measurements vector $z_k \in \mathbb{R}^m$.

Extended Kalman filter process

By deriving the Jacobian matrix of the partial derivatives of $f(x_k, w_k)$ and $h(x_k, v_k)$ with respect to the state x_k and the noise vectors w_k and v_k, we obtain the linearized

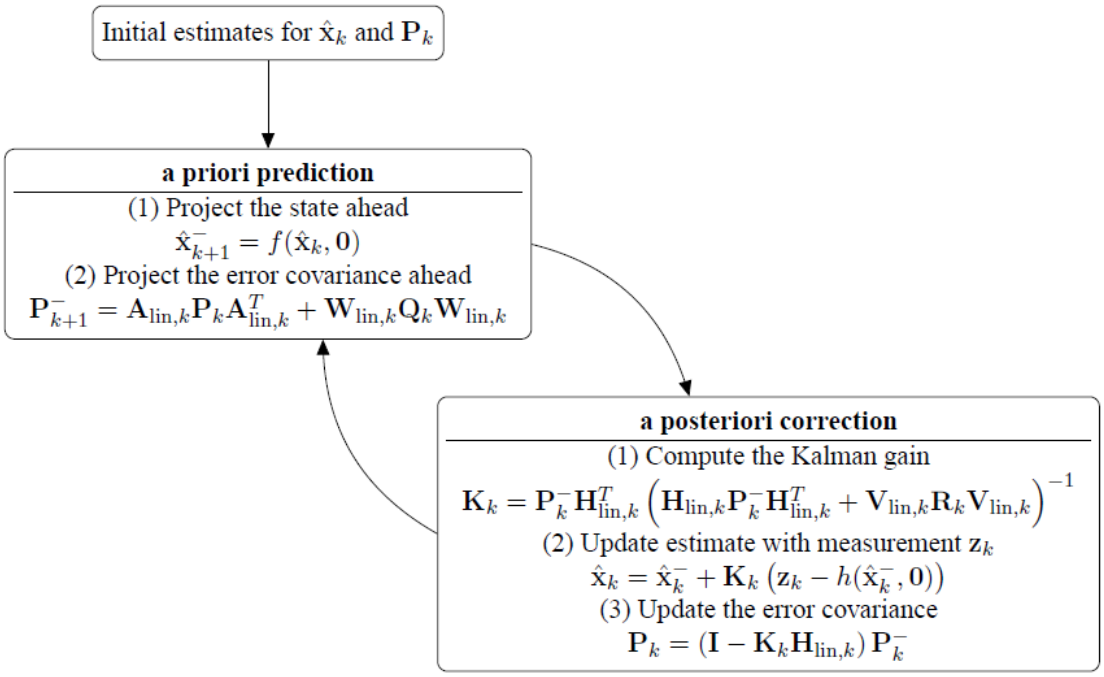


Figure 1.7 – Data flow of the Extended Kalman filter operation

approximation of the matrices:

$$\begin{aligned} A_k &= \left(\frac{\partial f(x_k, w_k)}{\partial x_k}\right)^T\Big|_{\hat{x}_k^-} \\ W_k &= \left(\frac{\partial f(x_k, w_k)}{\partial w_k}\right)^T\Big|_{\hat{x}_k^-} \\ H_k &= \left(\frac{\partial h(x_k, v_k)}{\partial x_k}\right)^T\Big|_{\hat{x}_k^-} \\ V_k &= \left(\frac{\partial h(x_k, v_k)}{\partial v_k}\right)^T\Big|_{\hat{x}_k^-} \end{aligned} \tag{1.13}$$

The EKF is implemented as shown in Figure 1.7, and a detailed process can again be found in [96].

1.8.2 Fault tolerance mechanisms for data fusion

The risk of software and hardware faults increases, in terms of sensor failures and processing failures, due to the increasing number of sensors and the underlying data fusion mechanisms [4]. Hence, there is a need to apply a fault tolerant strategy to overcome these issues and detect any failures, and to ensure more reliable performance outcomes with respect to autonomous systems.

To our best knowledge, there exist only few works in the literature regarding fault tolerance in the field of data fusion. The approaches that can be found use the duplication and comparison techniques to tolerate physical faults [18]. These approaches can be categorized into two categories: duplication based on an analytical model, and duplication based on hardware redundancy.

1.8.2.1 Model Based approaches

The model based approaches, also known as analytical redundancy approaches [72], determine functional relationships between the measured states through a mathematical model. This mathematical model can either be developed from physics analyses or obtained from the measurements directly. Subsequently, a residual r_k is then generated between the actual sensor output y_k and the estimated modeled output $\hat{y}_k$, i.e.,

$$r_k = y_k - \hat{y}_k \tag{1.14}$$

A residual zero-mean, that is, $\sum_k \frac{r_k}{k} = 0$ means no fault and the mean deviation from zero means the existence of a fault. A Nadaraya-Watson statistical estimator and a priori observations are used in [158] to validate sensor measurements. Residuals or innovations generated by Kalman filter (KF) were used in [52, 68, 43] to detect faults: statistical tests on residual whiteness, mean, and covariance identify the faults. In [43], a failure detection approach for a KF-based GPS integrity monitoring system was proposed. The idea is to process subsets of the measurements by an auxiliary KF component and to use the estimate generated as a reference for detection of failures. The KF prediction was used as a reference for detecting inconsistencies in the measurement of sensors in [77]. An adaptive sensor/actuator detection and isolation scheme for a Unmanned Aerial Vehicle (UAV) based on KF has been proposed in [61]. The detection of system failure in this method is done by applying statistical tests on the KF's innovation covariance. In [66], this method is used to improve the accuracy of personal outdoor positioning systems. Common tools for assessing residual statistical characteristics are generalized probability ratio tests [71], chi-square tests [150], and multiple hypothesis tests [69]. Some authors also proposed approaches based on Extended KF (EKF) [107, 46] and Unscented KF (UKF) [134] with the objective of detecting inconsistencies in the perception of non-linear systems. Multi-sensor data fusion for multi-robot system based on Kullback-Leibler Divergence (KLD) was proposed in [5]. The method calculates the KLD between an Information Filter a priori and a posteriori distributions and uses the threshold of the Kullback-Leibler Criterion to detect and remove suspicious sensor data.

1.8.2.2 Redundancy based approaches

In data/hardware/sensor redundancy based approaches, two or more sensors measure the same critical state and then detect as well as isolate the faulty sensors by consistency checks and majority voting [72]. For example, in [81], the authors proposed a voter-based fault detection system for multiple sensors subsystems of inertial navigation system (INS), GPS, attitude sensor and heading reference system (DAHRS). A sensor voting algorithm was presented in [44] to manage three redundant sensors.

In [19], the authors introduced an approach for tolerating faults using multi-sensor data fusion. This approach is based on the method of duplication/comparison, which can offer detection and diagnosis of faults in a data fusion mechanism. Fault tolerance is done through error detection and system recovery. Error detection helps to detect the erroneous state of the system before the propagation of the error can cause the failure of the system. System recovery allows an error-free state to be substituted in place of an erroneous state.

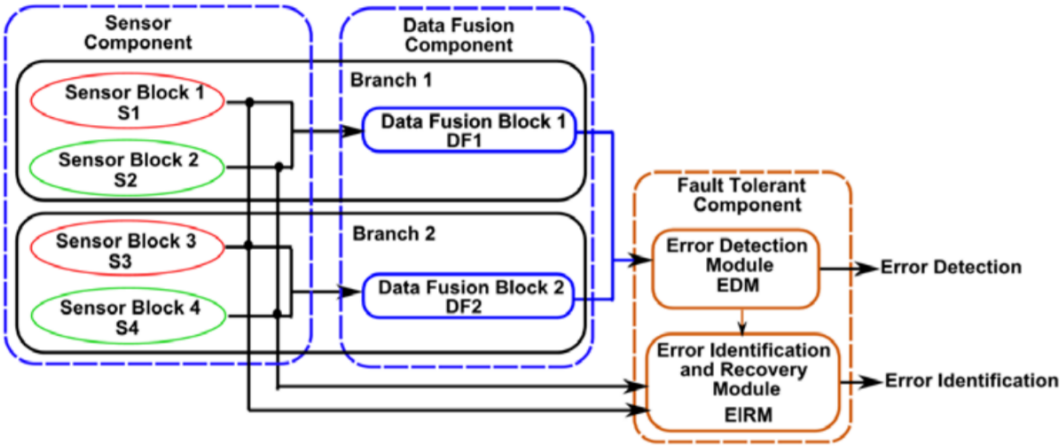


Figure 1.8 – Duplication-comparison architecture for fault tolerance in multi-sensor perception

The Figure 1.8 illustrates the architecture for fault tolerance using duplication-comparison and multi-sensory perception. This architecture will be further discussed in Chapter 6.

Based on redundant multi-sensor navigation systems, inconsistency detection for hypersonic cruise vehicles (HCVs) was proposed in [156]. Two sensor blocks were involved: the first block consists of an inertial navigation system (INS) and a GPS, while the second block consists of an INS and a navigation system. The method uses chi-square test and sequential probability ratio test to detect inconsistencies in each block's local sensor estimates before sending their data to a central node for a global estimate. In another work, an application for failure detection and isolation on redundant aircraft sensors based on fuzzy logic and majority voting was proposed in [24]. A method for detecting spurious sensor data based on the Bayesian framework without any prior information was proposed in [84]. This method adds a term to a Bayesian probabilistic approach that increases the a posteriori distribution if measurement from one sensor is inconsistent with the other.

1.9 Summary

In this chapter, we presented a state of the art in fault-tolerant control for unmanned aerial vehicles:

- First, we introduced some fundamental concepts and principles of dependability: attributes, threats and means. The notions of faults, errors and failures used throughout the thesis are defined and faults are classified according to different criteria. A special focus was laid on the concept of robustness and FTC, where a detailed description of the techniques employed in the robotic field is given.

- Then we described the classifications of FTC systems as passive fault tolerant control (PFTC) systems and active fault tolerant control (AFTC) systems. PFTC strategies consists in masking the faults by using a robust and reliable controller able to deal with all expected fault without the need to detect and identify them. In AFTC strategies, the fault tolerance process is carried out via three successive steps:

(1) Error detection, using a fault detection unit, (2) fault diagnosis, which identifies the occurrence and the type of a fault after the detection, (3) system recovery.

- Then, We presented a state of the art regarding wind perturbations and their effects on an UAV's dynamics. Many research articles focuses on passive strategies to deal with wind perturbations using adaptive robust controllers, and only few works considered active solutions with online estimation of the wind perturbations.

- Finally, we introduced the works done in the field of fault tolerance for multi-sensors data fusion, where we presented the Kalman filter theory and fault tolerant approaches to deal with sensor and data processing failures. The majority of scientific research about fault tolerance of UAV systems focuses on the problem of actuator failures and to our knowledge, there exist only few works regarding fault tolerance in the field of perception and data fusion. The approaches that can be found on data fusion focuses on duplication and comparison techniques and can be categorized into two categories: duplication based on analytical model and duplication based on hardware redundancy.

Dynamic Model and parameters estimation

Contents

This chapter outlines the dynamic modeling of multirotor UAVs. First, the earth and body frames and the transformations of coordinates from each frame to the other are introduced. These coordinates are used to define the position and orientation of the multirotor. Second, the forces and moments acting on the UAV are detailed and the kinematics and dynamics expressions are derived using the Newton-Euler formalism. Finally, the complete and simplified model of a coaxial octorotor is presented and its parameters are identified using real data sets extracted from static tests on the motors and propellers systems.

2.1 Modeling formalism and Assumptions

A multirotor UAV system is a highly nonlinear, multivariable, strongly coupled, unstable system. In practice, there exists several configurations of multirotor UAVs, such as: quadrotors, tricopters, hexarotors, coaxial octorotors... In this thesis, only quadrotors and a coaxial counter-rotating octorotor are considered to validate our contributions. Note that in coaxial octorotors, the actuators are stacked in counter-rotating pairs so as to resemble a quadrotor as in Fig. 2.3.

In general, a multirotor UAV is a 6-Degrees Of Freedom (DOF) symmetric rigid body which includes several actuators that are attached at the ends of the system's arms and on which propellers are fixed. Each motor can be controlled individually via an electronic speed controller unit (ESC), thus modifying the states (position and orientation) of the drone and allowing it to maneuver into the three dimensional space.

In the literature, two main formalisms are used to derive the dynamic model of a multirotor UAV:

- The Euler-Lagrange formalism
- The Newton-Euler formalism

The Newton-Euler formalism approach is expressed in terms of the body frame coordinates and then derived in terms of the earth frame coordinates using kinematics transformation. The Euler-Lagrange formalism, instead, directly uses generalized coordinates in the frame coordinates, but the equations of motion requires the formulation of the kinetic and potential energies of the system under this frame, which leads to a more complex formulation. However, the final result is equivalent in both approaches, but written with different notations. In [30], a detailed model of the quadrotor is given including all rigid body dynamics, aerodynamic forces and gyroscopic effects. Both methods, Euler-Lagrange and Newton-Euler formalisms, are consistent for the description of the dynamics of the aerial vehicle. However, it has been noted, as previously explained, that the Newton-Euler formalism is easier to understand. This formalism is chosen in this thesis to model the quadrotor and the coaxial octorotor dynamics.

In this work, some assumptions are made to facilitate the modeling process:

- **Assumption MOD1** The structure of the vehicle is supposed to be rigid and symmetrical. Specifically, the center of gravity is supposed to be fixed and the actuator's position are symmetrical with respect to the vehicle axes, which will allow much simpler equations concerning the forces applied on the system.
- **Assumption MOD2** The motor dynamics are ignored. This will allow to not consider, the equations between the motor's rotational speed and the feeding current and voltage.
- **Assumption MOD3** The center of gravity and the body-fixed frame origin are assumed to coincide. This will allow the off-diagonal terms in the inertia matrix to be zero.
- **Assumption MOD4** The propellers are supposed to be rigid. Thus, we ignore the blade flapping (the up and down movement of a rotor blade).
- **Assumption MOD5** The thrust and the drag are proportional to the square of the rotors speed. This will be useful in the identification procedure of the motor parameters.

2.2 Modeling using Newton-Euler formalism

For multirotor UAVs, there are several configurations including tricopter, quadrotor, hexarotor and octorotor. The following Figure 2.1 shows the UAVs configurations that are supported by open source autopilots, where the cyan colored motor refers to a motor rotating in clockwise sense and the grey motor refers to a motor rotating in a counterclockwise sense. In this work, we will focus on the quad X and X8 configurations.

The quadrotor and the coaxial octorotor used in this thesis are modeled using the conventions shown in Figures 2.2 and 2.3. For the octorotor, the motors spinning in the

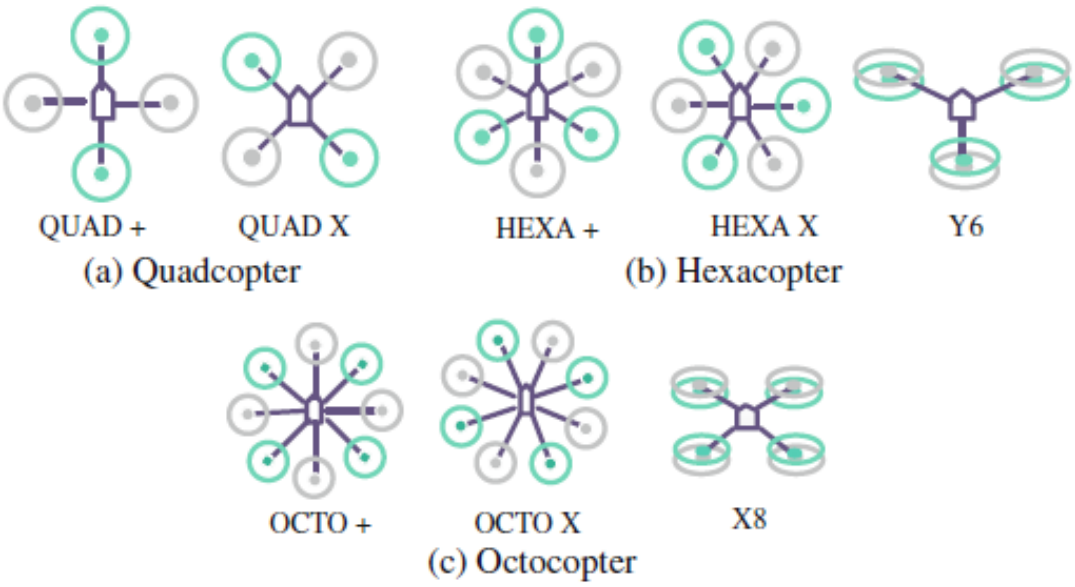


Figure 2.1 – Multirotor configurations

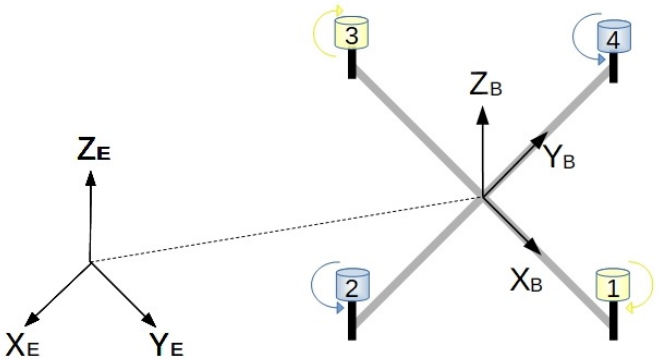


Figure 2.2 – Quadrotor's configuration

counter-rotating direction are the bottom front left, bottom rear right, top front right and top rear left motors (M_7, M_3, M_2 and M_6) with angular velocities ω_7, ω_3, ω_2 and ω_6, while the remaining four motors (M_1, M_4, M_5 and M_8) spin in the clockwise direction with angular velocities ω_1, ω_4, ω_5 and ω_8.

In this section, we will first introduce the frames used to study the motion of the multirotor, then we will describe the Euler angles with respect to the orientation of the vehicle, finally we will describe the derivation of the rotation matrix used to express the vectors in different frames.

2.2.1 Definitions of the Frames

In order to study the motion of the multirotor UAV, we need first to consider two Cartesian frames of reference:

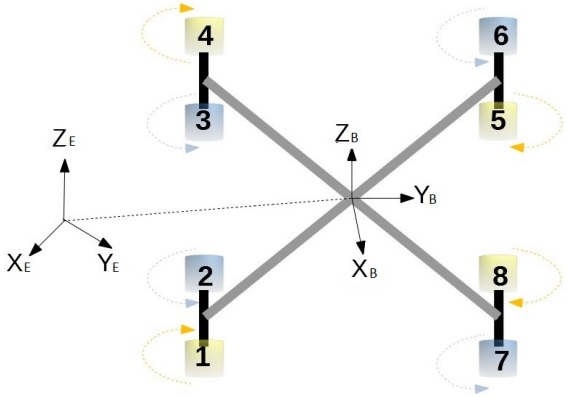


Figure 2.3 – Octorotor's configuration

- The earth frame R_E $\{X_E, Y_E, Z_E\}$
  This frame is used to define the starting location of the mission, or *home*. It will be used to derive the translational equations of motion. In this frame, X_E points north, Y_E points East and Z_E points upward.

- The body frame R_B $\{X_B, Y_B, Z_B\}$
  This frame is fixed to the UAV's body with its origin considered at the center of gravity of the UAV. The positive X_B-axis of the body frame points to the front direction of the UAV. The positive Y_B-axis is pointing to the left and the Z_B-axis is perpendicular to the X_B and Y_B axes, pointing upward. As we are using the quad X and X8 configurations, the X_B and Y_B axes are found between the UAV's arms rather than along them.

2.2.2 Euler angles

The orientation of the vehicle is determined by the Euler angles. They are known as the yaw ψ (or heading), pitch θ, and roll ϕ angles.

The transformation between the earth and body frames can be realized by three successive Euler rotations.

Figure 2.4 presents the three successive Euler rotations needed to transform the earth frame into the body frame. This transformation is done by applying a rotation around each of the three Cartesian axes successively, following the right-hand rule. The first rotation around the z-axis by an angle ψ transforms the inertial coordinate frame into an Intermediate Frame 1. This is followed by a rotation around the new y_1 axis by an angle θ to get the second intermediate frame 2. Finally, the last Euler angle ϕ defines the rotation around the new x_2 axis to obtain the body frame. As we will see in Section 2.2.3, a rotation matrix is associated to each transformation. The inverse transformation can be done by calculating the inverse of each rotation matrix.

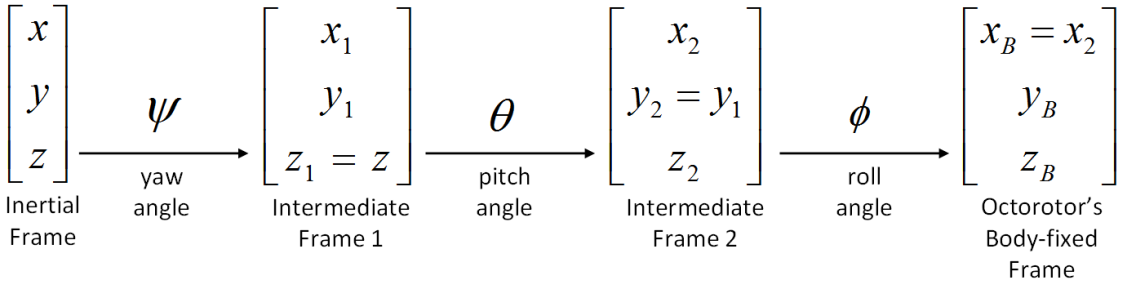


Figure 2.4 – Transformation between Earth and body-fixed frame using Euler angles

2.2.3 Rotation Matrix

Any vector v expressed in the earth frame is transformed into a vector v_B expressed in the body frame by mean of the rotation matrix $R_{E\to B}$. The equation between these vectors is given as follows:

$$\mathbf{v}_B = R_{E\to B}\mathbf{v} \tag{2.1}$$

This rotation matrix is obtained by multiplying each of the three relative matrices associated with each Euler angle.These rotation matrices are respectively given by:

$$\begin{aligned} R_\psi &= \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \\ R_\theta &= \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \\ R_\phi &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{pmatrix} \end{aligned} \tag{2.2}$$

The rotation matrix, also called *Direction Cosine Matrix* (DCM), is then expressed as:

$$R_{E\to B} = R_\phi R_\theta R_\psi \tag{2.3}$$

or

$$R_{E\to B} = \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{pmatrix} \tag{2.4}$$

It is worth to note that the DCM is an orthogonal matrix. Thus the rotation matrix $R_{B\to E}$ that transforms a vector $\mathbf{v}_B$ expressed in the body frame into a vector expressed in the inertial earth frame is obtained as follows:

$$R_{B\to E} = (R_{E\to B})^{-1} = (R_{E\to B})^T \tag{2.5}$$

Following the same manner, given the angular velocity vector $\omega = \begin{bmatrix} p & q & r \end{bmatrix}^T$, where p, q and r are the angular velocities around the X_B, Y_B and Z_B axes respectively, we can

obtain the Euler rates $\dot{\eta} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^{\mathbf{T}}$ by means of a transformation matrix $\mathbf{T}$:

$$\dot{\eta} = \mathbf{T}\omega \tag{2.6}$$

where

$$\mathbf{T} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \tag{2.7}$$

For further information about the derivation of the $\mathbf{T}$ matrix, please refer to [31].

2.3 Aerodynamic Forces and Moments

The multirotor UAV is considered as a rigid body. The dynamics under the external forces that are applied to the center of mass are described in this section. All the forces vectors are then transformed into the earth frame and the dynamic model is finally obtained using the Newton-Euler formalism. Let:

- $\mathbf{F}$ be the vector of forces acting on the multirotor UAV in the body frame.
- τ be the vector of moments acting on the multirotor UAV in the body frame.
- $\mathbf{v}$ be the vector of linear velocities along x, y and z axes in the earth frame.
- ω be the vector of angular velocities p, q and r in the body frame.
- G, m, and $\mathbf{I}$ be the gravity constant, the mass of the vehicle and the inertia matrix of the vehicle respectively. I is a diagonal matrix because of the symmetrical structure of the system, expressed as:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{2.8}$$

  where I_{xx}, I_{yy}, I_{zz} are the moments of inertia around the axes X_B, Y_B, Z_B respectively.

In this section, we will describe the forces and the moments acting on the multirotor, then we will derive the equations governing the motion of the multirotor UAV.

2.3.1 Forces Modeling

The different forces acting on the drone are given below:

- Force of Gravity: This force is directed in the -z direction. It is expressed in the earth frame as:

$$[F_G]_E = \begin{bmatrix} 0 \\ 0 \\ -mG \end{bmatrix} \tag{2.9}$$

  Since its point of application is the center of gravity, this force does not generate any moment.

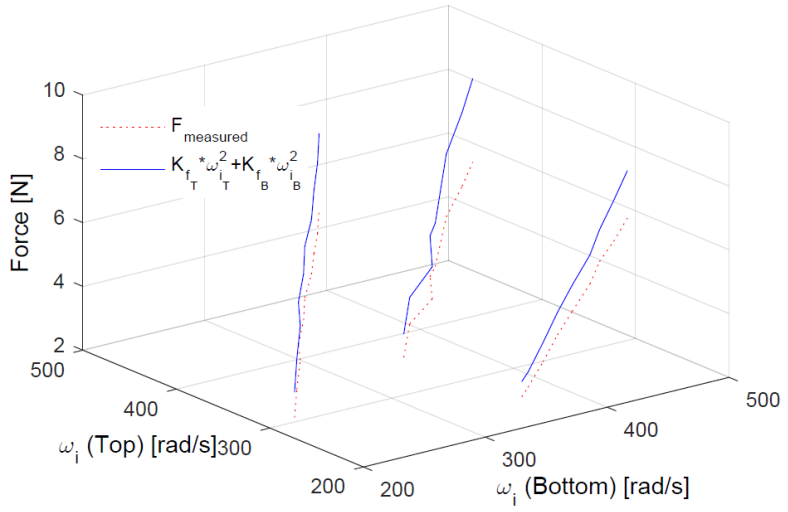


Figure 2.5 – Top Motor: Aerodynamic force as a function of the motor speed

- Thrust Force: Each motor/propeller produces an upward thrust force $\mathbf{f}_i$ where $i = 1,2,...4$ in case of a quadrotor and $i = 1,2,...8$ in case of an octorotor, directed along the positive z_B-axis and at a distance d from the center of mass.

$$\mathbf{f}_i = \begin{bmatrix} 0 \\ 0 \\ f_i \end{bmatrix} \tag{2.10}$$

The expression of the thrust can be derived using the actuator disk and blade elements theory [30]:

$$f_i = C_T \rho A r^2 \omega_i^2 = K_f \omega_i^2 \tag{2.11}$$

where C_T is the thrust coefficient, ρ is the air density, A is the rotor disk area, r is the rotor radius, and ω_i is the rotational speed of the motor i. In this thesis, the gain K_f will encompasses most of the thrust force's variables (as seen in equation 2.11) and will be identified for each experimental platform in Section 2.4.

In the case of the coaxial octorotor, we must also take into account the interaction between the two coaxial rotors which causes a loss of net rotor aerodynamic efficiency as mentioned in [124]. Thus, the combined force produced by two coaxial rotors is given as:

$$f_{ij} = \alpha_{ij}(f_i + f_j)S \tag{2.12}$$

$\alpha_{ij} = 1.8$ is the coefficient of loss of aerodynamic efficiency due to the aerodynamic interference between the upper and lower rotors in a pair of coaxial rotors. $S = 1 + \frac{S_S}{S_{prop}}$ represents the shape factor of the propellers, with S_S referencing the propellers surface and S_{prop} the surface of the circle that the propeller would make when rotating.

Indeed, the total thrust f_{12} remains less than the sum of individual thrusts of each rotor f_1 and f_2 as demonstrated in Fig. 2.5.

This figure shows the data collected from different static tests that were carried out when the top and bottom motors were rotating at the same speeds and at different speeds.

The total thrust value F_u produced by the propellers of the quadrotor and the octorotor can be expressed in the body frame as:

◇ For a quadrotor:

$$F_u = f_1 + f_2 + f_3 + f_4 \tag{2.13}$$

◇ For a coaxial octorotor:

$$F_u = f_{12} + f_{34} + f_{56} + f_{78} \tag{2.14}$$

And using the DCM matrix, we can express the total thrust generated by each system in the earth frame as follows:

$$[\mathbf{F}_u]_E = R_{B\to E} \begin{bmatrix} 0 \\ 0 \\ F_u \end{bmatrix} \tag{2.15}$$

- Drag Forces: The drag forces on the vehicle body are due to the friction with the wind during horizontal motion. They are expressed in the earth frame as follows:

$$\begin{array}{rcl} F_{f_x} & = & \frac{1}{2} C_x A_c \rho \dot{x} |\dot{x}| \\ F_{f_y} & = & \frac{1}{2} C_y A_c \rho \dot{y} |\dot{y}| \end{array} \tag{2.16}$$

where C_x, C_y are friction coefficients and A_c is the fuselage area or the area of the drone exposed to wind.

2.3.2 Moments Modeling

The different moments acting on the multirotor are analyzed below.

- Propeller Gyroscopic Effect: The high rotation speed of the propeller blades causes a gyroscopic effect, which affects the motion of the vehicle when the propellers do not spin with the same velocity. The gyroscopic torque is expressed as:

$$\tau_g = J_R \left(\omega \, \mathrm{x} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \Omega_r \tag{2.17}$$

where $\Omega_r = \omega_1 + \omega_3 - \omega_2 - \omega_4$ for a quadrotor and $\Omega_r = \omega_2 + \omega_3 + \omega_6 + \omega_7 - \omega_1 - \omega_4 - \omega_5 - \omega_8$ for a coaxial octorotor. Ω_r is the overall residual propeller speed from the unbalanced rotor rotation, and $\omega_1 ... \omega_8$ are the motors speeds. J_R is the propellers inertia and ω is the vehicle angular velocity expressed in the body frame.

This effect is neglected most of the time since the propellers inertia is small compared to the body inertia.

- Drag Moment: The drag moment around the rotor shaft is caused by the aerodynamic forces acting on the blade elements. This moment is directed along the Z_B-axis.

$$\boldsymbol{\tau}_i = \begin{bmatrix} 0 \\ 0 \\ \tau_i \end{bmatrix} \tag{2.18}$$

The expression of the drag moment generated by the motor i is obtained using the actuator disk and blade elements theory, similarly to the thrust force:

$$\tau_i = C_Q \rho A r^3 \omega_i |\omega_i| = K_t \omega_i |\omega_i| \tag{2.19}$$

where C_Q is the drag coefficient. In this thesis,the gain K_t encompasses most of the drag moment's variables, as seen in Equation 2.19, and will be identified in Section 2.4. The drag moment τ_i is proportional to the square of the motor i speed. Its sign depends on the sense of rotation of the corresponding motor i.

- Torques: the actuators generate roll and pitch torques depending on their positions with respect to the X_B and Y_B axis. Let τ_ϕ and τ_θ represent respectively the roll and pitch torque. The expressions of these torques will be given in the next section for the case of a quadrotor and an octorotor.

2.3.3 Equations of Motion

The dynamical model of the translation motion of the UAV can be obtained using Newton's law. We can write:

$$m\dot{\mathbf{v}} = R_{B\to E}.\mathbf{F} \tag{2.20}$$

where $\mathbf{F}$ in the vector representing the sum of all the forces presented in subsections 2.3.1. Therefore, we obtain:

$$\begin{cases} m\ddot{x} &= (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)u_f - \frac{1}{2}C_x A_c \rho \dot{x}|\dot{x}| \\ m\ddot{y} &= (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)u_f - \frac{1}{2}C_y A_c \rho \dot{y}|\dot{y}| \\ m\ddot{z} &= (\cos\phi \cos\theta)u_f - mG \end{cases} \tag{2.21}$$

Using Euler's law, we can model the dynamic of the UAV's attitude $\boldsymbol{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^T$ by the following:

$$I\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times I\boldsymbol{\omega} + \boldsymbol{\tau} \tag{2.22}$$

or

$$\begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{2.23}$$

where τ is the vector of moment acting on the multirotor. Now a simplification is made by setting $[\dot{\phi}, \dot{\theta}, \dot{\psi}] = [p, q, r]$. This assumption holds true for small angles of movement.

So, the full dynamic model of the UAV in the inertial frame is:

$$\begin{cases} m\ddot{x} &= (\cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi)u_f-\frac{1}{2}C_xA_c\rho\dot{x}|\dot{x}| \\ m\ddot{y} &= (\cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi)u_f-\frac{1}{2}C_yA_c\rho\dot{y}|\dot{y}| \\ m\ddot{z} &= (\cos\phi\cos\theta)u_f-mG \\ I_{xx}\ddot{\phi} &= \dot{\theta}\dot{\psi}(I_{yy}-I_{zz})-J_r\dot{\theta}\Omega_r+\tau_\phi \\ I_{yy}\ddot{\theta} &= \dot{\phi}\dot{\psi}(I_{zz}-I_{xx})+J_r\dot{\phi}\Omega_r+\tau_\theta \\ I_{zz}\ddot{\psi} &= \dot{\phi}\dot{\theta}(I_{xx}-I_{yy})+\tau_\psi \end{cases} \tag{2.24}$$

And the virtual input vector is given by:

- In case of a quadrotor:

$$\begin{bmatrix} u_f \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} f_1+f_2+f_3+f_4 \\ l(f_4-f_2) \\ l(f_3-f_1) \\ (\tau_1+\tau_3)-(\tau_2+\tau_4) \end{bmatrix} \tag{2.25}$$

- In case of a coaxial octorotor:

$$\begin{bmatrix} u_f \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} f_{12}+f_{34}+f_{56}+f_{78} \\ l\frac{\sqrt{(2)}}{2}(f_{78}+f_{56}-f_{34}-f_{12}) \\ l\frac{\sqrt{(2)}}{2}(f_{34}+f_{56}-f_{78}-f_{12}) \\ (\tau_2+\tau_3+\tau_6+\tau_7)-(\tau_1+\tau_4+\tau_5+\tau_8) \end{bmatrix} \tag{2.26}$$

where l is the length of the arm.

2.4 Experimental Platforms

In this section, we present the different experimental drones used to validate our contributions. The first UAV is the $Modulo-X_8$ octorotor (Fig. 2.6), the second UAV is the *DJI S*500 quadrotor (Fig. 2.9), and the third UAV is the *Tarot* 650 quadrotor (Fig. 2.10). All these systems will be used as experimental platforms for our experimental validations. We start by introducing the hardware for each UAV, including the sensors and actuators, and the experimental setups (Figures 2.7 and 2.8). Then we present the work that has been done on these drones for the identification of modeling parameters corresponding to the gains K_t and K_f presented in the previous section.

2.4.1 $Modulo-X_8$

The hardware used for the $Modulo-X_8$ includes:

- **Mechanical Structure**:
  The mechanical basis of the $Modulo-X_8$ octorotor prototype was designed, using the Catia software, developed and assembled at the Heudiasyc laboratory. The frame is made from carbon fiber material and it accommodates the control electronics, the power distribution board, the flight sensors and the power supply.

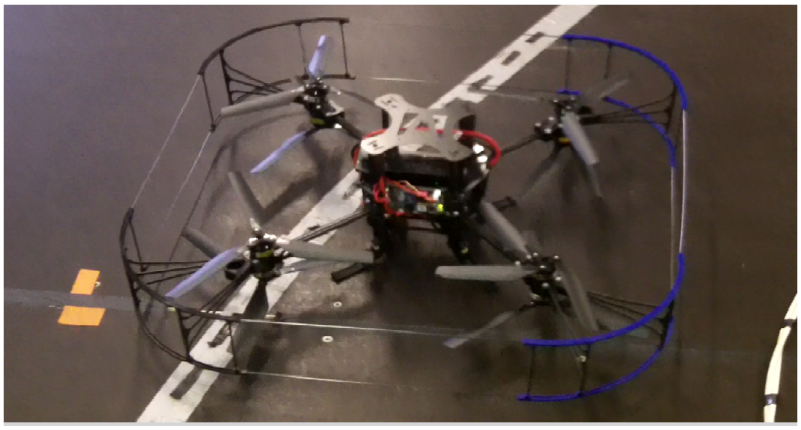


Figure 2.6 – *Modulo* – X_8 coaxial Octorotor

The structure consists of four arms equally spaced, each one carrying at its extremity two brushless coaxial counter-rotating DC motors and the associated electronic speed controllers (ESCs).

- **Development Framework**:

  The software used to write applications for the octorotor is called FL-AIR, it has been produced at the Heudiasyc laboratory to ease development and integration of research algorithms. FL-AIR is based on Linux and is compatible with real time features with Xenomai.

- **Propulsion System**:
  The propulsion system of the platform consists of eight brushless DC (BLDC) motors with electronic speed controllers (ESC) and propellers. The motors used are BL2827-35 driven by BLCTRLV2 controllers which provide speed, current, temperature and voltage measurements.

- **Central Processing Unit (CPU)**:
  The CPU used in our experiments is an IGEPv2. It is programmed in the C++ language compiler. The sensors measure the state of the vehicle and they transmit the data to the CPU through different communication interfaces. The information is processed in order to compute the control law, which is sent to the actuators as control signals. In addition, the microprocessor is able to exchange information with a ground station in order to transmit the data relative to the octorotor state or to receive directives regarding to the ongoing mission.

- **Sensors**:
  The sensors onboard the octorotor are as follows:

  ◇ **Inertial Measurement Unit**: The IMU equipping the octorotor is a 3DM-GX3-25 Microstain. It is composed of accelerometers, gyroscopes, magnetometers, a temperature sensor, and an on-board processor running a sensor

fusion algorithm to provide static and dynamic orientation, and inertial measurements.

- ◇ **Telemeter**: An ultrasonic telemeter SRF08 is used for altitude measurement. It consists of a transmitter and a receiver. It works on a principle similar to that of transducers used in radar and sonar systems, which evaluate attributes of a target by interpreting the echoes from radio or sound waves, respectively. It generates high-frequency sound waves and evaluates the echo which is received back by the sensor, measuring the time interval between sending the signal and receiving the echo to determine the distance to the ground.
- ◇ **Camera**: The octorotor is equipped with a downward pointing optical flow camera module that uses the ground texture and visible features to determine the vehicle ground velocity. It provides the image for setup purposes, but it is not designed to capture images like a traditional camera. Note that we don't use the optical flow camera in our experiments.

- **Motion Capture System**: The indoor aviary where we experiment with the $Modulo-X8$ is equipped with a Motion Capture System from Optitrack. Twenty-four infrared cameras are fixed on the walls and are connected by Ethernet cables to a computer on which is installed the *Motive* tracking software. By tracking reflective markers placed on the octorotor's body, it is possible to reconstruct its position and orientation in space through triangulation with an accuracy of about 1 mm.

- **Ground Station**:
  The octorotor is connected to a ground station by a wireless link. The ground station is written in C++ using the QT libraries. It can run on a laptop in a real time environment. It consists of the following modules:

  - ◇ a communication link with the UAV (wireless) allowing the modification of the control laws, sensors calibration and filters parameters anytime during the flight ;
  - ◇ a communication link with the joystick (Bluetooth) during manual flights;
  - ◇ a data logging module allowing real-time plotting of the sensors measurements as well as the outputs of the control laws.

  The ground station receives the following data in real time for logging purposes:

  - ◇ the required and measured motors speeds from the ESC;
  - ◇ the estimated roll, pitch and yaw angles and velocities from the inertial measurement unit;
  - ◇ the estimated altitude from the ultrasonic sensor;
  - ◇ the UAV's position from the optitrack system.

The figure 2.7 shows the different connections between all the hardware used for the octorotor.

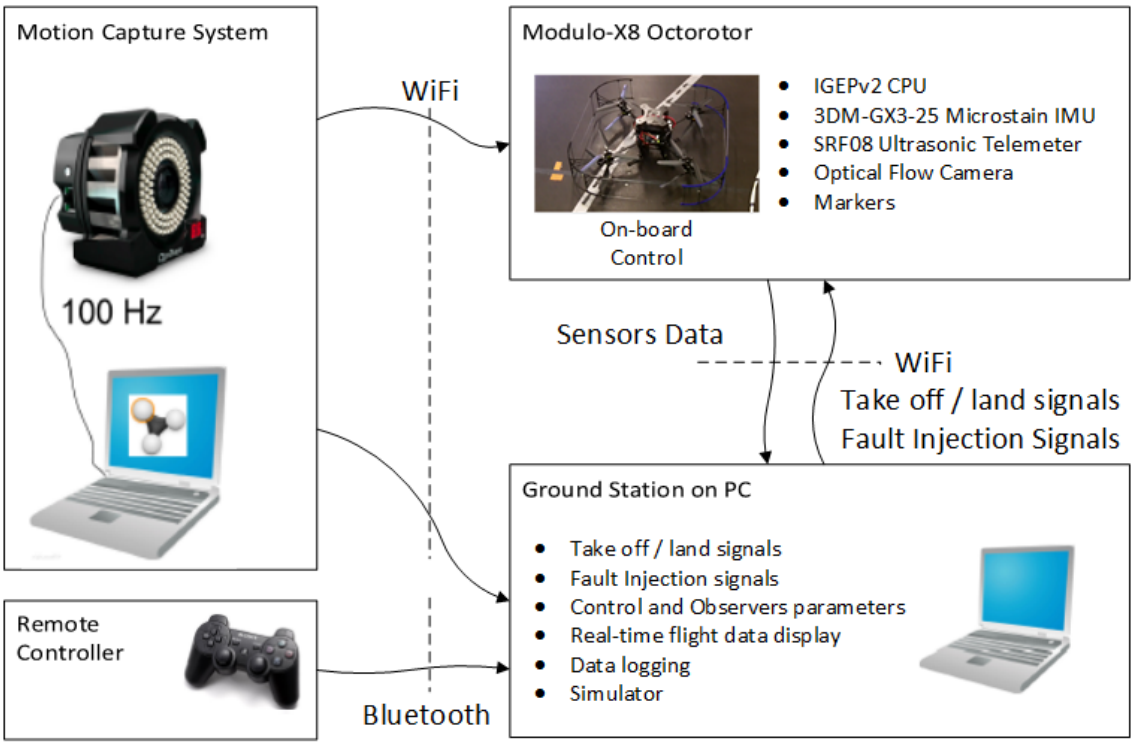


Figure 2.7 – The experimental setup of the real-time experiments for the coaxial octorotor

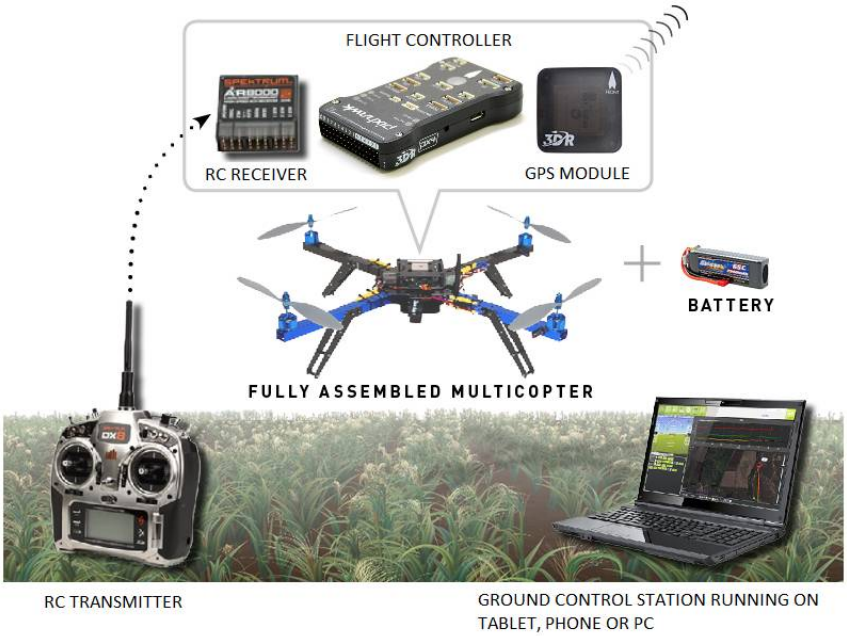


Figure 2.8 – The experimental setup of the real-time experiments for the DJI S500 and the tarot 650 quadrotors. Adapted from arducopter site

Figure 2.9 – Experimental DJI S500 quadrotor

Model parameters of the $Modulo - X_8$

All the model parameters were identified in the work done in [128]. They are given in Table 2.1.

K_f	Thrust factor	$2.25 * 10^{-5} Ns^2/rad^2$
K_t	Drag factor	$3.5 * 10^{-7} Nm/rad^2$
m	Mass of the vehicle	$1.6\,kg$
l	Length of the arm	$0.25\,m$
I_{xx}, I_{yy}	Inertia	$2.55 * 10^{-2} Kg.m^2$
I_{zz}	Inertia	$3.88 * 10^{-2} Kg.m^2$

Table 2.1 – The $Modulo - X_8$ model's parameters

2.4.2 *DJI S*500

The *DJI S*500 is a UAV from DJI Innovations. It can use a DJI WKM, NAZA or pixhawk autopilot system, and is aimed for research and leisure activities. The implemented UAV using this frame is shown in Figure 2.9 and is intended for outdoor experimentations.

The *S*500 is equipped with a Raspberry pi 3 + Navio2 open-source autopilot wich runs the Arducopter flight stack. This autopilot has the following specifications:

- Processor: the processor of the Navio2 is a 1.2GHz 64-bit quad-core ARMv8 CPU with 1GB RAM.
- Sensors: the Navio2 is equipped with 2 IMU, namely the MPU9250 and the LSM9DS1. It also include a barometer (MS5611) for measuring altitude and an RC I/O co-processor to handle the received signal of the transmitter. For localization, it supports the U-blox M8N Glonass/GPS/Beidou.

- Power: the Navio2 is equipped with a triple redundant power supply
- Interfaces: the Navio2 has several input and output interfaces to connect the sensors and the motors. It has the Universal Asynchronous Receiver-Transmitter (UART), Inter-integrated-circuit (I2C), Analog to Digital Convertor (ADC) interfaces for extensions, a PWM/S.Bus input to receive the transmitter signals and 14 PWM servo outputs to control the actuators.
- Dimensions: the weight of the Navio2 is 23g and the RPi3 is 54g. The size of the Navio is 55x65mm.
- Development Environment:
  - ◇ Arducopter flight stack: it is an open source autopilot system supporting multi-copters, traditional helicopters, fixed wing aircraft and rovers. The source code, written in C++, is developed by a large community but can be difficult to understand and modify. The flight stack can be found and downloaded from github [1].
  - ◇ Mission Planner: it is a ground control station for Copter and other vehicles that supports the ArduCopter flight stack. It is compatible with Windows only. It can be used as a configuration utility or as a dynamic control supplement for an autonomous vehicle.
- Propulsion System : The propulsion system of the S500 quadrotor consists of four BLDC motors (D2212-920KV) with 20A ESC and 1045 propellers.
- Additional sensors: the Navio2 can be equipped with a sensor to measure the altitude of the vehicle. This is useful because the barometer readings are not precise. For example, we use a Lidar Lite v3 which is a 2D downward pointing distance sensor .

The figure 2.8 shows the hardware used for a UAV running the Arducopter flight stack. In this architecture, we use for localization a GPS instead of the optitrack since we are outdoors. Also the manual control of the UAV is done using a direct RC transmitter and a radio connection instead of the bluetooth joystick.

2.4.2.1 Model identification of the *DJI S*500 quadrotor

All the model parameters are given by the manufacturers except the vehicle mass which was measured by a weight scale. They can be found in Table 2.2.

K_f	Thrust factor	$2.1 * 10^{-5} Ns^2/rad^2$
K_t	drag factor	$4 * 10^{-7} Nm/rad^2$
m	mass of the vehicle	$1.05\,kg$
l	length of the arm	$0.21\,m$
I_{xx}, I_{yy}	Inertia	$3.47 * 10^{-2} Kg.m^2$
I_{zz}	Inertia	$5.31 * 10^{-2} Kg.m^2$

[1] https://github.com/ArduPilot/ardupilot

Table 2.2 – The *DJI S*500 model's parameters

2.4.3 *Tarot* 650

The Tarot 650 is a commercial hobby type UAV commonly used for photography and recreational use. The implemented UAV using this frame is shown in Figure 2.10.

Figure 2.10 – Experimental Tarot 650 quadrotor

The *Tarot*650 is equipped with a Cube flight controller which is an updated version of the Pixhawk flight controller. It is an open-source and open-hardware autopilot which runs the Arducopter flight stack. This autopilot has the following specifications:

- Processor: the processor of the Cube is a 32-bit ARM Cortex M4 core with FPU with 168 Mhz/256 KB RAM/2 MB Flash and a 32-bit failsafe co-processor
- Sensors: the Cube is equipped with three redundant IMUs (accels, gyros and compass), namely the MPU9250, the ICM20948 and the ICM20648. It is also equipped with two redundant MS5611 barometers
- Power: the Cube has a redundant power supply with automatic failover and a servo rail high-power (7 V) and high-current ready. All peripheral outputs are over-current protected, and all inputs are ESD protected
- Interfaces: a list of the Cube's interfaces is given below.
  - ◇ 14x Pulse Width Modulation (PWM) servo outputs (8 from IO, 6 from FMU)
  - ◇ S.Bus servo output
  - ◇ R/C inputs for CPPM, Spektrum / DSM and S.Bus

- ◇ Analogue / PWM Received Signal Strength Indicator (RSSI) input
- ◇ 5x general purpose serial ports, 2 with full flow control
- ◇ 2x I2C ports
- ◇ Serial Peripheral Interface (SPI) port (un-buffered, for short cables only not recommended for use)
- ◇ 2x Controller Area Network (CAN) Bus interface
- ◇ 3x Analogue inputs (3.3V and 6.6V)
- ◇ High-powered piezo buzzer driver (on expansion board)
- ◇ High-power RGB LED (I2C driver compatible connected externally only)
- ◇ Safety switch / LED
- ◇ Optional carrier board for Intel Edison

- Propulsion System: The propulsion system of the Tarot 650 quadrotor consists of four BLDC motors (T-motor Airmax 920kc) with 20A ESC and 1045 propellers.

- Additional sensors: The Cube also supports the Lidar Lite v3 and the optical flow camera PX4Flow.

2.4.3.1 Model parameters of the *TAROT* 650 quadrotor

To our best knowledge, the model parameters of the *TAROT*650 were not identified elsewhere in the literature, and we couldn't obtain these information from the manufacturers. Therefore, we had to estimate these constants using different techniques. The *Tarot*650 parameters are given in Table 6.1.

m	Mass of the vehicle	$1.7\,kg$
l	Length of the arm	$0.23\,m$
I_{xx}, I_{yy}	Inertia	$3.38 * 10^{-2}\,Kg.m^2$
I_{zz}	Inertia	$2.25 * 10^{-2}\,Kg.m^2$

Table 2.3 – The *TAROT* 650 model's parameters

For the actuators, we identified the relations between the pulse width modulation PWM input of the motor and the generated force and torque of the motor. They are given by the following:

$$\begin{aligned} f_i &= (-1.4736u_{pwm}^3 + 11.0691u_{pwm}^2 - 16.7074u_{pwm} + 7.3007)/100 \\ \tau_i &= (-0.0905u_{pwm}^3 + 0.4771u_{pwm}^2 - 0.679u_{pwm} + 0.3045)/100 \end{aligned} \tag{2.27}$$

The estimation process of these model parameters, along with the actuators constants, is described in the section 2.5.

2.5 Model identification for the *TAROT* 650 quadrotor

In this section, the parameters of the *Tarot*650 quadrotor concerning the thrust forces and drag moments are identified. A thruster identification procedure is presented to determine the relation between the PWM input of the motors and the generated thrust and moment of the propeller system using an adapted setup. Also the identification of the parameters of the inertia matrix will be given.

2.5.1 Motor model Identification Procedure

As seen in sections 2.3.1 and 2.3.2, the relation between the generated aerodynamic forces and moments by the propellers are the following: $f_i = K_f \omega_i^2$ and $\tau_i = K_t \omega_i^2$. Since Round Per Minute (RPM) sensors are not yet compatible with the Cube flight controller, it is difficult to measure the motor coefficients K_f and K_t directly. However, since the Pulse Width Modulation (PWM) signals communicated between the autopilot and the ESCs are measured and identified, it is possible to identify a polynomial relation between the PWM signals and the generated forces and moments. These relationships can be considered as an alternative solution for the identification of the motor coefficients and a replacement for the K_f and K_t gains. To do this, we have set up two experiments: the first is used to identify the relation between the input PWM signals and the force generated by the actuator, and the second is used to identify the relation between the PWM signals and the generated torque.

2.5.2 Relationship between PWM inputs and generated thrust force

The first experiment is shown in Figure 2.11. It aims to find the relation between the PWM control signal and the thrust generated. The actuator and its propeller are mounted on top of a single bar attached to a base support. When the actuator start running, the weight scale under the support measures the added thrust force generated by the actuator. To identify the relation between the PWM signal and the thrust generated, we implemented a Matlab application to send PWM signals to the ESC, starting from 1000 up to 2000 with a step of 50. A value of 1000 means that no signal is sent to the motor, thus the motor is turn off, and a value of 2000 means that we run the motor at full power.

After collecting data, we obtain the curve of the thrust forces generated in Newton (N) with respect to each PWM signal sent to the motor u_{pwm}, as shown in Figure 2.12. Five tests of the same experiment were conducted to minimize the error due to the imperfections of the setup and the measurement noises, and to verify the repeatability of the results.

Using the polyfit function in Matlab, we found that a polynomial of degree three presents a good approximation of the collected data from the tests, with an approximation error of order 10^{-3}. The obtained relation is as follows:

$$f_i = (-1.4736 u_{pwm}^3 + 11.0691 u_{pwm}^2 - 16.7074 u_{pwm} + 7.3007)/100 \tag{2.28}$$

In fact, in hovering mode, the PWM values of motors is around 1500 and 1600 which corresponds to 2 N and 3 N of generated forces by the actuators. However, an

Figure 2.11 – Experimental setup for thrust identification of the Tarot 650

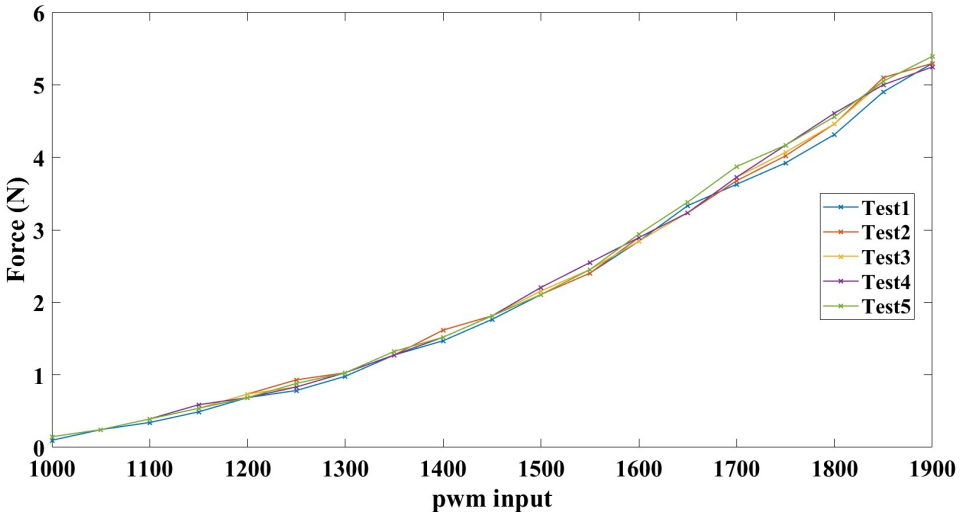


Figure 2.12 – Thrust force as function of the applied PWM input

approximation using a polynomial of second order gives an error of 2×10^{-2} which correspond to $1-2\%$ of percentage error in hovering mode. Even though this error is not huge, it adds up quickly with the passage of time, and we will get bad estimations after a short amount of time. Thus we consider this approximation unacceptable. Moreover, an approximation using a polynomial of fourth order gives an acceptable error of order 4×10^{-3}, while has a computation time significantly bigger than a polynomial of degree three which gives a similar error of 3×10^{-3}. Thus, we choose the approximation of the

polynomial of third order in our model.

2.5.3 Relationship between PWM inputs and generated torque

The second experiment is shown in Figure 2.13. It aims to find the relation between the PWM control signal sent to the motor and the torque generated. In this setup, a small bar, with a known length of 13 cm, is attached to the motor arm and placed on top of the weight scale. When the actuator starts running, the motor arm starts to turn slightly oppositely to the rotor's sense due to the generated torque. This will induce a force on the weight scale transmitted by the small bar. Thus by multiplying this force by the length of the small bar, we obtain the torque generated by the motor.

To identify the relation between the PWM signal and the generated torque, we used the same Matlab application than in Section 2.5.2 to send PWM signals to the ESC, starting from 1000 up to 2000 with a step of 50.

After collecting data, we obtain the curve of the torque generated in Newton.meter (N.m) with respect to each PWM signal sent to the motor u_{pwm}, as shown in Figure 2.14. Again, five tests of the same experiment are realized to minimize errors due to measurements noises and the imperfections of the setup.

Figure 2.13 – Experimental setup for torque identification of the Tarot 650

Using the polyfit function in Matlab, we found that a polynomial of degree three presents a good approximation of the collected data from the tests, with an approximation

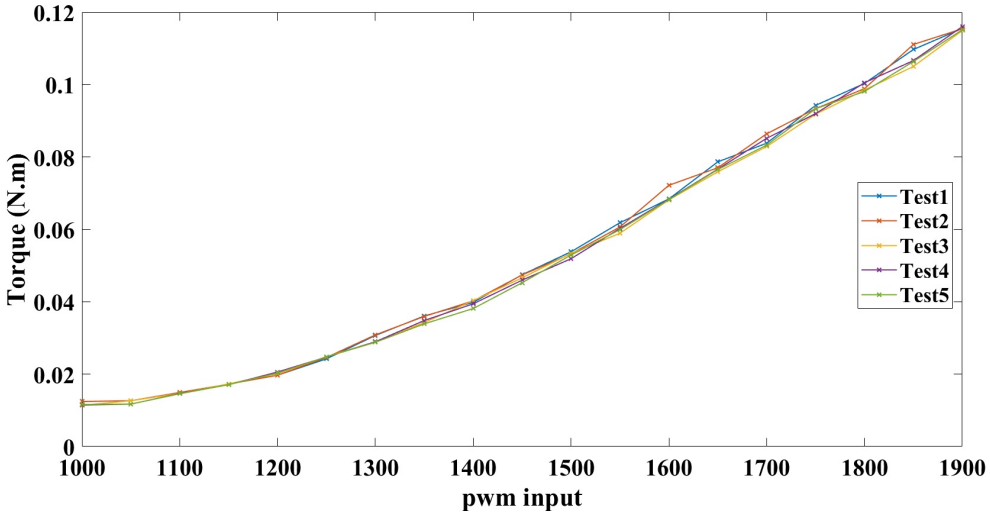


Figure 2.14 – Motor torque as function of the applied PWM input

error of order 10^{-3}. The obtain relation is as follows:

$$\tau_i = (-0.0905u_{pwm}^3 + 0.4771u_{pwm}^2 - 0.679u_{pwm} + 0.3045)/100 \tag{2.29}$$

Similarly as in Section 2.5.2, an approximation using a polynomial of second order gives an unacceptable error of 3×10^{-2}. An approximation using a polynomial of fourth order gives an acceptable error of 5×10^{-4}, but has a computation time bigger than a polynomial of degree three which gives a similar error of 2×10^{-4}. Thus, we choose the approximation of a polynomial of third order for our model.

2.5.4 Inertia Matrix

The moment of inertia (inertia matrix) I (see equation 2.30) of a rigid body is required to calculate the torque needed to achieve a desired angular acceleration about a rotational axis. The calculation of this moment of inertia depends on the body mass distribution and the chosen axis. Assuming a perfect symmetry about the three rotational axis, the off-diagonal terms of the inertia matrix (I_{xy}, I_{xz}, I_{yz}) become zero, and the diagonal terms can be calculated using the following equations:

$$\begin{aligned} I_{xx} &= \Sigma_i m_i (d_{y_i}^2 + d_{z_i}^2) \\ I_{yy} &= \Sigma_i m_i (d_{x_i}^2 + d_{z_i}^2) \\ I_{zz} &= \Sigma_i m_i (d_{x_i}^2 + d_{y_i}^2) \end{aligned} \tag{2.30}$$

with $\Sigma_i m_i$ the mass of the UAV's arm (including the actuator, ESC and the arm itself) and d_{x_i}, d_{y_i} and d_{z_i} the perpendicular distances from the end of the arm to the specific axis, which are the same in each case since the structure is symmetrical. In the case of the Tarot 650, by applying the equation 2.30, we get the following values:

$$\begin{aligned} I_{xx} = I_{yy} &= 3.38 * 10^{-2}\, Kg.m^2 \\ I_{zz} &= 2.25 * 10^{-2}\, Kg.m^2 \end{aligned} \tag{2.31}$$

2.6 Summary

This chapter presented the dynamic modeling of a quadrotor UAV and a coaxial counter-rotating octorotor UAV. After identifying the reference frames and coordinate systems, the forces and moments acting on the multirotor are detailed and the dynamics expressions of the model are derived using the Newton-Euler formalism. Then we presented the $Modulo - X_8$, $DJI\ S500$ and $Tarot\ 650$, three experimental UAV that will be used to validate our work. The model parameters of the octorotor were already identified in the literature, the model parameters of the S500 were provided by the manufacturers and the parameters of the Tarot platform parameters were identified using estimation techniques through experimental tests. Some of its parameters, the moments of inertia, were approximated using analytical expressions, while the propulsion system parameters, were identified using experimental data. In order to facilitate the modeling process of the UAV dynamics, some assumptions are made, such as supposing that the structure of the UAv is rigid and ignoring the motor dynamics. The obtained equation of motion in equation (2.24), and the expression of the virtual input vector in function of the real inputs in equations (2.25) and (2.26) will be used later in chapter 3, 4, 5 and 6 in order to derive the control laws and the FTC strategies.

Baseline Control Laws Experiments

Contents

Flight control systems are required to control the motion of the UAVs in order to fly and successfully complete their missions. This is done in terms of delivering the desired performance and following the desired path with sufficient accuracy. The control system's overall objective is to stabilize the UAV and minimize the tracking error between the desired reference command and the UAV's measured information provided by the onboard sensors. Flight control and stability are therefore a very critical part of the UAV because system failure due to malfunctioning components, lack of robustness or even unsuitable design will result in unacceptable mission performance or high risk of damage. In this chapter, we will present two strategies used to build the baseline control law for the different experimental UAVs used in our work: a linear Proportional Integral Derivative controller (PID) used in open source autopilots (Navio2 and Pixhawk Cube Flight Controller) and a nonlinear Sliding Mode Controller based on the Super-Twisting algorithm for attitude and altitude control. These two controllers are used in Chapter 4 to validate our wind-tolerant strategy. We will also present in this chapter the shared control law used in the fault tolerant control strategies discussed in Chapter 5.

In the following, we will first present the software architecture of the flight controller, before describing some basic concepts in control and the types of position control. Then, we will detail two control laws to control a UAV, namely a linear PID controller and a nonlinear robust controller based on the sliding mode theory. Finally, we will decompose the sliding mode robust controller into four subsystems shared by the three FTC schemes presented in section 5.1.

3.1 Software Architecture of Low-Level Flight Control of Multirotors

A classical multirotor UAV is an under-actuated system since it has six states (the position $\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ and the attitude angles $\Theta = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$), but only four independent inputs (the total thrust u_f and the three angular moments τ_ϕ, τ_θ and τ_ψ). Thus, a multirotor can only track four desired references ($\mathbf{p}_d$ and ψ_d), and the remaining references (ϕ_d and θ_d) are calculated internally by the system to obtain these desired references.

The low-level flight control of all the configurations of multicopters can be divided into four parts, namely the position control, attitude control, control allocation and motor control. Figure 3.1 illustrates this framework.

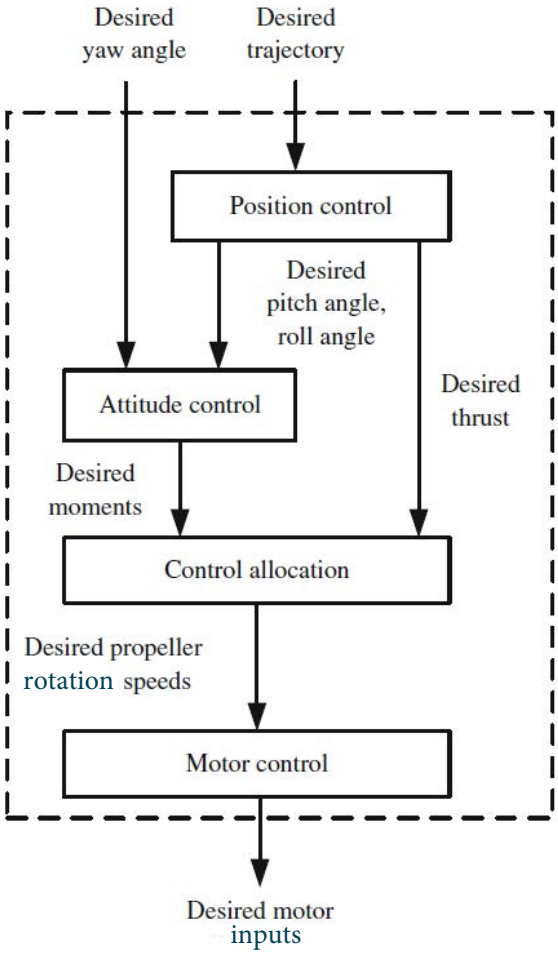


Figure 3.1 – Software Architecture of low-level flight control of multicopters

Since the UAV is generally controlled in position, the position control loop is generated first. The x and y positions are not controlled directly, but through the modification of the roll and pitch angles of the attitude control loop. Also, it is worth noting that the position control is executed at a low frequency (10 hz in Ardupilot), and

the attitude control is executed at a high frequency (400 hz in Ardupilot). Thus the desired roll and pitch angles are considered as constant inputs in the attitude control loop.

The objective of each part is as follows:

1. Position control aims to determine the desired roll angle ϕ_d, the desired pitch angles θ_d and the total thrust u_f according to the desired position $\mathbf{p}_d = \begin{bmatrix} x_d & y_d & z_d \end{bmatrix}^T$.
2. Attitude control aims to determine the desired angular moments τ_ϕ, τ_θ and τ_ψ based on the desired attitude angles $\Theta_d = \begin{bmatrix} \phi_d & \theta_d & \psi_d \end{bmatrix}^T$.
3. Control allocation aims to calculate the desired angular speeds of the propellers $\omega_{d,i}$, $i = 1,2,...,n_r$ where n_r represents the number of motors, with the intention of obtaining the desired control inputs u_f, τ_ϕ, τ_θ and τ_ψ.
4. Motor control aims to calculate the desired PWM signals for each motor according to $\omega_{d,i}$, $i = 1,2,...,n_r$.

The control architecture of the multirotor can be decomposed into different blocks, namely: the position controller, the attitude controller, the control allocator and the Multicopter (as shown in Figure 3.3). The position control block generates the thrust needed to control the altitude based on the desired altitude z_d and the measured one z. Also, it generates the desired roll and pitch angles ϕ_d and θ_d based on the differences between x_d, y_d and x, y. The attitude controller block generates the desired torques to the system based on the errors between the desired angles and the measured angles. The control allocator block manages the so-called multiplexing of the control input, which determines the individual control signal for each motor in PWM based on the desired thrust and toques. Finaly, the Multicopter block represents the plant or the equations of motion of the UAV.

3.2 Basic Control Concept

To better understand the control problem of tracking the desired references, we introduce some basic concepts in this section. In this thesis, we will consider that the position control of a vehicle can be divided into three types based on the given position $\mathbf{p}_d$: namely set-point, trajectory tracking and path following (see Figure 3.2)

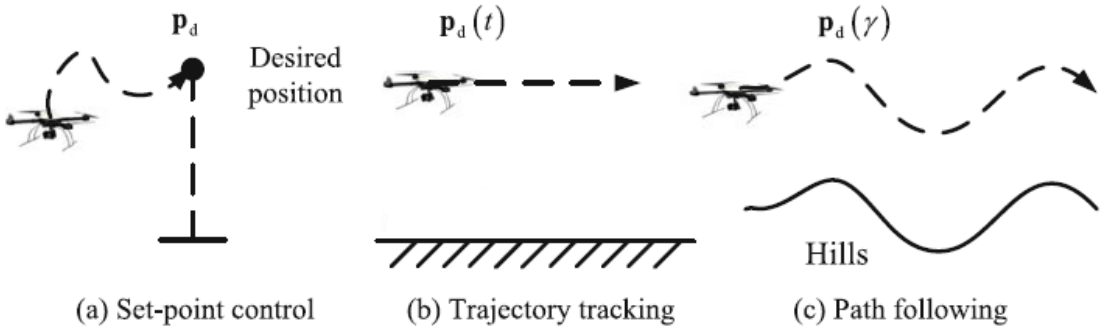


Figure 3.2 – Types of position control

First, let us define the three types of position control:

1. Set-point control: in this scenario, the desired position $\mathbf{p}_d = \begin{bmatrix} x_d & y_d & z_d \end{bmatrix}^T$ is a constant point in the earth frame. The goal is to design a controller which guarantees the position converges towards the desired one $\|\mathbf{p}(t) - \mathbf{p}_d\| \to 0$ as $t \to \infty$. In this case, the multirotor can fly to the desired position, regardless of the trajectory. This type of control is illustrated in Figure 3.2(a).

2. Trajectory tracking: in this case, the desired position $\mathbf{p}_d(t)$ is a time-dependent trajectory. The goal is to design a controller which guarantees that $\|\mathbf{p}(t) - \mathbf{p}_d(t)\| \to 0$ as $t \to \infty$. This type of application includes the case where a multirotor is required to follow a path at a given speed, as shown in Figure 3.2(b), or when it is asked to be at one position at a given time.

3. Path following: here the desired position $\mathbf{p}_d(\gamma)$ is a path which is determined by the parameter γ rather than the time. This parameter represent the set of point which describe the desired geometric path to be followed by the drone. The goal is to design a controller which guarantees that $\|\mathbf{p}(t) - \mathbf{p}_d(\gamma)\| \to 0$ as $t \to \infty$. This type of application includes. that a multirotor is required to fly along a profile over a hill or any geometric path, as shown in Figure 3.2(c).

The main difference between a trajectory tracking problem and a path following problem is whether the curves which describe the desired reference depends upon time or is independent of time. The path following is also referred to as three-dimensional 3D tracking, while trajectory tracking is referred to as four-dimensional 4D tracking as time is considered as an added dimension.

In this thesis, we mainly focuses on the set-point control problem and trajectory tracking problem. Readers who are interested in path following can refer to [32, 2].

3.3 Linear PID controller

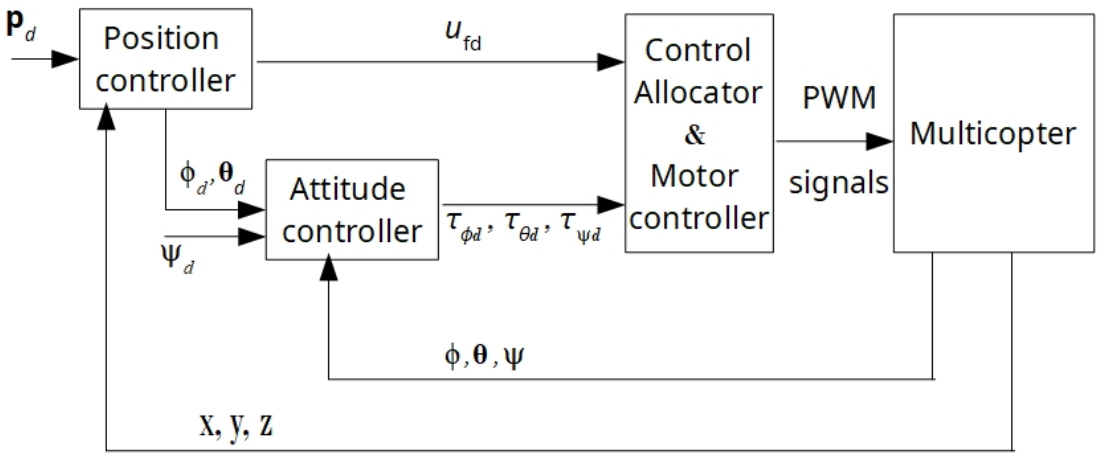


Figure 3.3 – Closed-loop structure of a low-level fight control system for multicopters

In the case of a PID controller, the generalized equation of the control law is given by the following expression:

$$u = K_p e + K_d \dot{e} + K_i \int_0^t e(\tau) d\tau \tag{3.1}$$

where u is the control input, K_p, K_d and K_i are the controller's gains, and e is the state error compared to the desired position.

It can be noticed that the PID controller can be decomposed into three parts, namely: the proportional term $K_p e$, the integral term $K_i \int_0^t e(\tau) d\tau$ and the derivative term $K_d \dot{e}$.

The proportional term produces an output value that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by a constant K_p, called the proportional gain constant. A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable.f

The integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error is then multiplied by the integral gain K_i and added to the controller output. The integral term accelerates the movement of the process towards setpoint and eliminates the residual error that occurs with a pure proportional controller.

Finally, the derivative term is calculated by determining the slope of the error over time and multiplying this rate of change by the derivative gain K_d. The magnitude of the contribution of the derivative term to the overall control action is termed the derivative gain, K_d. Derivative action predicts system behavior and thus improves settling time and stability of the system.

3.3.1 Linear Model Simplification

The full nonlinear model has been established in Chapter 2. However, in order to apply a linear controller, the nonlinear model in equation (2.24) needs to be linearized and simplified. Thus, the PID control law design is performed under several assumptions:

Assumption PID1 The system dynamics are limited to small angles and small variations of linear and angular velocities, and thus no acrobatic behavior of the UAV can occur.

Assumption PID2 the system is always considered near its hovering state (or equilibrium point).

To obtain the simplified model, the first simplification (based on the **Assumption PID1**) is to ignore the effect of the propeller gyroscopic effect τ_g, drag moments τ_i and the corrective term $\omega \times I\omega$ which are described in section 2.3.2. Thus, the simplified model

becomes:

$$\begin{cases} m\ddot{x} &= (\cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi)u_f \\ m\ddot{y} &= (\cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi)u_f \\ m\ddot{z} &= (\cos\phi\cos\theta)u_f - mG \\ I_{xx}\ddot{\phi} &= \tau_\phi \\ I_{yy}\ddot{\theta} &= \tau_\theta \\ I_{zz}\ddot{\psi} &= \tau_\psi \end{cases} \tag{3.2}$$

However, the system in equation (3.2) is still non-linear, so more simplifications are needed. If we consider that the system will only turn smoothly with no acrobatic behavior, we can acknowledge that the roll and pitch angles will always remain small (**Assumption PID1**). Thus, the following simplifications are applicable:

$$\begin{aligned} \sin\phi &\approx \phi \\ \sin\theta &\approx \theta \\ \cos\phi &\approx 1 \\ \cos\theta &\approx 1 \end{aligned} \tag{3.3}$$

Still considering that we have only small roll and pitch angles, we can also assume that the system is always considered near its hovering state (or equilibrium point), that is where the roll and pitch angles of the drone are zero (**Assumption PID2**). In this state, the total thrust of the drone approximates to the weight of the multirotor and we can then simplify the UAV's thrust as:

$$u_f \approx mG \tag{3.4}$$

Finally, the original system given in equation (3.2) is divided into three linear models, namely the horizontal position model, altitude model and attitude model. These three models are detailed in the following sections.

3.3.2 Horizontal Position Model

According to the small-angle assumptions in equations (3.3) and (3.4), the first two equations of the system (3.2) can be simplified as:

$$\begin{aligned} \ddot{x} &= (\theta\cos\psi+\phi\sin\psi)G \\ \ddot{y} &= (\theta\sin\psi-\phi\cos\psi)G \end{aligned} \tag{3.5}$$

In the horizontal position model (3.5), the yaw angle ψ is known, thus the terms $\cos\psi$ and $\sin\psi$ are constant. The roll and pitch angles ϕ and θ represents the inputs and the variables x and y represents the outputs of the system (3.5). Therefore, the horizontal position model (3.5) is, in fact, a linear model.

3.3.3 Altitude Model

According to the small-angle assumptions in equations (3.3) and (3.4), the third equation of the system (3.2) is simplified as:

$$\ddot{z} = \frac{u_f}{m} - G \tag{3.6}$$

The control input is u_f and the output z. Here the simplification of the term u_f is not applied since this simplification would make the control input u_f disappears from the equation. Obviously, the altitude model (3.6) is also a linear model.

3.3.4 Attitude Model

The last three equations of the system (3.2) represent the attitude model:

$$\begin{array}{rl} \ddot{\phi} & = \frac{\tau_\phi}{I_{xx}} \\ \ddot{\theta} & = \frac{\tau_\theta}{I_{yy}} \\ \ddot{\psi} & = \frac{\tau_\psi}{I_{zz}} \end{array} \tag{3.7}$$

which is obviously a linear model. The control input of the roll, pitch and yaw dynamics are respectively τ_ϕ, τ_θ and τ_ψ, whereas the output are the roll, pitch and yaw angles (ϕ, θ and ψ).

3.3.5 Traditional PID Controller

We present in this section the traditional PID controllers often used in UAVs, and prove that they converge to the desired position and orientation in a finite time.

3.3.5.1 Horizontal position control and attitude control and attitude control

According to the linear position model in equation (3.2), the desired attitude angles ϕ_d and θ_d are expected to be designed such that:

$$\begin{array}{l} \lim_{t\to\infty} e_x(t) = 0 \\ \lim_{t\to\infty} e_y(t) = 0 \end{array} \tag{3.8}$$

where $e_x = x - x_d$ and $e_y = y - y_d$. First, in PID controllers, we need to put the errors under this form:

$$\begin{array}{l} \ddot{e}_x = -K_{xd}\dot{e}_x - K_{xp}e_x - K_{xi}\int e_x \\ \ddot{e}_y = -K_{yd}\dot{e}_y - K_{yp}e_y - K_{yi}\int e_y \end{array} \tag{3.9}$$

where K_{xp}, K_{xd}, K_{yp} and K_{yd} are the control gains. If these gains verify the conditions $K_{xp} > 0$, $K_{xd} > 0$, $K_{yp} > 0$ and $K_{yd} > 0$, we get $\lim_{t\to\infty} e_x(t) = 0$ and $\lim_{t\to\infty} e_y(t) = 0$ in the case were the roll and pitch errors converge to zero. In order to comply with equation (3.9), the accelerations $\ddot{x}$ and $\ddot{y}$ must satisfy

$$\begin{array}{l} \ddot{x} = \ddot{x}_d - K_{xd}\dot{e}_x - K_{xp}e_x - K_{xi}\int e_x \\ \ddot{y} = \ddot{y}_d - K_{yd}\dot{e}_y - K_{yp}e_y - K_{yi}\int e_y \end{array} \tag{3.10}$$

By combining equations (3.5) with equations (3.10), we get:

$$\begin{array}{l} (\theta_d \cos\psi + \phi_d \sin\psi)G = \ddot{x}_d - K_{xd}\dot{e}_x - K_{xp}e_x - K_{xi}\int e_x \\ (\theta_d \sin\psi - \phi_d \cos\psi)G = \ddot{y}_d - K_{yd}\dot{e}_y - K_{yp}e_y - K_{yi}\int e_y \end{array} \tag{3.11}$$

It these equations, we have replaced ϕ and θ by ϕ_d and θ_d, because by defining $e_\phi = \phi_d - \phi$ and $e_\theta = \theta_d - \theta$ as the roll and pitch errors respectively, the conditions

$\lim_{t\to\infty} e_x(t) = 0$ and $\lim_{t\to\infty} e_y(t) = 0$ are satisfied only if $\theta = \theta_d$ and $\phi = \phi_d$ in (3.5), equivalently $e_\phi = 0$ and $e_\theta = 0$. In practice when a set-point control problem is considered, namely $\ddot{x}_d = \dot{x}_d = \ddot{y}_d = \dot{y}_d = 0$, the equation (3.11) becomes:

$$\begin{aligned}\phi_d &= (\cos\psi(K_{yd}\dot{y} + K_{yp}(y - y_d)) - \sin\psi(K_{xd}\dot{x} + K_{xp}(x - x_d) + K_{xi}\int x - x_d)/G \\ \theta_d &= (-\cos\psi(K_{xd}\dot{x} + K_{xp}(x - x_d)) - \sin\psi(K_{yd}\dot{y} + K_{yp}(y - y_d) + K_{yi}\int y - y_d)/G\end{aligned} \tag{3.12}$$

Following the same manner, the attitude control input τ_ϕ, τ_θ and τ_ψ can be expressed as:

$$\begin{aligned}\tau_{\phi_d} &= (-K_{\phi d}\dot{\phi} - K_{\phi p}(\phi - \phi_d))/I_{xx} \\ \tau_{\theta_d} &= (-K_{\theta d}\dot{\theta} - K_{\theta p}(\theta - \theta_d))/I_{yy} \\ \tau_{\psi_d} &= (-K_{\psi d}\dot{\psi} - K_{\psi p}(\psi - \psi_d))/I_{zz}\end{aligned} \tag{3.13}$$

where $K_{\phi d}$, $K_{\phi p}$, $K_{\theta d}$, $K_{\theta p}$, $K_{\psi d}$, and $K_{\psi p}$ are the control gains for the attitude control.

3.3.5.2 Altitude control

For to the altitude model in equation (3.6), the control objective is :

$$\lim_{t\to\infty} e_z(t) = 0 \tag{3.14}$$

where $e_z = z - z_d$. As seen in the previous section the transient process for the PID controller is of the form:

$$\ddot{e}_z = -K_{zd}\dot{e}_x - K_{zp}e_x \tag{3.15}$$

where K_{zp} and K_{zd} are the control gains for the altitude control. If the gains verify the Hurwitz condition ([42]), then according to the stability theory we get $\lim_{t\to\infty} e_z(t) = 0$. In order to comply with equation (3.15), the acceleration $\ddot{z}$ must satisfy

$$\ddot{z} = \ddot{z}_d - K_{zd}\dot{e}_z - K_{zp}e_z - K_{zi}\int e_z \tag{3.16}$$

By combining equations (3.6) with equations (3.16), we get:

$$\frac{u_{f_d}}{m} - G = \ddot{z}_d - K_{zd}\dot{e}_z - K_{xp}e_z - K_{zi}\int e_z \tag{3.17}$$

where u_{f_d} is the desired thrust. From the above equation, u_{f_d} has the following expression:

$$u_{f_d} = m(\ddot{z}_d - K_{zd}\dot{e}_z - K_{zp}e_z - K_{zi}\int e_z) + mG \tag{3.18}$$

The desired value of the thrust is calculated from the above equation in order to satisfy the condition that $\lim_{t\to\infty} e_z(t) = 0$. Thus, if the real generated thrust u_f by the UAV is equal to the desired thrust, in other work if $u_f = u_{f_d}$, then the altitude of the UAV z will follow the desired altitude z_d.

In practice, when a set-point control problem is considered, namely $\dot{z}_d = \ddot{z}_d = 0$, the desired thrust u_{f_d} is written explicitly as follows:

$$u_{f_d} = m(-K_{zd}\dot{z} - K_{zp}e_z - K_{zi}\int e_z) + mG \tag{3.19}$$

3.3.6 PID Controllers in the open source autopilot Ardupilot

In Ardupilot, the PID controller is decomposed into two steps. In the first step, the desired velocities are calculated based on the position errors, and in the second step, a PID controller is applied on the system velocities to drive the system errors in position and velocity to zero. The main advantage of this decomposition is that it can be useful to simplify the tuning of the control gains, where the gains in the first step are intended to manage the responsiveness of the system (relationship between position and velocity), and the gains in the second step of the PID controller of the velocities are intended to manage the other characteristics of the controller (overshoot, settling time...). The work presented in this section was done using retro-engineering from the flight stack of the Ardupilot (version 3.6 of Arducopter December 10, 2019).

The design of the position controller based on the ArduPilot flight stack supported by the Navio2 and the pixhawk Cube is detailed in the following sections.

3.3.6.1 Horizontal position control

Let be:

$$\begin{aligned} \dot{x} &= v_x \\ \dot{y} &= v_y \end{aligned} \tag{3.20}$$

In this PID controller, the desired values of v_x and v_y, namely v_{xd} and v_{yd}, are designed as follows

$$\begin{aligned} v_{xd} &= K_{xp}(x_d - x) \\ v_{yd} &= K_{yp}(y_d - y) \end{aligned} \tag{3.21}$$

where K_{xp}, $K_{yp} \in R_+$ are the positive control gains. By defining the velocity errors as follows: $e_{vx} = v_{xd} - v_x$ and $e_{vy} = v_{yd} - v_y$, if the following conditions are true $\lim_{t\to\infty} v_{xd}(t) = 0$ and $\lim_{t\to\infty} v_{yd}(t) = 0$, $\lim_{t\to\infty} e_{vx}(t) = 0$ and $\lim_{t\to\infty} e_{vy}(t) = 0$, then the conditions $\lim_{t\to\infty} e_x(t) = 0$ and $\lim_{t\to\infty} e_y(t) = 0$ are satisfied. The horizontal control loop is built via Equations (3.20) and (3.5), according to the following relations:

$$\begin{aligned} \dot{v}_x &= (\theta \cos\psi + \phi \sin\psi)G \\ \dot{v}_y &= (\theta \sin\psi - \phi \cos\psi)G \end{aligned} \tag{3.22}$$

The next step is to design the desired roll and pitch angles, namely ϕ_d and θ_d. Following the same procedure than in Equation (3.5), the PID controller is designed as follows:

$$\begin{aligned} (\theta_d \cos\psi + \phi_d \sin\psi)G &= -K_{vxp}e_{vx} - K_{vxi}\int e_{vx} - K_{vxd}\dot{e}_{vx} \\ (\theta_d \sin\psi - \phi_d \cos\psi)G &= -K_{vyp}e_{vy} - K_{vyi}\int e_{vy} - K_{vyd}\dot{e}_{vy} \end{aligned} \tag{3.23}$$

where K_{vxp}, K_{vxd}, K_{vxi}, K_{vyp}, K_{vyd} and K_{vyi} are positive control gains. Under the assumption that $\dot{v}_x = 0$ and $\dot{v}_y = 0$, the conditions $\lim_{t\to\infty} e_x(t) = 0$ and $\lim_{t\to\infty} e_y(t) = 0$

are satisfied if $\lim_{t\to\infty}(\phi(t)-\phi_d(t))=0$ and $\lim_{t\to\infty}(\theta(t)-\theta_d(t))=0$. The desired Euler angles are derived from (3.23) as

$$\begin{aligned}\phi_d &= (\cos\psi(K_{vyd}\dot{v}_y + K_{vyp}(v_y - v_{yd})) - \sin\psi(K_{vxd}\dot{v}_x + K_{vxp}(v_x - v_{xd}) + K_{vxp}\int v_x - v_{xd})/G\\ \theta_d &= (-\cos\psi(K_{vxd}\dot{v}_x + K_{vxp}(v_x - v_{xd})) - \sin\psi(K_{vyd}\dot{v}_y + K_{vyp}(v_y - v_{yd}) + K_{vyp}\int v_y - v_{yd})/G\end{aligned} \tag{3.24}$$

In the case of a set-point problem we can choose $\dot{x}_d = 0$, $\dot{v}_{xd} = 0$, $\dot{y}_d = 0$ and $\dot{v}_{yd} = 0$. Also, to avoid the noises generated by the time derivative signals, the terms $K_{vxd}\dot{e}_{vx}$ and $K_{vyd}\dot{e}_{vy}$ can be omitted, but in this case the system will be more sensitive to the measurement noises and external perturbations.

Note that Eq. (3.21) is written in the form of a P controller, while Eq. (3.24) is designed as a PID controller. The reason for that change is that the model (3.20) is a kinematic model, thus uncertainty-free and a P controller is sufficient to satisfy its stability. However, the attitude model (3.22) is a dynamic model, thus subject to uncertainties and requiring a PID controller to compensate them.

3.3.6.2 Altitude control

Let be:

$$\dot{z} = v_z \tag{3.25}$$

the desired value of v_z, namely v_{zd}, is designed as follows

$$v_{zd} = K_{zp}(z_d - z) \tag{3.26}$$

where $K_{zp} \in R_+$ is the positive control gain. Under the assumption that $v_{zd} = 0$, if $\lim_{t\to\infty} e_{vz}(t) = 0$, then $\lim_{t\to\infty} e_z(t) = 0$, where $e_{vz} = v_{zd} - v_z$. Actually, the altitude control loop is built via Equations (3.25) and (3.26), according to the following relations:

$$\frac{u_f}{m} - G = \dot{v}_z \tag{3.27}$$

To obtain the desired thrust u_{fd}, we follow the same procedure in Equation (3.17), and the PID controller is designed as follows:

$$\frac{u_{f_d}}{m} - G = -K_{vzp}e_{vz} - K_{vzd}\dot{e}_{vz} - K_{vzi}\int e_{vz} \tag{3.28}$$

where K_{vzp}, K_{vzd} and K_{vzi} are the positive control gains. Under the assumption that $\dot{v}_z = 0$, the condition $\lim_{t\to\infty} e_z(t) = 0$ is satisfied if $\lim_{t\to\infty}(u_f(t) - u_{f_d}(t)) = 0$. The desired thrust is derived from (3.28) as

$$u_{f_d} = m(-K_{vzp}(v_z - v_{zd}) - K_{vzd}(\dot{v}_z - \dot{v}_{zd}) - K_{vzi}\int (v_z - v_{zd})) + mG \tag{3.29}$$

3.4 Sliding Mode Controller for attitude and altitude Control

The sliding mode controller is a well-known robust controller which is widely used in dynamical systems like UAVs. The main advantage of the sliding mode controller is its ability to passively compensate the uncertainties in the system parameters and some of the perturbations on the system. This section is devoted to the fundamental mathematical concepts of the sliding mode control theory and the application of the Super-Twisting SMC on the octorotor.

3.4.1 Sliding Mode Theory

Sliding Mode Control is a nonlinear control method that alters the dynamics of a nonlinear system by applying a discontinuous control signal that leads the system states onto a particular surface of the system in the state space, named the sliding surface. Once the sliding surface is reached, the sliding mode controller keeps the system in the states in the close neighborhood of the sliding surface (see Fig. 3.4) [146]. The state-feedback control law is not a continuous function of time. Instead, it can switch from one continuous structure to another based on the current position in the state space. Hence, Utkin called the sliding mode control a variable structure control method [147].

3.4.2 Matched and Unmatched perturbation

In the case of underactuated systems, we control some of the states of the system directly through the control inputs, while other state's variables are controlled indirectly through some of the other state's variables. In this case, we call the perturbations affecting the directly controlled state's variables matched perturbations, and the perturbations affecting the other state's variables are called unmatched perturbations.

It is worth to mention that sliding mode controllers can fully compensate the matched perturbations. However, the unmatched uncertainties are only partially compensated using the sliding mode controller. In order to extend the robustness of sliding mode controllers towards those unmatched perturbations (imperfect modeling values, external perturbations, etc.), other techniques should be used like adaptive controllers or observers used to estimate these perturbations.

3.4.2.1 First-Order Sliding Mode Control

The synthesis of a sliding mode control law consists of two steps: selecting a suitable sliding manifold in order to assign the desired dynamics, then designing a discontinuous control law that forces the system states to reach the sliding surface in finite time and to stay there despite the uncertainties and disturbances.

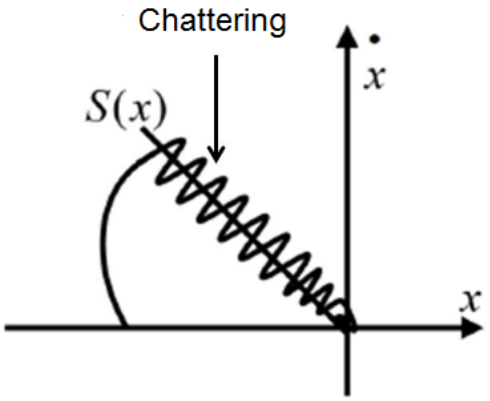


Figure 3.4 – Principle of Sliding Mode Control

Consider the nonlinear time-invariant system defined as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \vdots \\ \dot{x}_{n-1} \\ \dot{x}_n \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ \vdots \\ x_n \\ f_n(\mathbf{x},t) \end{bmatrix} + \begin{bmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ g_n(\mathbf{x},t) \end{bmatrix} u \qquad (3.30)$$
$$y = x_1$$

where $\mathbf{x} = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}^T \in X \subset \mathbb{R}^n$ is the system state with $u \in U \subset \mathbb{R}^n$ is a bounded discontinuous control input. y represents the output of the system. $f_n(\mathbf{x},t)$ and $g_n(\mathbf{x},t)$ are continuously differentiable functions. Unknown parameter uncertainties and external disturbances are included in these two functions. We can also model the external disturbances as additive terms outside these functions. We assume that the objective of the control is to follow a certain reference trajectory.

Selection of the Sliding Manifold

Let $s(\mathbf{x},t)$ be a continuous function such that the system trajectory exhibits desirable behavior when it is zero. This function is called the switching variable.
The set

$$S = \{\mathbf{x} \in X | s(\mathbf{x},t) = 0\} \qquad (3.31)$$

of dimension (n-1) is called the sliding manifold.
A sliding mode regime exists if there exists a finite time t_s such that the solution of (3.30) satisfies $s(\mathbf{x},t) = 0$ for each $t \geq t_s$.
In general, the sliding surface is selected as an hyperplane passing through the origin of the state space, and is in most cases a linear combination of the state variables. The sliding surface should have a relative degree equal to 1 with respect to the control input u. This is a necessary condition for establishing the sliding mode regime.

Control Law

Once the sliding surface is selected, the second step is to design a control law that stabilizes the switching variable around the origin in finite time. Since sliding mode trajectories belong to a manifold of a dimension lower than the one of the original system, the order of the system is reduced as well. This enables a designer to simplify the design procedure and decouple the overall system motion into independent partial components of lower dimension.
The objective of the control law is to force the system states to reach and stay on the sliding manifold. In other words, the control law should make the sliding surface locally attractive (trajectories outside the surface should converge to it). Thus it can be calculated by checking the convergence condition of $s(x,t)$ to zero.
The behavior of the closed-loop system can be decomposed into two modes:

- **Reaching mode**: This phase corresponds to the time interval $t \in [0, t_s]$ during which the system state trajectories are not on the sliding surface S, but try to reach it. In this phase, the system remains sensitive to parameter variations. However, to avoid a large influence of these variations, one solution is to shorten the duration of this phase.

- **Sliding mode**: This phase corresponds to the time interval $t \in [t_s, \infty[$ during which the state trajectories are confined throughout the set S. The magnitude of the control discontinuity depends on the system states and disturbances, and the control gains can be adapted according to the nature and level of these disturbances and thus tolerate them. The system is then insensitive to any perturbation involved in the same direction as the control input during the sliding mode.

The control law consists of an equivalent control u_{eq} and a switch control u_s, respectively corresponding to the sliding mode and the reaching mode. System states are kept on the sliding surface by the equivalent control and they attain the sliding surface by the switch control.
The equivalent control law u_{eq} can be found by solving

$$s = \dot{s} = 0 \tag{3.32}$$

To counteract disturbances and uncertainties in $f_n(\mathbf{x},t)$ and $g_n(\mathbf{x},t)$, a discontinuous loop is used in order to satisfy the control objective $s(\mathbf{x},t) = 0$. The control law is then written as:

$$u = u_{eq} - g(\mathbf{x},t)^{-1} K \text{sign}(s(\mathbf{x},t)) \tag{3.33}$$

where K is the control gain.
In order to govern the system states to reach the sliding surface $s = 0$ in a limited time and to remain there, the control law should be designed such that the following sliding condition is satisfied:

$$s\dot{s} \leq -\gamma|s|, \text{ with } \gamma > 0. \tag{3.34}$$

Chattering

The main disadvantage of the sliding mode controller is the chattering: finite-amplitude high-frequency oscillations of the controlled structure. Chattering can cause low control

accuracy, high wear of moving mechanical parts, or might even damage the system being controlled. Chattering is thus the main obstacle in implementing sliding mode controllers, and the problem is exarcerbated in practice for two main reasons. High bandwidth dynamics are often neglected in the open-loop plant model used to design controllers. In SMC implementations, these dynamics are exacerbated by the switch control input and often have significant effects on the system. Three main approaches were proposed to reduce the chattering:

- **Boundary Layer Control**: To remedy to the chattering, the strict requirement of movement on the sliding surface is relaxed and a quasi sliding mode is obtained by piecewise linear or smooth approximations of the switching element in a boundary layer of the sliding manifold. Possible linear approximations of the sign function are the saturation and the sigmoid functions (see [38]).
- **Observer-Based Sliding Mode Control**: The use of an asymptotically stable observer in the closed-loop system can eliminate the phenomenon of chattering in spite of the discontinuity of the control. The basic idea is to generate a sliding regime in the system observer instead of generating it in the system itself. Since the observer does not depend on the unmodeled dynamics, the ideal sliding mode takes place in the closed loop of the observer. Thus, the system output follows the output of the observer without any chattering. However, the synthesis of an asymptotically stable observer is not an easy task when the system is nonlinear and uncertain. Also, the applicability of this method is limited to systems for which the synthesis of an observer is possible (see [131]).
- **Higher-Order Sliding Modes (HOSM)**: The Higher-Order Sliding Modes technique, which is a generalization of the first order sliding mode control with higher order derivatives of the sliding variable, can reduce or eliminate this undesirable phenomenon (see [137]).

In the next section, we will present a second order sliding mode technique which counts amongst the HOSM.

3.4.2.2 Second-Order Sliding Mode: Super-Twisting Algorithm

The Super-Twisting Algorithm (STA) is a second order sliding mode control that handles a relative degree equal to one, which means that the control input appears in the expression of the first derivative of the sliding variable. It generates the continuous control function that drives the sliding variable and its derivative to zero in finite time in the presence of the smooth matched disturbances with bounded gradient, when this boundary is known. Since the STA algorithm contains a discontinuous function under the integral, chattering is not eliminated but only attenuated.

The control law design is performed under several assumptions:

Assumption SMC1 The desired trajectory controllers are continuous, differentiable and their derivatives are bounded.

Assumption SMC2 The external disturbances and their first derivatives are bounded. For example when considering wind perturbations, the wind forces variations are considered continuous.

Assumption SMC3 The roll, pitch and yaw angles are constrained to $(-\pi/2 < \phi, \theta < \pi/2)$ and $(-\pi < \psi < \pi)$, so acrobatic behaviors such as looping are not allowed.

Assumption SMC4 The system dynamics are limited to small angles and small variations of linear and angular velocities. This behavior is common in practice with UAV.

Assumption SMC5 All the control inputs are bounded. In practice, they are normalized as $u_f \in]0,1]$ and $\tau_\phi, \tau_\theta, \tau_\psi \in [-1;1]$.

The objective of the controller is to ensure the convergence of the actual state vector $[x, y, z, \psi]$ to the desired trajectories $[x_d, y_d, z_d, \psi_d]$. Since the system is underactuated, we design a controller for the altitude z, and for the states ϕ, θ and ψ since the states x and y can be controlled through these angles. Let us first consider the following vectors: $\mathbf{x}_1 = [z, \phi, \theta, \psi]^T$, and $\mathbf{x}_2 = [\dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}]^T$ so we end up with the following state-space form of the model:

$$\begin{cases} \dot{\mathbf{x}}_1 = \mathbf{x}_2 \\ \dot{\mathbf{x}}_2 = \mathbf{f}(\mathbf{x_1}, \mathbf{x_2}) + \mathbf{g}(\mathbf{x_1}, \mathbf{x_2})\mathbf{u} + \mathbf{w} \end{cases} \tag{3.35}$$

where $\mathbf{w} = \begin{bmatrix} w_z & w_\phi & w_\theta & w_\psi \end{bmatrix}^T$ represents the external disturbances vector, and we have $\mathbf{x_1}, \mathbf{x_2}, u$ and $\mathbf{w}$ are dependent of time. The vector $\mathbf{f}(\mathbf{x_1}, \mathbf{x_2})$, the matrix $\mathbf{g}(\mathbf{x_1}, \mathbf{x_2})$ and $\mathbf{u}$ are defined as (using equation 2.24):

$$\mathbf{f}(\mathbf{x_1}, \mathbf{x_2}) = \begin{pmatrix} -G \\ (I_{yy} - I_{zz})\dot{\theta}\dot{\psi} - J_r\dot{\theta}\Omega_r \\ (I_{zz} - I_{xx})\dot{\phi}\dot{\psi} + J_r\dot{\phi}\Omega_r \\ (I_{xx} - I_{yy})\dot{\phi}\dot{\theta} \end{pmatrix} \tag{3.36}$$

$$\mathbf{u} = \begin{pmatrix} u_f \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{pmatrix} \tag{3.37}$$

$$\mathbf{g}(\mathbf{x_1}, \mathbf{x_2}) = \begin{bmatrix} \frac{1}{m}(\cos\theta\cos\phi) & 0 & 0 & 0 \\ 0 & \frac{1}{I_{xx}} & 0 & 0 \\ 0 & 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & 0 & \frac{1}{I_{zz}} \end{bmatrix} \tag{3.38}$$

Note that in equation (3.36) the terms $J_r\dot{\phi}\Omega_r$, $-J_r\dot{\theta}\Omega_r$ will be neglected by means of the (**Assumption SMC4**) in the rest of this thesis. According to the assumption (**Assumption SMC3**), the control matrix $\mathbf{g} = (\mathbf{x_1}, \mathbf{x_2})$ is non-singular and thus invertible, and the following inequalities are established (using **Assumption SMC2**) and the equation of motions of the system):

$$\begin{array}{l} \mid f_i(\mathbf{x_1}, \mathbf{x_2}) \mid < \tilde{f}_i, i = 1, ..., 4 \\ \mid \dot{w}_i \mid < \tilde{w}_i, \ i = z, \phi, \theta, \psi \end{array} \tag{3.39}$$

where $\tilde{f}_i$ and $\tilde{w}_i$ are known positive values. The sliding manifolds vector $\mathbf{s} = \begin{bmatrix} s_z & s_\phi & s_\theta & s_\psi \end{bmatrix}^T$ is defined as:

$$\mathbf{s} = \dot{\mathbf{e}} + \lambda\mathbf{e} \tag{3.40}$$

where $\mathbf{e} = \begin{bmatrix} e_z & e_\phi & e_\theta & e_\psi \end{bmatrix}^T$ represent the errors between the actual states of the system and the desired ones and $\lambda = \text{diag}(\lambda_z, \lambda_\phi, \lambda_\theta, \lambda_\psi)$ is a positive definite gain matrix. According to [49], we choose the control input as the following expression:

$$\mathbf{u} = \mathbf{g}(\mathbf{x_1}, \mathbf{x_2})^{-1}(\ddot{\mathbf{x}}_{1d} - \lambda \dot{\mathbf{e}} - K_1 \sqrt{|\mathbf{s}|}\,\text{sign}(\mathbf{s}) - K_2 \textstyle\int_0^t \text{sign}(\mathbf{s}(\varepsilon))d\varepsilon - \mathbf{f}(\mathbf{x_1}, \mathbf{x_2})) \tag{3.41}$$

where $\ddot{\mathbf{x}}_{1d}$ is the desired acceleration provided by the trajectory generator, $K_1 = \text{diag}(K_{1z}, K_{1\phi}, K_{1\theta}, K_{1\psi})$ and $K_2 = \text{diag}(K_{2z}, K_{2\phi}, K_{2\theta}, K_{2\psi})$ are super twisting control positive gain matrices, and sign(.) is the sign function. Thus we obtain the following closed-loop error dynamics (A more detailed process is given in the Section 3.4.4):

$$\dot{\mathbf{s}} = -K_1 \sqrt{|\mathbf{s}|}\,\text{sign}(\mathbf{s}) - K_2 \int_0^t \text{sign}(\mathbf{s}(\varepsilon))d\varepsilon \tag{3.42}$$

To prove the stability and the time convergence of this system, we define the variable $\zeta = [\zeta_1, \zeta_2]^T = [\mathbf{s}, -K_2 \int_0^t \text{sign}(\mathbf{s})d\varepsilon]^T$, then the equation (3.42) can be rewritten as:

$$\begin{aligned} \dot{\zeta}_1 &= -K_1 \sqrt{|\zeta_1|}\,\text{sign}(\zeta_1) + \zeta_2 \\ \dot{\zeta}_2 &= -K_2\,\text{sign}(\zeta_1) + \dot{\mathbf{w}} \end{aligned} \tag{3.43}$$

The proof of finite convergence of the variables ζ_1 and ζ_2 follows the same procedure as in [50]. It is worth noting that the translational motion in the x-axis and y-axis depends on the pitching and rolling torques, so the development of the control inputs is divided into an altitude and heading control upon which depends an X-Y translational motion control, as detailed in the following sections.

3.4.3 Altitude and heading control

We define the altitude tracking error, which is the difference between the desired and the measured altitudes as follows:

$$e_z = z_d - z \tag{3.44}$$

We also define the sliding variables for these errors:

$$s_z = \dot{e}_z + \lambda_z e_z \tag{3.45}$$

In this case, the dynamics of the altitude sliding variable s_z is given by:

$$\dot{s}_z = \ddot{e}_z + \lambda_z \dot{e}_z \tag{3.46}$$

By substituting (2.24) and (3.44) into (3.46), we obtain:

$$\dot{s}_z = \ddot{z}_d + G - \frac{(\cos\theta\cos\phi)}{m} u_f - w_z + \lambda_z \dot{e}_z \tag{3.47}$$

As previously stated in section 3.4.2.1, the control input u_t is composed of two parts: a continuous part called equivalent input which is responsible of the reaching phase, and a discontinuous part based on the super twisting algorithm (3.48) which is responsible of the convergence phase.

$$u_f = u_{feq} + u_{fdis} \tag{3.48}$$

The equivalent input u_{feq} is computed by applying the sliding condition, $\dot{s}_z = 0$, and the discontinuous part u_{fdis} is designed to obtain the closed-loop error dynamics as given in (3.42) in each subsystem. By applying the sliding condition to the altitude i.e. $\dot{s}_z = 0$, we obtain from (3.47) a sliding property that we want u_{feq} to satisfy:

$$\ddot{z}_r + G - \frac{(\cos\theta\cos\phi)}{m} u_{feq} - w_z + \lambda_z \dot{e}_z = 0 \tag{3.49}$$

As the matched uncertainties contained in w_z will be tolerated by the super twisting controller, we thus choose for the equivalent input:

$$u_{feq} = \frac{m}{(\cos\theta\cos\phi)} (\ddot{z}_d + G + \lambda_z \dot{e}_z) \tag{3.50}$$

Note however that if we had unmatched uncertainties in w_z, we would have them added to the expression of u_{feq}. This is not the case here because the altitude z of the system is controlled directly using the thrust input u_f. For the discontinuous part, we choose as given by the super twisting algorithm:

$$u_{fdis} = -K_{1z}\sqrt{|s_z|}\,\mathrm{sign}(s_z) - K_{2z}\int_0^t \mathrm{sign}(s_z(\varepsilon))d\varepsilon \tag{3.51}$$

By adding the equivalent term and the discontinuous term, the full expression of the control input becomes:

$$u_f = \frac{m}{(\cos\theta\cos\phi)} (\ddot{z}_r + G + \lambda_z \dot{e}_z) - K_{1z}\sqrt{|s_z|}\,\mathrm{sign}(s_z) - K_{2z}\int_0^t \mathrm{sign}(s_z(\varepsilon))d\varepsilon \tag{3.52}$$

By choosing the altitude control gains as follows:

$$\begin{aligned} K_{1z} &= \frac{mK'_{1z}}{(\cos\theta\cos\phi)} \\ K_{2z} &= \frac{mK'_{2z}}{(\cos\theta\cos\phi)} \end{aligned} \tag{3.53}$$

and by substituting (3.53) into (3.52), then finally by substituting the control law (3.52) in (3.47), the dynamics of s_z becomes:

$$\dot{s}_z = -K'_{1z}\sqrt{|s_z|}\,\mathrm{sign}(s_z) - K'_{2z}\int_0^t \mathrm{sign}(s_z(\varepsilon))d\varepsilon + w_z \tag{3.54}$$

Which satisfy having the same form as the equation (3.42), thus assuring the stability of the controller.

The same procedure is used to obtain the yaw torque:

$$\tau_\psi = I_{zz}(\ddot{\psi}_r + \lambda_\psi \dot{e}_\psi - \tfrac{(I_{xx}-I_{yy})}{I_{zz}}\dot{\phi}\dot{\theta}) - K_{1\psi}\sqrt{|s_\psi|}\,\mathrm{sign}(s_\psi) - K_{2\psi}\int_0^t \mathrm{sign}(s_\psi(\varepsilon))d\varepsilon \tag{3.55}$$

3.4.4 Translational motion in X-Y direction

In this subsection, the procedure to determine the roll and pitch torques is introduced. The first challenge is to obtain the desired roll and pitch variables from the position dynamic model. We define the following tracking errors:

$$\begin{aligned} e_x &= x_d - x \\ e_y &= y_d - y \end{aligned} \tag{3.56}$$

By applying the sliding conditions, i.e. $\dot{s}_x = 0$, $\dot{s}_y = 0$, we get:

$$\begin{aligned} \ddot{x}_d - \frac{(u_f)}{m} u_{xeq} - \frac{F_x}{m} + \lambda_x \dot{e}_x = 0 \\ \ddot{y}_d - \frac{(u_f)}{m} u_{yeq} - \frac{F_y}{m} + \lambda_y \dot{e}_y = 0 \end{aligned} \tag{3.57}$$

Where the wind forces F_x and F_y represent the only unmatched uncertainties in the terms w_x and w_y. The terms u_{xeq} and u_{yeq} are given by:

$$\begin{aligned} u_{xeq} = \cos\phi \cos\psi \sin\theta + \sin\phi \sin\psi \\ u_{yeq} = \cos\phi \sin\psi \sin\theta - \sin\phi \cos\psi \end{aligned} \tag{3.58}$$

Following the same procedure as in the previous section, we define:

$$\begin{aligned} \frac{u_f}{m} u_x = \ddot{x}_d - \frac{F_x}{m} + \lambda_x \dot{e}_x - K_{1x} \sqrt{|s_x|}\, \mathrm{sign}(s_x) - K_{2x} \int_0^t \mathrm{sign}(s_x(\varepsilon)) d\varepsilon \\ \frac{u_f}{m} u_y = \ddot{x}_d - \frac{F_y}{m} + \lambda_y \dot{e}_y - K_{1y} \sqrt{|s_y|}\, \mathrm{sign}(s_y) - K_{2y} \int_0^t \mathrm{sign}(s_y(\varepsilon)) d\varepsilon \end{aligned} \tag{3.59}$$

The desired roll and pitch can be determined from u_x and u_y and by using the desired heading:

$$\begin{aligned} \phi_d = \arcsin(\sin\psi_d u_x - \cos\psi_d u_y) \\ \theta_d = \arcsin(\frac{\cos\psi_r u_x + \sin\psi_d u_y}{\cos\phi_d}) \end{aligned} \tag{3.60}$$

Thus, we can define the roll and pitch errors by:

$$\begin{aligned} e_\phi = \phi_d - \phi \\ e_\theta = \theta_d - \theta \end{aligned} \tag{3.61}$$

These values are used in order to calculate the appropriate torques as follows:

$$\begin{aligned} \tau_\phi = I_{xx}(\ddot{\phi}_d + \lambda_\phi \dot{e}_\phi - \frac{(I_{yy}-I_{zz})}{I_{xx}} \dot{\theta}\dot{\psi} - K_{1\phi} \sqrt{|s_\phi|}\, \mathrm{sign}(s_\phi) - K_{2\phi} \int_0^t \mathrm{sign}(s_\phi(\varepsilon)) d\varepsilon \\ \tau_\theta = I_{yy}(\ddot{\theta}_d + \lambda_\theta \dot{e}_\theta - \frac{(I_{zz}-I_{xx})}{I_{yy}} \dot{\phi}\dot{\psi} - K_{1\theta} \sqrt{|s_\theta|}\, \mathrm{sign}(s_\theta) - K_{2\theta} \int_0^t \mathrm{sign}(s_\theta(\varepsilon)) d\varepsilon \end{aligned} \tag{3.62}$$

Based on the **Assumption SMC5**, the desired linear and angular accelerations as well as the desired angular velocities in the control inputs expressions can be ignored.

Thus the expression of the torques becomes:

$$\begin{aligned} \tau_\phi = I_{xx}(\lambda_\phi \dot{\phi} - \frac{(I_{yy}-I_{zz})}{I_{xx}} \dot{\theta}\dot{\psi} - K_{1\phi} \sqrt{|s_\phi|}\, \mathrm{sign}(s_\phi) - K_{2\phi} \int_0^t \mathrm{sign}(s_\phi(\varepsilon)) d\varepsilon \\ \tau_\theta = I_{yy}(\lambda_\theta \dot{\theta} - \frac{(I_{zz}-I_{xx})}{I_{yy}} \dot{\phi}\dot{\psi} - K_{1\theta} \sqrt{|s_\theta|}\, \mathrm{sign}(s_\theta) - K_{2\theta} \int_0^t \mathrm{sign}(s_\theta(\varepsilon)) d\varepsilon \end{aligned} \tag{3.63}$$

3.5 Shared control law for the fault tolerant control strategies

In this section, we present four aspects of the control law that will be shared between three FTC controllers in Chapter 5 to compare their fault tolerance. The first point is

the determination of subsystems for the control laws. This decomposition is very useful in order to simplify the design of the controllers and the FTC strategies in Chapter 5. Alongside with the subsystems decomposition, the second point is to present the common sliding manifold for the SMC of the three FTC. The objective of the SMC controller is to achieve a desired dynamics of the sliding manifold in order to drive the system errors to zeros. The third point is the modelization of the control inputs and the motors health. In this modelization, we include the relationship between the virtual inputs, the real control inputs and the coefficients representing the losses in control effectiveness in case of motor failures. Finally, the fourth point is to define the control allocation problem for the coaxial octorotor, and how to find the solutions of the real control inputs given the virtual control inputs.

3.5.1 Common subsystems formulation and sliding manifolds for all FTC schemes

Here, we present a decomposition of the dynamic model of the UAV into four subsystems (Altitude, Roll, Pitch and Yaw subsystems). This is done in order to simplify the design of the controller and it is considered as an alternative of the work done in Sections 3.4.3 and 3.4.4.

Since the controller stabilizes both the altitude z and the attitude ϕ, θ, ψ of the octorotor, we will consider the state vector as

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \end{bmatrix}^T \\ &= \begin{bmatrix} z & \dot{z} & \phi & \dot{\phi} & \theta & \dot{\theta} & \psi & \dot{\psi} \end{bmatrix}^T \end{aligned} \tag{3.64}$$

Given the nonlinear equations of motion in (2.24), and by ignoring the effect of propeller gyroscopic effect and drag forces (from our fifth assumption in Section 3.4.2.2), the octorotor dynamic model can be divided into four subsystems as follows:

$$\text{Altitude subsystem} \begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = f_1 + g_1 u_1 \end{cases} \tag{3.65}$$

$$\text{Roll subsystem} \begin{cases} \dot{x}_3 = x_4 \\ \dot{x}_4 = f_2 + g_2 u_2 \end{cases} \tag{3.66}$$

$$\text{Pitch subsystem} \begin{cases} \dot{x}_5 = x_6 \\ \dot{x}_6 = f_3 + g_3 u_3 \end{cases} \tag{3.67}$$

$$\text{Yaw subsystem} \begin{cases} \dot{x}_7 = x_8 \\ \dot{x}_8 = f_4 + g_4 u_4 \end{cases} \tag{3.68}$$

where f_i and g_i are defined as:

$$\begin{aligned} f_1 &= -G \\ f_2 &= x_6 x_8 (I_{yy} - I_{zz})/I_{xx} \\ f_3 &= x_4 x_8 (I_{zz} - I_{xx})/I_{yy} \\ f_4 &= x_4 x_6 (I_{xx} - I_{yy})/I_{zz} \\ g_1 &= \cos(x_3)\cos(x_5)/m \\ g_2 &= 1/I_{xx} \\ g_3 &= 1/I_{yy} \\ g_4 &= 1/I_{zz} \end{aligned} \tag{3.69}$$

With the formulation above, and for control design purposes, it is useful to represent each subsystem as a single input nonlinear system given by:

$$\begin{cases} \dot{x}_{2i-1} = x_{2i} \\ \dot{x}_{2i} = f_i + g_i u_i \end{cases} \text{ with } i = 1,2,3,4 \tag{3.70}$$

Next, we present the sliding manifold for a UAV's sliding mode controller. This sliding manifold will be used on the three presented FTCs, in order to compare only the impact of the fault tolerant mechanisms as much as possible.

The integral sliding manifold S_i is proposed as follows:

$$S_i = \{\mathbf{X} \in \mathbb{R}^n \,|\, s_i(\mathbf{X}) = 0\} \tag{3.71}$$

The integral switching function $s(\mathbf{X})$ is defined as:

$$s_i(\mathbf{X}) = \dot{e}_i + \lambda_i e_i + k_i \int e_i dt - \lambda_i e_{i0} - \dot{e}_{i0} \tag{3.72}$$

where λ_i and k_i are the positive gains for the sliding variables and (3.73) gives e_i for (3.65), (3.66), (3.67) and (3.68) with $i = 1,..,4$, such that e_i and $\dot{e}_i$ represent the position and velocity errors between the real states x_i measured by the vehicle sensors, and the desired values x_i^d calculated by the trajectory generator. e_{i0} and $\dot{e}_{i0}$ are the initial errors.

$$\begin{aligned} \dot{e}_i &= x_{2i} - x_{2i}^d \\ e_i &= x_{2i-1} - x_{2i-1}^d \end{aligned} \tag{3.73}$$

3.5.2 Modelization of the control inputs and the motors health and Control allocation problem

First, let us consider the general nonlinear system model:

$$\begin{aligned} \dot{\mathbf{X}} &= \mathbf{f}(\mathbf{X}) + \mathbf{g}(\mathbf{X})\mathbf{u} \\ \mathbf{u} &= \mathbf{B}\mathbf{L}\mathbf{u}^* \end{aligned} \tag{3.74}$$

where $\mathbf{u}$ is the virtual control input, that is the desired thrust and the moments, $\mathbf{u}^*$ represents the real control input, that is the each motor's desired power, and $\mathbf{X} \in \mathbb{R}^n$ is the state vector of the system, defined in the next section. The nonlinear functions

$\mathbf{f}(\mathbf{X}) \in \mathbb{R}^n$ and $\mathbf{g}(\mathbf{X}) \in \mathbb{R}^n$ are respectively the drift vector field and the control vector field. $\mathbf{L} = \text{diag}(l_1, ..., l_m)$ with m is the number of actuators and $0 \leq l_i \leq 1$ represents the control effectiveness of the actuators, so if $l_i = 1$, the i-th actuators is working perfectly, whereas $l_i = 0$ means the complete failure of the i-th actuators, and $0 < l_i < 1$ denotes a partial failure on the i-th actuator. $B \in \mathbb{R}^{p \times m}$ with $p < m$ is the control effectiveness matrix, representing the relation between the virtual and real control inputs. Moreover, as $\mathbf{u}$ is bounded from our fourth assumption (**Assumption SMC4**), we can say from (3.74) that $\mathbf{u}^*$ belongs to a compact set Σ defined as:

$$\Sigma = \{\mathbf{u}^* \in \mathbb{R}^m \,|\, u^*_{imin} \leq u^*_i \leq u^*_{imax}\} \quad i = 1, ..., m \tag{3.75}$$

In the coaxial octorotor configuration, the UAV is equipped with eight actuators where each pair is on the same arm and has the same axle, but rotates in opposite directions (four motors rotating clockwise, four motors rotating counter-clockwise). Usually, FTC schemes take advantage of the added hardware redundancy in order to achieve fast responses when a motor failure occurs. During fault-free operation, the high level controller generates virtual control inputs $\mathbf{u} = \begin{bmatrix} u_f & \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$. This virtual input vector $\mathbf{u}$ is redistributed among the set of m healthy motors where $m \leq 8$. This redistribution is known as the control allocation. The control allocation problem is formulated as:

$$\begin{aligned} \mathbf{BLu}^*(t) &= \mathbf{u}(t) \\ u^*_{imin} \leq u^*_i &\leq u^*_{imax} \end{aligned} \tag{3.76}$$

where $\mathbf{u}^* = \begin{bmatrix} \omega_1^2 & \omega_2^2 & \dots & \omega_8^2 \end{bmatrix}^T$, and the control effectiveness matrix $\mathbf{B}$ is defined as:

$$\mathbf{B} = \begin{bmatrix} t_1 & \dots & t_8 \\ r_1 & \dots & r_8 \\ p_1 & \dots & p_8 \\ y_1 & \dots & y_8 \end{bmatrix} \tag{3.77}$$

with:

$$\begin{aligned} t_i &= \alpha_{ij} K_f S \\ r_i &= \pm \alpha_{ij} K_f S l \frac{\sqrt{2}}{2} \\ p_i &= \pm \alpha_{ij} K_f S l \frac{\sqrt{2}}{2} \\ y_i &= \pm K_t \end{aligned} \tag{3.78}$$

where r_i, p_i and y_i are of the same sign than the moment generated by the i-th motor, and the other parameters are defined in equations (2.12),(2.11), (2.26) and (2.19).

Since the coaxial octorotor is an over-actuated system, the control allocation problem has a finite number of solutions. In this regard, by taking into account some optimization criterion and actuators constraints (see Section 6.2 in [128]), it is possible to find

the optimal solution by considering the quadratic programming approach based on minimizing the control input as follows:

$$\begin{aligned} J = &\text{argmin}\, \mathbf{u}^{*T} W_i \mathbf{u}^* \\ &\text{such that: } \mathbf{B}\mathbf{u}^* = \mathbf{u} \end{aligned} \tag{3.79}$$

where J is the value of the function to be minimized, which has the explicit solution:

$$\mathbf{u}^* = W_i \mathbf{B}^T (\mathbf{B} W_i \mathbf{B}^T)^{-1} \mathbf{u} \tag{3.80}$$

where $W_i = W_i^T$ is a symmetric positive definite weighting matrix.

3.6 Summary

In this chapter, we presented the two control laws that we will be using in experiments to control the attitude and altitude of our multirotors in chapter 4. The first control law is a linear PID controller used in the open source autopilot Ardupilot. It has the advantage to be simple to apply and does not require heavy computations. In addition, it is suitable for a multirotor UAV flying at small speed and with small angles variations.

The second control law is a second-order sliding mode controller based on the Super-Twisting algorithm. Although we faced some difficulties while implementing this control law particularly caused by gain tuning and chattering, it can be used as a baseline controller for passive fault tolerance of the multirotor due to its robustness and insensitivity to uncertainties.

Also we presented four aspects of the shared control law used in the fault tolerant control strategies which will be discussed in chapter 5. These four aspects are:

- Common subsystems formulation for all FTC schemes
- Shared sliding manifold function for all FTC schemes
- Modelization of the control inputs and the motors failures
- Control allocation problem

Wind Force Compensation Strategy

Contents

This chapter describes the design and implementation of a wind force compensation strategy for a quadrotor. This strategy relies on a second order sliding mode controller based on the super twisting algorithm (STA) with a nonlinear observer. The robust STA controller ensures robustness to matched external disturbances and time varying, parametric and nonlinear uncertainties. To extend the robustness of the controller against wind perturbation, we propose the integration of a wind observer in the closed-loop system. This estimation will allow a better monitoring of the system's status than passive robustness, providing the opportunity for a recovery procedure such as an emergency landing when the external perturbations become too strong for the system. The effectiveness of the proposed strategy is compared to an adaptive gain controller through simulations and validated in real experiments on the DJI S500 quadrotor.

4.1 Non linear multi-rotor model with wind perturbations

The presence of wind forces causes an erroneous flight trajectory shifting the UAV away from the desired path. For this reason, such unknown disturbances must be included in the control design. In the nonlinear model (2.24), the wind effect is considered as only due to the air frame drag which does not reflect real flight conditions, since it affects also the propeller dynamics. In this chapter, we are going to include the full unknown induced forces and moments of wind disturbances in the nonlinear dynamic model of the quadrotor in order to extend the robustness of the controller. Thus, the nonlinear model of

the UAV presented in section 2.3.3 of Chapter 2 becomes as follows:

$$\begin{cases} m\ddot{x} &= (\cos\phi\sin\theta\cos\psi+\sin\phi\sin\psi)u_f+F_x \\ m\ddot{y} &= (\cos\phi\sin\theta\sin\psi-\sin\phi\cos\psi)u_f+F_y \\ m\ddot{z} &= (\cos\phi\cos\theta)u_f-mG+F_z \\ I_{xx}\ddot{\phi} &= \dot{\theta}\dot{\psi}(I_{yy}-I_{zz})-J_r\dot{\theta}\Omega_r+\tau_\phi+M_\phi \\ I_{yy}\ddot{\theta} &= \dot{\phi}\dot{\psi}(I_{zz}-I_{xx})+J_r\dot{\phi}\Omega_r+\tau_\theta+M_\theta \\ I_{zz}\ddot{\psi} &= \dot{\phi}\dot{\theta}(I_{xx}-I_{yy})+\tau_\psi+M_\psi \end{cases} \tag{4.1}$$

where $F_p=[F_x\ F_y\ F_z]^T$ and $M_r=[M_\phi\ M_\theta\ M_\psi]^T$ are the induced aerodynamic forces and moments acting on the quadrotor caused by external wind and gusts. It is worth mentioning that the aerodynamic moments M_r and the aerodynamic force F_z are named matched disturbances since they affect the system states, which are directly controlled by the motors, while only the forces F_x and F_y are unmatched ones [80]. The robust STA controller, by itself, is able to compensate only the matched disturbances, thus the forces F_x and F_y are not fully tolerated by the controller and can cause trajectory shifting and system instability in the case of aggressive winds.

4.2 Smooth sliding mode controller robust to external perturbations

The goal of this section is to propose a smooth second order sliding mode controller which enhances the tracking accuracy of the system despite the existence of external perturbations. The estimation of the external perturbation is done via a robust observer used to determine the value of these perturbations by comparing the real behavior of the system to the desired one. The estimated values of the external perturbation are then used by the sliding mode controller in order to retain the system stability in the presence of disturbance inputs.

In the following sections, we will first present the desired dynamics of the sliding variable (which is of relative degree equals to 1 in our case) and how to design a control law in order to obtain these desired dynamics, then we will introduce the conception of the disturbance observer and differentiator which will be used to estimate the wind perturbations and can also be used to estimate the derivatives of the disturbance up to the m-th order, finally we will describe the mechanism of the disturbance cancellation through a cooperation between the control law and the disturbance observer, which will be used to obtain the desired dynamics of the sliding variable.

4.2.1 Desired sliding variable dynamics

The problem addressed in this section is to design a smooth control law u for the robust controller drives to zero the sliding mode variable s and its derivative $\dot{s}$. First let us output (SISO) system with a relative degree of the sliding variable equal to 1, meaning that the control law u appears in the expression of the first derivative of the sliding variable. Thus, we have:

$$\dot{s}=d(t)+u \tag{4.2}$$

where $d(t)$ represents the external perturbation applied on the system and the condition $s = 0$ defines that the system motion is on the sliding surface. However, we will consider in the rest of this section the more general case of a sliding variable with a relative degree to the control input equals to m and apply it in our specific case of m equals 1 in section 4.3.

The goal is to use an observer that estimates the perturbation term $d(t)$, and then by designing a smooth control law u under the form $u = -d(t) + \rho$ that uses this estimation, we ensure that the system will converge to the desired destination in finite time despite the perturbation. The estimation procedure of $d(t)$ and ρ are given in sections 4.2.2 and 4.2.3 respectively. In the case of a SISO system with a relative degree to the sliding variable equal to m, we will prove that the following sliding variable s is finite time stable, allowing the system to converge to the desired values in finite time:

$$\begin{cases} s = \zeta_1 \\ \dot{\zeta}_1 = -\alpha_1 |\zeta_1|^{(p-1)/p} \operatorname{sign}(\zeta_1) + \zeta_2 \\ \dot{\zeta}_2 = -\alpha_2 |\zeta_1|^{(p-2)/p} \operatorname{sign}(\zeta_1) \end{cases} \tag{4.3}$$

where ζ_1 and ζ_2 are intermediate variables used to represent the dynamics of s, p is a positive integer equals to m +1 (m being the relative degree of the sliding variable to the control input, as previously stated), and α_1, α_2 are positive gains.

Proposition 1. *By choosing $p \geqslant 2$, $\alpha_1, \alpha_2 > 0$ the system (4.3) is finite time stable (asymptotically stable with a finite settling time), meaning that the errors between the desired values and the actual values of the system will converge to zero in finite time.*

Proof. Considering the following Lyapunov function candidate:

$$\begin{aligned} V &= \frac{\zeta_2^2}{2} + \int_0^{\zeta_1} \alpha_2 |\beta|^{1/3} \operatorname{sign}(\beta) d\beta \\ &= \frac{\zeta_2^2}{2} + \frac{p}{2p-2} \alpha_2 \zeta_1^{(2p-2)/p} \end{aligned} \tag{4.4}$$

By defining $\gamma = \begin{bmatrix} \zeta_1 & \zeta_2 \end{bmatrix}^T$, we obtain the first derivative of this Lyapunov function as follows:

$$\dot{V} = \frac{\partial V}{\partial \gamma} \dot{\gamma} \tag{4.5}$$

Then, by substituting ζ_1 and ζ_2 with those defined in our sliding surface in equations (4.3), we obtain:

$$\begin{aligned} &\dot{V} = \tfrac{\partial V}{\partial \gamma} \dot{\gamma} \\ &= \begin{bmatrix} \alpha_2 |\zeta_1|^{(p-2)/p} \operatorname{sign}(\zeta_1) & \zeta_2 \end{bmatrix} \cdot \begin{bmatrix} \zeta_2 - \alpha_1 |\zeta_1|^{(p-1)/p} \operatorname{sign}(\zeta_1) \\ -\alpha_2 |\zeta_1|^{(p-2)/p} \operatorname{sign}(\zeta_1) \end{bmatrix} \\ &= \alpha_2 \zeta_2 |\zeta_1|^{(p-2)/p} \operatorname{sign}(\zeta_1) - \alpha_1 \alpha_2 |\zeta_1|^{(2p-3)/p} \\ &\quad -\alpha_2 \zeta_2 |\zeta_1|^{(p-2)/p} \operatorname{sign}(\zeta_1) \\ &= -\alpha_1 \alpha_2 |\zeta_1|^{(2p-3)/p}. \end{aligned} \tag{4.6}$$

By applying the LaSalle theorem, the set of γ verifying the condition $\dot{V}(\gamma) = 0$ consists of the axis $\zeta_1 = 0$. However the unique solution that satisfies $\dot{\zeta}_1 = 0$ in the first equation of (4.3) is $\zeta_2 = 0$. So the largest invariant set is the origin $\zeta_1 = \zeta_2 = 0$, thus the asymptotic stability is verified, and the variables ζ_1 and ζ_2 converge both to zero. Subsequently, it is proved in [138] that the system (4.3) is homogeneous, in which case the asymptotic stability implies the finite-time stability of the system [17],[25],[90]. □

4.2.2 Disturbance Observer

From Eq. (4.2), it can be seen that the sliding variable s is sensitive to the unknown perturbation $d(t)$. In order to obtain a controller that can drives the system to the desired sliding surface (4.3), we need first to estimate the value of the perturbation via a robust observer.

Let us suppose that s and u are available in real time, that $d(t)$ is $m-1$ times differentiable so that $d^{(m-1)}(t)$ has a known Lipshitz constant $M > 0$. The following observer is proposed:

$$\begin{cases} \dot{z}_0 = v_0 + u, \\ v_0 = -\lambda_0 M^{1/(m+1)} |z_0 - s|^{m/(m+1)} \operatorname{sign}(z_0 - s) + z_1 \\ \dot{z}_1 = v_1 \\ . \\ . \\ . \\ \dot{z}_{m-1} = v_{m-1}, \\ v_{m-1} = -\lambda_{m-1} M^{1/2} |z_{m-1} - v_{m-2}|^{1/2} \operatorname{sign}(z_{m-1} - v_{m-2}) + z_m \\ \dot{z}_m = -\lambda_m M \operatorname{sign}(z_m - v_{m-1}) \end{cases} \tag{4.7}$$

Proposition 2. *Considering that s and u are measured with Lebesgue-measurable noises bounded, respectively, by $\varepsilon > 0$ and $k\varepsilon^{(m-1)/m}$, $k > 0$ being any fixed constant, and the parameters λ_i being chosen sufficiently large in the reverse order, the following inequalities are established in finite time for some positive constants μ_i, η_i depending exclusively on k and the choice of parameters [89],[138]:*

$$\begin{cases} |z_0 - s| \leq \mu_0 \varepsilon \\ . \\ . \\ . \\ |z_i - d^{i-1}(t)| \leq \mu_i \varepsilon^{(m-i+1)/(m+1)}, i = 1, ..., m \\ |v_j - d^j(t)| \leq \eta_i \varepsilon^{(m-j)/(m+1)}, j = 0, 1, ..., m-1 \end{cases} \tag{4.8}$$

Proof. The proof is similar to the one of the differentiator convergence presented in [89]. Let us introduce the notation:

$$\begin{cases} \sigma_0 = z_0 - s, \\ \sigma_1 = z_1 - d, \\ . \\ . \\ \sigma_m = z_m - d^m, \end{cases} \tag{4.9}$$

By substituting (4.7) and (4.3) in (4.9), the derivative of σ_0 can be obtained as follows:

$$\begin{aligned}\dot{\sigma}_0 &= -\lambda_0 M^{(1/m+1)}|\sigma_0|^{(m/(m+1)}\,\mathrm{sign}(\sigma_0) + z_1 + u - d - u\\ \dot{\sigma}_0 &= -\lambda_0 M^{(1/m+1)}|\sigma_0|^{(m/(m+1)}\,\mathrm{sign}(\sigma_0) + \sigma_1\end{aligned} \tag{4.10}$$

With absence of measurement noises, i.e. $\varepsilon = 0$ in (4.8), any solution of the system (4.9) satisfies the differential inclusion (4.11), which can be obtained by substituting (4.7) and (4.3) in (4.9):

$$\begin{cases}\dot{\sigma}_0 = -\lambda_0 M^{(1/m+1)}|\sigma_0|^{(m/(m+1)}\,\mathrm{sign}(\sigma_0) + \sigma_1\\ \dot{\sigma}_1 = -\lambda_1 M^{(1/m)}|\sigma_1 - \dot{\sigma}_0|^{((m-1)/m}\,\mathrm{sign}(\sigma_1 - \dot{\sigma}_0) + \sigma_2\\ .\\ .\\ \dot{\sigma}_{m-1} = -\lambda_{m-1} M^{(1/2)}|\sigma_{m-1} - \dot{\sigma}_{m-2}|^{(1/2}\,\mathrm{sign}(\sigma_{m-1} - \dot{\sigma}_{m-2}) + \sigma_m\\ \dot{\sigma}_m = -\lambda_m M\,\mathrm{sign}(\sigma_m - \dot{\sigma}_{m-1}) + \beta\\ \beta \in [-M, M]\end{cases} \tag{4.11}$$

The derivatives of σ_i are excluded here from the right-hand side [89]. The obtained inclusion does not "remember" anything on the unknown signal $d(t)$. Indeed, we need to know the value of the Lipchitz constant M in order to calculate the values of σ_i in this inclusion (similarly to (4.15)). Moreover, (4.11) coincides with the inclusion appearing in the proof of the Theorem 5 [89]. Thus the proof and the choice of the parameters are the same (λ_i must be sufficiently large). The inclusion is invariant with respect to the homogeneity transformation

$$G_\eta : (t, \sigma_i) \longrightarrow (kt, k^{m-i+1}\sigma_i), i = 0, ..., m \tag{4.12}$$

Thus it is homogeneous with the homogeneity degree -1. The measurement noises corresponding to σ and u belong to $[-\varepsilon, \varepsilon]$ and $[-k\varepsilon^{(m-1)/m}, k\varepsilon^{(m-1)/m}]$, respectively according to [89]. □

In the case of exact measurement, i.e. $\varepsilon = 0$, from the system (4.8) we conclude that the following equalities are established in finite time:

$$\begin{cases} z_0 = s,\\ z_1 = d(t),\\ .\\ .\\ z_i = v_{i-1} = d^{(i)},\ i = 1, ..., m\end{cases} \tag{4.13}$$

To better understand how the algorithm (4.7) work, let's take the case from (4.8) where $m = 2$. We obtain the following system:

$$\begin{cases}\dot{z}_0 = v_0 + u\\ v_0 = -\lambda_0 M^{1/(3)}|z_0 - s|^{2/(2+1)}\,\mathrm{sign}(z_0 - s) + z_1\\ \dot{z}_1 = v_1\\ v_1 = -\lambda_1 M^{1/2}|z_1 - v_0|^{1/2}\,\mathrm{sign}(z_1 - v_0) + z_2\\ \dot{z}_2 = -\lambda_2 M\,\mathrm{sign}(z_2 - v_1)\end{cases} \tag{4.14}$$

The discrete version of this system is as follows:

$$\begin{cases} z_0(k) = (v_0(k-1) + u(k-1))\Delta t + z_0(k-1) \\ v_0(k) = -\lambda_0 M^{1/(3)} |z_0(k-1) - s(k-1)|^{2/(2+1)} \operatorname{sign}(z_0(k-1) - s(k-1)) + z_1(k-1) \\ z_1(k) = v_1(k-1)\Delta t + z_1(k-1) \\ v_1(k) = -\lambda_1 M^{1/2} |z_1(k-1) - v_0(k-1)|^{1/2} \operatorname{sign}(z_1(k-1) - v_0(k-1)) + z_2(k-1) \\ z_2(k) = (-\lambda_2 M \operatorname{sign}(z_2(k-1) - v_1(k-1)))\Delta t + z_2(k-1) \end{cases} \tag{4.15}$$

If the system is stable in finite time, we can choose any initial conditions as we will converge towards a stable point anyway. Then, we choose zero as initial values for $z_0(0), z_1(0), z_2(0), v_0(0), v_1(0)$. After calculating the value of the input $u(0)$, the first equation of (4.14) is used to calculate the value of z_0 in the next iteration, that is $z_0(1)$, using the calculated value of $u(0)$ only (since $v_0(0)$ is zero in the first iteration). Next, the value of $v_0(1)$ is calculated in the second equation using the known gain value λ_0, the known Lipchtiz value of M, the initialized values of $z_0(0) = 0$ and $z_1(0) = 0$ and the measured value of $s(0)$. The other variables values $(z_1(1), v_1(1), z_2(1))$ are obtained following the same manner. In the second iteration (case of $k = 2$), the obtained variable values in the first iteration $(z_0(1), v_0(1), z_1(1), v_1(1), z_2(1))$ are then used instead of the initialized values to calculate the newest values of the variables in this iteration.

4.2.3 Cancellation of external perturbation

Having the sliding variable s under the form (4.2) with the perturbation $d(t)$ being $(m-1)$ differentiable with a known Lipchitz constant $M > 0$ of $d^{(m-1)}(t)$, the desired dynamics of the sliding variable (4.3) with $p = m+1, m \geqslant 1$ are achieved using the observer (4.8) via the following control law:

$$\begin{cases} u = -z_1 - \alpha_1 |s|^{m/(m+1)} \operatorname{sign}(s) + \varsigma \\ \dot{\varsigma} = -\alpha_2 |s|^{(m-1)/(m+1)} \operatorname{sign}(s) \end{cases} \tag{4.16}$$

In the case of exact measurement, the equality $z_1 = d(t)$ is achieved in finite time by the observer from the equations (4.14) and by substituting (4.16) in (4.2), the desired dynamics of the sliding variable (4.3) are established only after the convergence of the observer, meaning that the system motion is retained on the sliding surface. In practice, it is required to have the observer converging much faster than the controller, which can be done by choosing sufficient large gains for the observer. In our case, the tuning of the gains is done empirically by conducting several real tests.

4.3 Estimation of Wind Disturbances using the Nonlinear Observer

As previously stated in section 4.1, since the wind induced forces F_x and F_y are used to obtain the control law in (3.59), we need to take them into account. For that, we estimate them through a super-twisting observer. We only develop the observer for the estimation of F_x and F_y since they are not compensated directly by the control law unlike F_z and the

moment disturbances. Actually, the wind forces F_x and F_y represents the unknown terms of the perturbation $d(t)$ in (4.2).

In order to use the approach in [138] and [45] for the estimation of the unknown wind forces F_x and F_y using the available sensor measurements, we put the x-direction state model under the form:

$$\begin{cases} \dot{x} = V_x \\ m\dot{V}_x = \hat{u}_x u_f + F_x \end{cases} \tag{4.17}$$

where F_x represent the real unknown wind force in the x-direction that should be estimated using the observer, and u_f is the thrust force. The goal is to develop a virtual control law $\hat{u}_x$ which is able to stabilize the sliding manifold s_x defined in (4.18), by driving $s_x \longrightarrow 0$ and $\dot{s}_x \longrightarrow 0$, and to cancel out the perturbation F_x. To do this, we choose s_x as follows

$$s_x = V_x - V_{xd} \tag{4.18}$$

since we are using Ardupilot, the desired velocity is chosen as $V_{xd} = K_p(x - x_d)$, thus s_x becomes:

$$s_x = V_x - K_p(x - x_d) \tag{4.19}$$

where V_x is the velocity in the x-direction and K_{px} is the tracking gain. We choose a tracking gain $K_p x = 1$, a higher value of K_{px} will make the UAV more responsive to thrust inputs, but if K_{px} becomes too high, the UAV will oscillate quickly in roll and/or pitch. Moreover, a lower value of K_{px} will make it slower. This command will be followed by the virtual control law to allow the vehicle to track the desired trajectory. Moreover the (s_x) dynamics is of relative degree 1, since the control law $\hat{u}_x$ appears in the first derivative of s_x:

$$\dot{s}_x = \frac{u_t}{m}\hat{u}_x + \frac{F_x(t)}{m} - K_p(V_x - V_{xd}) \tag{4.20}$$

is under the form

$$\dot{s}_x = U + b_x(t) \tag{4.21}$$

where $U = \frac{u_t}{m}\hat{u}_x - K_p(V_x - V_{xd})$ is the control input and $b_x(t) = \frac{F_x(t)}{m}$ is the perturbation in the x direction. Thus the system (4.21) correspond to the general form defined in (4.2). It can be seen that the first derivative of $b_x(t)$ is given by:

$$\dot{b}_x(t) = \frac{\dot{F}_x(t)}{m} \tag{4.22}$$

We assume that the wind forces (and particularly F_x and F_y) are continuous and differentiable anytime, i.e. the wind perturbation are considered smooth with no abrupt changes, and its derivative has a Lipschitz constant by assuming that the disturbances and their derivative are bounded i.e. $|F_x| < F_x^{LIM}$ and $|\dot{F}_x| < \dot{F}_x^{LIM}$. Then the function $b_x(t)$ is also differentiable and has a Lipschitz constant L_x and the hypotheses of section 4.2.2 are thus verified in this system. The term F_x is estimated by defining the following system:

$$\begin{cases} \dot{z}_{0x} = v_{0x} + \frac{u_t}{m}u_x - K_p(\dot{x} - \dot{x}_d) \\ v_{0x} = -\alpha_{0x}|z_{0x} - s_x|^{2/3}\,\text{sign}(z_{0x} - s_x) + z_{1x} \\ \dot{z}_{1x} = v_{1x} \\ v_{1x} = -\alpha_{1x}|z_{1x} - v_{0x}|^{1/2}\,\text{sign}(z_{1x} - v_{0x}) + z_{2x} \\ \dot{z}_{2x} = -\alpha_{2x}\,\text{sign}(z_{2x} - v_{1x}) \\ \hat{F}_x/m = z_{1x} \end{cases} \tag{4.23}$$

where $v_i (i = 1,2)$ are intermediate variables used in the calculation. In absence of input noises, and by choosing $\alpha_{0x} \geq \lambda_{0x}(L_x)^{1/3} > 0$, $\alpha_{1x} \geq \lambda_{1x}(L)^{1/2} > 0$ and $\alpha_{2x} \geq \lambda_{2x}(L_x)^{1/2} > 0$, we obtain $\hat{F}_x = F_x$ after finite-time with a suitable choice of the positive gains λ_{0x},λ_{1x} and λ_{2x}. The system (4.23) correspond to the system (4.7) by taking $m = 2$, thus it is finite-time stable.

4.3.1 Smooth second order controller

Considering the controller developed in (4.16) and by choosing $m = 2$, we get the following smooth second order virtual control laws:

$$\begin{aligned} \frac{u_t}{m}\hat{u}_x = &-z_{1x} + K_p(V_x - V_{xd}) - K_{1x}|s_x|^{2/3}\,\mathrm{sign}(s_x) \\ &- K_{2x}\int_0^t |s_x(\varepsilon)|^{1/3}\,\mathrm{sign}(s_x(\varepsilon))d\varepsilon \end{aligned} \tag{4.24}$$

$$\begin{aligned} \frac{u_t}{m}\hat{u}_y = &-z_{1y} + K_p(V_y - V_{yd}) - K_{1y}|s_y|^{2/3}\,\mathrm{sign}(s_y) \\ &- K_{2y}\int_0^t |s_y(\varepsilon)|^{1/3}\,\mathrm{sign}(s_y(\varepsilon))d\varepsilon \end{aligned} \tag{4.25}$$

where the estimation of F_y is done by means of z_{1y} following the same procedure. The new desired roll and pitch angles become (4.26), which replaces those in Eq. (3.60).

$$\begin{aligned} \hat{\phi}_r &= \arcsin(\sin\psi_r\hat{u}_x - \cos\psi_r\hat{u}_y) \\ \hat{\theta}_r &= \arcsin\left(\frac{\cos\psi_r\hat{u}_x + \sin\psi_r\hat{u}_y}{C_{\hat{\phi}_r}}\right) \end{aligned} \tag{4.26}$$

4.3.2 Controller and observer in closed loop

It is proven in section 4.2 that the observer under the form (4.23) converges in a finite time when exact measurements are available. Hence, z_{1x} converges to the external disturbances $\hat{F}_x/m$ corresponding to the wind force's impact in the x-direction (same for z_{1y} and $\hat{F}_y/m$) in finite time. Thereafter, by substituting respectively z_{1x} and z_{1y} by $\hat{F}_x/m$ and $\hat{F}_y/m$ in (4.24) and (4.25), and by substituting the smooth controller (4.24) in (4.20), the sigma-dynamics (with $x_1 = \sigma_x$ and $x_2 = -\int_0^t K_{2x}|x_1(\varepsilon)|^{1/3}\,\mathrm{sign}(x_1(\varepsilon))d\varepsilon$) becomes:

$$\begin{cases} \dot{x}_1 = -K_{1x}|x_1|^{2/3}\,\mathrm{sign}(x_1) + x_2 \\ \dot{x}_2 = -K_{2x}|x_1|^{1/3}\,\mathrm{sign}(x_1) \end{cases} \tag{4.27}$$

This system correspond to the system (4.16) by taking $m = 2$, thus it is finite-time stable, i.e., it is asymptotically stable with finite settling time for any initial conditions. Thus, the control laws developed in (4.24) and (4.25) drive σ_x, σ_y, $\dot{\sigma}_x$ and $\dot{\sigma}_y$ to zero, which defines the system motion on the sliding surface and the system actual states will converge to the desired ones.

It can be noted that we use the saturation function sat instead of the sign function in both the observer and the control law in order to attenuate the chattering phenomenon

which appears when the sliding variable value approaches zero, and to a obtain a smoother estimation. The expression of the function sat is given by:

$$\text{sat}(x) = \begin{cases} 1 & \textbf{if } x > 1 \\ -1 & \textbf{if } x < -1 \\ x & \textbf{otherwise} \end{cases} \tag{4.28}$$

However, in real-world applications, false alarms may occur when the measurement of σ terms is imprecise due to sensor noises. For this reason, we define a detection threshold for the observer, as will be shown in section 4.4. Moreover, we consider the presence of wind disturbances in the system only when the absolute value of the estimated wind forces exceed this specific threshold. We choose 0.05N as the threshold value for our experiments. This threshold was identified empirically by conducting several tests. The reason for which we need to this threshold is that assumption of noiseless measurements (case of $\varepsilon = 0$) cannot be achieved in real world. Thus, in absence of external perturbations, the observer becomes sensitive to measurement noises and will eventually affect the system stability. Thus, the threshold 0.05N eliminates the output of the observer in this case.

Moreover, let us recall that another condition of the strategy is to make the observer much faster than the controller, and this highly depends on the precision of the sensors that are been used to measure the states of the system. This can be done by choosing sufficiently large gains for the observer. The easiest way to check if the gains are sufficient for the observer design is empirically after several experiments or simulations.

4.4 Simulation and Experimental validation

The quadrotor *DJI*500 presented in Section 2.4 is the experimental drone that we used to validate our wind observer and its integration in the control law presented in section 4.3. We used the Raspberry pi 3 + Navio2 open-source autopilot and the Arducopter flight stack to perform the experimentation and to show the effectiveness of our strategy.

$\lambda_{1z} = 2.5$	$K_{1z} = 4.5$	$K_{2z} = 4.5$
$\lambda_{1x} = 0.2$	$K_{1x} = 0.5$	$K_{2x} = 0.25$
$\lambda_{1y} = 0.2$	$K_{1y} = 0.5$	$K_{2y} = 0.25$
$\lambda_{1\phi} = 3.5$	$K_{1\phi} = 4.5$	$K_{2\phi} = 4.5$
$\lambda_{1\theta} = 3.5$	$K_{1\theta} = 4.5$	$K_{2\theta} = 4.5$
$\lambda_{1\psi} = 2.1$	$K_{1\psi} = 5$	$K_{2\psi} = 3.5$

Table 4.1 – Control gains

$\alpha_{0x} = 1$	$\alpha_{1x} = 0.5$	$\alpha_{2x} = 0.05$
$\alpha_{0y} = 1$	$\alpha_{1y} = 0.4$	$\alpha_{2y} = 0.045$

Table 4.2 – Observer gains

$\lambda_{1x,y,z} = 2.5, 2.5, 5$	$\omega_{x,y,z} = 4.1, 4.1, 6.8$
$\lambda_{2x,y,z} = 1.25, 1.25, 3.12$	$\eta_{x,y,z} = 1.03, 1.03, 3.25$
$\lambda_{3x,y,z} = 1.83, 1.83, 2.01$	$\beta_{x,y,z} = 0.5, 0.5, 0.8$

Table 4.3 – adaptive gains

The controllers parameters and the observer gains were determined empirically and are given respectively in Table 4.1 and Table 4.2.

In this section we will provide some simulation results to compare our proposed observer-based controller to the adaptative controller given in [122]. The adaptive controller gains used in these simulations are given in the Table 4.3. We will present real experimental results to validate the effectiveness of our controller compared to the standard PID controller.

Note that the adaptive controller used in our simulation is based on a second order sliding mode algorithm and it has high number of gains to be tuned as shown in Table 4.3. However, we couldn't tune these gains in real flight, since it is a very complicated process and requires a lot of time. Comparatively, the real gains for our observer-based STA controller were easy to find and close to those used in the simulation. We thus used the PID controller as a comparison because it is still widely used and determining the gains can be done automatically using the auto-tune mode of the Arducopter.

Note that the only theoretical drawback of the proposed strategy compared to classical controllers is an increased computational time due to the estimation of the external forces, which had no impact on our real flight experiments.

4.4.1 Simulation results (Adaptive STA vs Observer-based STA)

In the following simulations, we compare our proposed observer-based controller to an adaptative controller in terms of robustness against wind. The simulated UAV is required to follow a circular trajectory of 3 meters radius, starting from $\xi_i = [0,0,0]^T$.

In Fig. 4.1, we can clearly see that with small wind forces of 0.2 N both controllers follow closely the desired trajectory. Note that in real flights, mechanical constraints, model uncertainties and sensor noises would obviously affect negatively the followed trajectory.

In the second scenario presented in Fig. 4.2, an external wind perturbation of 5 N is applied. We can notice that both controllers succeed to compensate the wind effect and maintain the requested path of the system. However the results show that the observer-based STA controller is smoother in this case. Note that such strong winds would probably be difficult to navigate for the UAV in real conditions, and that the adaptative controller had good similar results up to this wind strength. Nonetheless, these simulations show more potential tolerance for our observer-based approach. By comparing the collected errors data of these simulations, given in Table 4.4, we can see that the use of the observer is an improvement over the adaptive method in terms of the robustness to the wind disturbances.

4.4.2 Outdoor experimental results (PID vs Observer-based STA)

In our real experiments, an on-board GPS is used as the UAV's localization system. The position control in this experiment is a Set-point control (see Section 3.2). The desired positions are the four points which form the edges of the rectangular path as shown in Fig. 4.3. The quadrotor has to fly to the desired position, regardless of the trajectory, starting

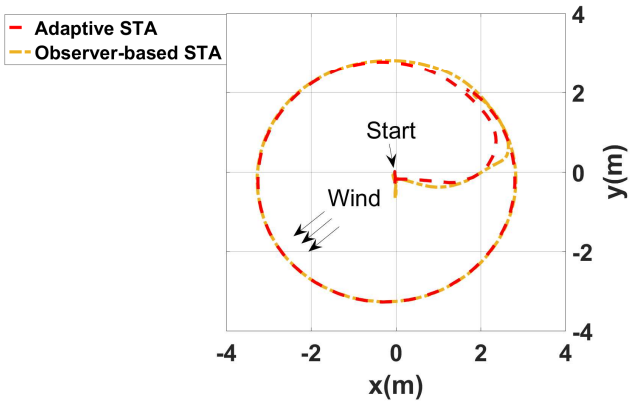


Figure 4.1 – Behavior of adaptive controller and observer-based controller with small wind perturbations of 0.2N (Simulation)

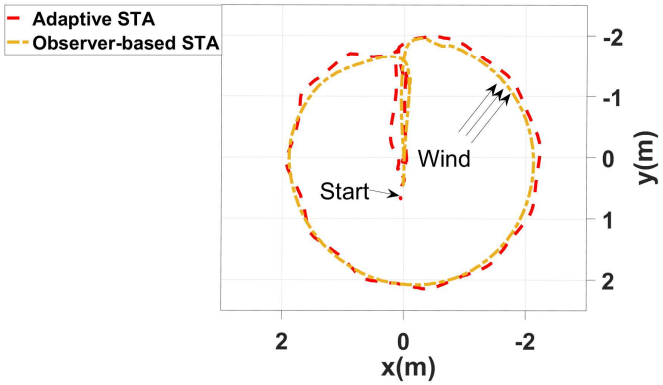


Figure 4.2 – Behavior of adaptive controller and observer-based controller with wind perturbations up to 5N (Simulation)

	Test 1	
	$\mu_{ex}(m)$	$\mu_{ey}(m)$
Observer-based STA	0.249	0.482
Adaptive STA	0.367	0.597
	Test 2	
	$\mu_{ex}(m)$	$\mu_{ey}(m)$
Observer-based STA	0.5404	0.508
Adaptive STA	0.801	0.673

Table 4.4 – Position mean squared errors in x-direction (μ_{e_x}) and y-direction (μ_{e_y}) for the simulations illustrated in Fig. 4.1 (Test 1) and in Fig. 4.2 (Test 2)

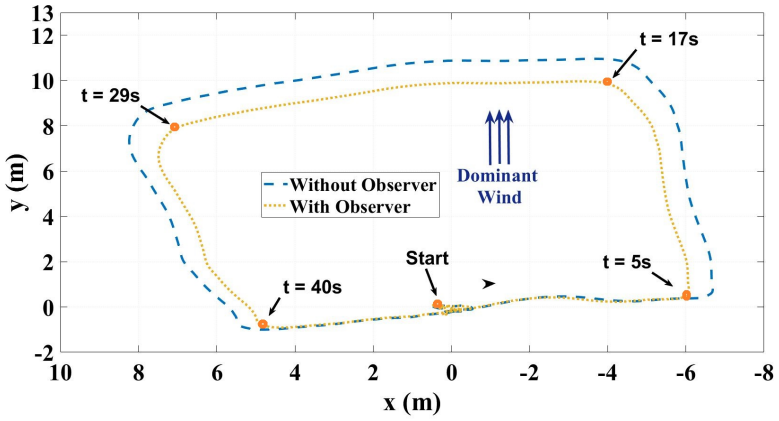


Figure 4.3 – Behavior of PID controller with and without Observer in the presence of wind perturbations (Experiment)

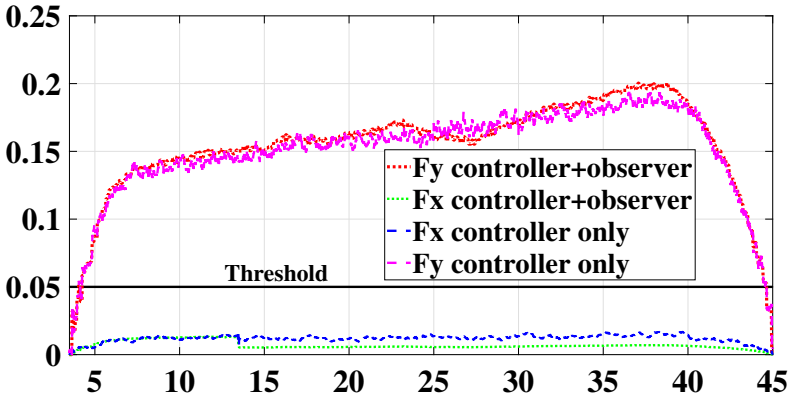


Figure 4.4 – Estimated wind forces FX and FY (N) in earth frame during the experiments (Experiment)

	Experiment	
	$\mu_{ex}(m)$	$\mu_{ey}(m)$
Controller with Observer	1.519	1.745
Controller without Observer	1.605	2.37

Table 4.5 – Position mean squared errors in x-direction (μ_{e_x}) and y-direction (μ_{e_y}) for the experiment illustrated in Fig. 4.3

from the initial position $\xi_i = [0,0,1]^T$ as shown in the Fig. 4.3.. The wind forces F_x and F_y (in Earth-Frame) are estimated in real time by the developed observer and are presented in the Fig. 4.4. We can notice that the wind force is higher in the Y direction where it reaches a peak of F_y=0.2 N. This mostly explains why the PID controller drifts from the desired trajectory, as it is unable to tolerate the wind perturbations. Our observer-based controller gives better results, as it is able to reach every set points. Since neither the observer model nor its measurement are perfect, we used a threshold on the observer's output to avoid small perturbations due to noise values. As long as the estimated wind force stays between the threshold values, we consider it 0, and as soon as it goes beyond them we integrate it in the control law. From indoor experiments without wind perturbations, we empirically chose threshold values of -0.05 and 0.05 N.

In order to compare the wind conditions in both experiments, we activated the observer in both cases to estimate the wind forces, but the disturbance rejections strategy is only applied for the observer-based STA controller. Fig. 4.4 shows that the wind forces were very similar during both experiments.

F_x is very small and below the defined threshold, and as previously stated this estimated force is not taken into account in the controller to avoid discontinuities due to the gain tuning and the hypothesis of the absence of measurement noises from the proof of section 4.2.2.

Despite the fact that the dominant wind is in the y direction, the pitch and roll of the UAV are affected (see equation (4.26)), which explains the PID controller's positions errors on the x axis that we see on Fig. 4.3.

By comparing the means and medians of the quadrotor's positions errors in Table 4.5, we can see that our proposed method improved the trajectory of the quadrotor compared to the PID controller.

However, supplementary tests and improvements are needed to completely validate this method: we only tested its effectiveness against moderate wind (up to 0.2 N), and we need to experiment in more aggressive environment where wind force can reach up to several newtons.

4.5 Conclusion and future works

In this chapter, we presented a strategy using an observer based on the Super Twisting Algorithm aiming to tolerate wind perturbations. The super Twisting controller is able to passively tolerate only matched uncertainties. The horizontal components of wind forces F_x and F_y are considered as unmatched uncertainties, thus we proposed an observer to estimate these perturbations. The estimations of this observer are then used in a STA controller to tolerate these wind perturbations. The robust controller and the nonlinear observer are both designed based on the STA to compensate for matched uncertainties such as unmodeled forces and imperfect UAV's parameters. We implemented this observer-based controller and validate it on real experiments. The proposed solution guarantees the convergence of the position errors for different values and directions of external airflow attacking the vehicle. This method was validated and compared with an adaptative controller through simulation, and a PID controller though real flights. In both cases, our proposed method gave better results although the adaptive method also gives

very good results in simulations. It is however very hard to implement in real experiments as it has numerous interdependent gains to tune. To our knowledge, it has never been presented in real experiments in this second order form, which is necessary to tolerate wind forces. Stability analysis of the observer controller loop has also been presented in the chapter.

FTC strategies for successive failures in an Octorotor UAV

Contents

This chapter presents three fault-tolerant control (FTC) strategies for a coaxial octorotor UAV regarding motor failures. The first FTC is based on a control mixing strategy which consists of a set of control laws designed offline, each one is dedicated to a specific fault situation. In the second FTC, a robust adaptive sliding mode control allocation is presented, where the control gains of the controller are adjusted online in order to redistribute the control signals among the healthy motors in order to stabilize the overall system. These two strategies have been introduced in our state of the art Section 1.6.2. The third FTC strategy is a new strategy proposed in this thesis, which is based on a self-tuning sliding mode control (STSMC) where the control gains are readjusted from the detected error to maintain the stability of the system. Multiple indoor experiments on an octorotor UAV are conducted to show and compare the effectiveness and the behavior of each FTC scheme after successive faults are injected.

5.1 Self-tuning sliding mode control applied to the coaxial octorotor

When a motor's failure occurs, the control allocation should redistribute in a specific way some control signals to the healthy actuators in order to compensate for the loss of thrust. This could be done by changing the weighting matrix W_i in the STSMC and control mixing techniques, or by modifying gains in the higher level controller u_i to attain the new stability conditions as it is done in the ASMCA and STSMC techniques. In this section, we will first present our contribution for FTC methods: a self-tuning sliding mode control (STSMC) that uses error detection to modify multiplexing and the control law's gains and to adapt to the erroneous state. Then, we will introduce two other FTC methods: a FTC based on off-line control mixing [129] and an adaptive sliding mode control allocation (ASMCA) for future comparison and validation of our proposed method.

5.1.1 Self-Tuning sliding mode control (STSMC)

We propose in this section an AFTC scheme which automatically tunes itself to an error once this error has been detected and identified. The detection and isolation can be done for example using additional current sensors equipped to each actuator (see chapter 5 in [128]). This FDI module must identify the faulty actuator and provide the effectiveness loss of each actuator for our method to be applied.

Let us recall the subsystems formulation presented in (3.65) to (3.68), the definition of the integral sliding variable in (3.72) and the definition of the system errors given in (3.73). The goal of the controller is to drive the sliding variable s_i and its first derivative to zero, which implies that the difference between the system's position and the desired one will also be driven to zero and the system will follow the desired trajectory. To do this, we design a control law u that ensure the following desired dynamics of the sliding variables s_i:

$$\dot{s}_i = -K_i \, \mathrm{sat}(s_i/\varepsilon_i) \tag{5.1}$$

where K_i are positive gains of the discontinuous part of the control law and ε_i are small positive values defining the boundaries of the linear part of the sat(.) function, defined as:

$$\mathrm{sat}(s_i/\varepsilon) = \begin{cases} \mathrm{sign}(s_i) & \text{if} \quad |\, s_i \,| > \varepsilon_i \\ s_i/\varepsilon_i & \text{if} \quad |\, s_i \,| < \varepsilon_i \end{cases} \tag{5.2}$$

From Eq. (3.72) and (3.73), the first derivative of s_i is calculated as:

$$\begin{aligned} \dot{s}_i &= \ddot{e}_i + \lambda_i \dot{e}_i + k_i e_i \\ &= (\dot{x}_{2i} - \dot{x}_{2i}^d) + \lambda_i \dot{e}_i + k_i e_i \end{aligned} \tag{5.3}$$

By substituting the expressions of $\ddot{x}_{2i}$ from the subsystems (3.65), (3.66), (3.67) and (3.68), we get:

$$\dot{s}_i = (f_i + g_i u_i - \ddot{x}_{2i}^d) + \lambda_i \dot{e}_i + k_i e_i \tag{5.4}$$

In order to obtain the desired dynamics of the sliding variable, we choose the control law as follows:

$$\begin{gathered} u_i = \tfrac{1}{g_i}(\dot{x}_{2i}^d - \lambda_i \dot{e}_i - k_i e_i - f_i) - \tfrac{1}{g_i} K_i \, \mathrm{sat}(s_i/\varepsilon_i) \\ \text{for } i = 1,..,4 \end{gathered} \tag{5.5}$$

By substituting this control law in Eq. (5.4), the desired dynamics of the sliding variable (5.1) are established.

Proposition 3. *Given the nonlinear subsystems (3.65), (3.66), (3.67) and (3.68), by applying the control law (5.5) and by choosing $K_i \geq \eta_i > 0$, where η_i is any fixed positive value, the condition $s_i(\mathbf{X}) = 0$ is satisfied and thus the system errors e_i and $\dot{e}_i$ converge to zeros, meaning that the system's state will eventually be the desired one.*

Proof. Consider the following Lyapunov function V_i :

$$V_i = \frac{1}{2} s_i^2 \tag{5.6}$$

Then the derivative of this function would be by using (3.72), (3.73) and (3.65) to (3.68):

$$\begin{aligned}\dot{V}_i &= s_i\dot{s}_i \\ &= s_i(\ddot{e}_i + \lambda_i\dot{e}_i + k_i e_i) \\ &= s_i(\dot{x}_{2i} - \dot{x}_{2i}^d + \lambda_i\dot{e}_i + k_i e_i) \\ &= s_i(f_i + g_i u_i - \dot{x}_{2i}^d + \lambda_i\dot{e}_i + k_i e_i)\end{aligned} \tag{5.7}$$

and by substituting (5.5) into (5.7), we get:

$$\begin{aligned}\dot{V}_i &= -K_i s_i \,\mathrm{sat}(s_i/\varepsilon) \\ &\leq -\eta_i s_i \,\mathrm{sat}(s_i/\varepsilon) \\ &\leq -\eta_i \mid s_i \mid\end{aligned} \tag{5.8}$$

Thus the system satisfies the η-reachability condition, and the system will asymptotically reach the desired trajectory. The proof is satisfied for $i = 1, ..., 4$. □

In faulty scenarios, actuators failures are modeled as losses in the control effectiveness. We represent these losses by the $\mathbf{L} = \mathrm{diag}(l_1, ..., l_8)$ matrix, where $l_i = 1$ and $l_i = 0$ indicate respectively that the i-th motor is fully healthy or completely failing. Then the real force $\overline{F_i}$ produced by each actuator is actually:

$$\overline{F_i} = l_i F_i \tag{5.9}$$

where F_i is the force produced in fault-free situation. In this situation, the real virtual input vector $\overline{\mathbf{u}} = \begin{bmatrix}\overline{u}_f & \overline{\tau}_\phi & \overline{\tau}_\theta & \overline{\tau}_\psi\end{bmatrix}^T$ produced by the system becomes:

$$\begin{bmatrix}\overline{u}_f \\ \overline{\tau}_\phi \\ \overline{\tau}_\theta \\ \overline{\tau}_\psi\end{bmatrix} = \begin{bmatrix}u_f \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi\end{bmatrix} + \begin{bmatrix}\Delta u_f \\ \Delta\tau_\phi \\ \Delta\tau_\theta \\ \Delta\tau_\psi\end{bmatrix} \tag{5.10}$$

where $\mathbf{u} = \begin{bmatrix}u_f & \tau_\phi & \tau_\theta & \tau_\psi\end{bmatrix}^T$ is the nominal virtual control input calculated by the controller and $\Delta\mathbf{u} = \begin{bmatrix}\Delta u_f & \Delta\tau_\phi & \Delta\tau_\theta & \Delta\tau_\psi\end{bmatrix}^T$ is the input caused by the erroneous actuators, or input fault vector. In practice, an error could be caused either by a damage affecting the motor or the propeller. In this work, we do not distinguish these faults.

Depending on the faulty actuator, additive faults in torques ($\Delta\tau_\phi$, $\Delta\tau_\theta$, $\Delta\tau_\psi$) can be either positive or negative depending on the direction of the actuator rotation and its position with respect to the center of gravity of the vehicle. In contrast to faults altering torques, the additive faults affecting the thrust (Δu_f) can only be negative since the generated thrust doesn't depend on the rotation direction nor on the actuator's position.

First, we consider that a single fault is present in the system. In this case, the additive faults input can be defined as:

$$\begin{aligned}
\Delta u_f &= -(1-l_j)F_j\alpha_{jk}S \\
\Delta\tau_\phi &= \pm(1-l_j)F_i\alpha_{jk}Sl\frac{\sqrt{2}}{2} \\
\Delta\tau_\theta &= \pm(1-l_j)F_i\alpha_{jk}Sl\frac{\sqrt{2}}{2} \\
\Delta\tau_\psi &= \pm(1-l_j)\tau_i
\end{aligned} \tag{5.11}$$

where S is the shape factor of the propeller (see section 2.3.1), l is the length of the arm given in Table 2.1, the j-th and k-th terms represent respectively the faulty actuators and its redundant actuator. Note that this notation will be adopted in the rest of this chapter. The signs of the additive faults in torques are given in Table 5.1.

Actuator$_i$	$\Delta\tau_\phi$	$\Delta\tau_\theta$	$\Delta\tau_\psi$
1	+	+	+
2	+	+	-
3	+	-	-
4	+	-	+
5	-	-	+
6	-	-	-
7	-	+	-
8	-	+	+

Table 5.1 – Additive faults in torques according to the failing actuator

A new control law $\mathbf{u}^f = \begin{bmatrix} u_f^f & \tau_\phi^f & \tau_\theta^f & \tau_\psi^f \end{bmatrix}^T$ is designed to recover the system from this faulty situation, such that:

$$\mathbf{u} = \mathbf{u}^f + \Delta\mathbf{u} \tag{5.12}$$

Thereby, the subsystems (3.65), (3.66), (3.67) and (3.68), become:

$$\text{Altitude subsystem} \begin{cases} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f_1 + g_1 u_f^f + g_1 \Delta u_f \end{cases} \tag{5.13}$$

$$\text{Roll subsystem} \begin{cases} \dot{x}_3 &= x_4 \\ \dot{x}_4 &= f_2 + g_2 \tau_\phi^f + g_2 \Delta\tau_\phi \end{cases} \tag{5.14}$$

$$\text{Pitch subsystem} \begin{cases} \dot{x}_5 &= x_6 \\ \dot{x}_6 &= f_3 + g_3 \tau_\theta^f + g_3 \Delta\tau_\theta \end{cases} \tag{5.15}$$

$$\text{Yaw subsystem} \begin{cases} \dot{x}_7 &= x_8 \\ \dot{x}_8 &= f_4 + g_4 \tau_\psi^f + g_4 \Delta\tau_\psi \end{cases} \tag{5.16}$$

Let us have:

$$\mathbf{u}^f = \begin{bmatrix} u_f^f & \tau_\phi^f & \tau_\theta^f & \tau_\psi^f \end{bmatrix}^T = \begin{bmatrix} u_1^f & u_2^f & u_3^f & u_4^f \end{bmatrix}^T \tag{5.17}$$

We propose the following control law to recover the system from erroneous situations:

$$\begin{aligned} u_i^f &= \frac{1}{g_i}(\dot{x}_{2i}^d - \lambda_i \dot{e}_i - k_i e_i - f_i) - \frac{1}{g_i} K_i^f \, \mathrm{sat}(s_i/\varepsilon_i) \\ K_i^f &= K_i + \Delta u_{ij}^{max}, \; i = 1, ..., 4 \end{aligned} \tag{5.18}$$

where Δu_{ij}^{max} is the additive fault term of the input fault vector Δu_j^{max} due to the fault injected to the j-th actuator given in Eq. (5.19). In fact, Δu_{ij}^{max} is positive and bounds the maximal value of Δu_i. Thus, by adding this positive value to the control gain K_i^f of the proposed control law (5.18), we can prove (using the Proposition 4 and its proof) that the system will be able to maintain its overall stability in case of actuator failures.

The expression of the input fault vector Δu_j^{max} is given by:

$$\Delta u_j^{max} = \begin{bmatrix} \Delta u_{fj}^{max} \\ \Delta \tau_{\phi j}^{max} \\ \Delta \tau_{\theta j}^{max} \\ \Delta \tau_{\psi j}^{max} \end{bmatrix} = \begin{bmatrix} \Delta u_{1j}^{max} \\ \Delta u_{2j}^{max} \\ \Delta u_{3j}^{max} \\ \Delta u_{4j}^{max} \end{bmatrix} = \begin{bmatrix} l_j F_j^{max} \alpha_{jk} S \\ l_j F_j^{max} \alpha_{jk} S l \frac{\sqrt{2}}{2} \\ l_j F_j^{max} \alpha_{jk} S l \frac{\sqrt{2}}{2} \\ l_j \tau_j^{max} \end{bmatrix} \tag{5.19}$$

where F_j^{max} and τ_j^{max} for our experimental drone's actuators have been identified in [128]. The l_j are the effectiveness losses detected by the FDI mechanism, as previously stated in this section, and the other terms are dependent of the system and are identified for our experimental UAV in section 5.2.2.

Proposition 4. *Given the nonlinear subsystems (3.65), (3.66), (3.67) and (3.68), by applying the new control law (5.18) and by choosing $K_i \geq \eta_i > 0$ in a similar way than for the fault-free control law, the condition $s_i(X) = 0$ is satisfied and thus the system errors e_i and $\dot{e}_i$ converge to zeros, meaning that the system's state will eventually be the desired one.*

Proof. Consider the following Lyapunov function V_i:

$$V_i = \frac{1}{2} s_i^2 \tag{5.20}$$

Then the derivative of this function would be

$$\begin{aligned} \dot{V}_i &= s_i \dot{s}_i \\ &= s_i(\ddot{e}_i + \lambda_i \dot{e}_i + k_i e_i) \\ &= s_i(\dot{x}_{2i} - \dot{x}_{2i}^d + \lambda_i \dot{e}_i + k_i e_i) \\ &= s_i(f_i + g_i u_i^f + g_i \Delta u_i - \dot{x}_{2i}^d + \lambda_i \dot{e}_i + k_i e_i) \end{aligned} \tag{5.21}$$

and by substituting (5.18) into (5.21), we get:

$$\begin{aligned} \dot{V}_i &= -K_i^f s_i \, \mathrm{sat}(s_i/\varepsilon) + g_i \Delta u_i s_i \\ &= -K_i s_i \, \mathrm{sat}(s_i/\varepsilon) - g_i \Delta u_{ij}^{max} s_i \, \mathrm{sat}(s_i/\varepsilon) + g_i \Delta u_i s_i \\ &= -K_i \mid s_i \mid - g_i(\Delta u_{ij}^{max} \mid s_i \mid - \Delta u_i s_i) \end{aligned} \tag{5.22}$$

Let us recall the expressions of g_i (from Eq. (3.69)):

$$\begin{aligned} g_1 &= \cos(\phi)\cos(\theta)/m \\ g_2 &= 1/I_{xx} \\ g_3 &= 1/I_{yy} \\ g_4 &= 1/I_{zz} \end{aligned} \tag{5.23}$$

Since the parameters I_{xx}, I_{yy} and I_{zz} are all positive, thus the functions g_2, g_3, g_4 are positives. Moreover, since the roll and pitch angles are constrained to $(-\pi/2 < \phi, \theta < \pi/2)$, and using the fact that m is the mass of the UAV which is a positive value, we can conclude that g_1 is also positive. Then, the following inequality holds true:

$$g_i > 0 \tag{5.24}$$

And, as Δu_{ij}^{max} is positive and bounds the maximal value of Δu_i, we can write:

$$\Delta u_{ij}^{max} > |\Delta u_i| \tag{5.25}$$

Thus, using the Eq. (5.24) and (5.24), we have:

$$\begin{aligned} \dot{V}_i &= -K_i \mid s_i \mid -g_i(\Delta u_{ij}^{max} \mid s_i \mid + \Delta u_i s_i) \\ &\leq -K_i \mid s_i \mid \\ &\leq -\eta_i \mid s_i \mid \end{aligned} \tag{5.26}$$

Then the system satisfies the η-reachability condition, and the system will asymptotically track the desired trajectory.

□

In case of multiple faults, the term Δu_{ij}^{max} is replaced by the sum of the input fault vectors corresponding to all the injected faults. The additive faults input for multiple faults thus becomes:

$$\begin{aligned} \Delta u_f &= -\sum_j (1-l_j) F_j \alpha_{jk} S \\ \Delta \tau_\phi &= \sum_j \pm (1-l_j) F_i \alpha_{jk} S l \frac{\sqrt{2}}{2} \\ \Delta \tau_\theta &= \sum_j \pm (1-l_j) F_i \alpha_{jk} S l \frac{\sqrt{2}}{2} \\ \Delta \tau_\psi &= \sum_j \pm (1-l_j) \tau_i \end{aligned} \tag{5.27}$$

where the j-th terms are the indices of all the failing actuators and, as we mentioned previously, the k-th terms represent the redundant actuators of the j-th failing actuators. The same proof of stability used in the case of one fault holds true in the case of multiple faults by adding the sum of the upper bounds $\sum_j \Delta u_{ij}^{max}$ of all the injected faults in equation (5.18).

Note that when an error occurs, the weighting matrix W_i, in Eq (3.80), is updated according to the output of the FDI module, i.e., the detected fault information, namely $w_i = 1/l_i$. When the i-th actuator becomes faulty, its weight value increases and its control signal also increases. If the control signal exceeds the power limit of the actuator, it will cause saturation. The following weighting algorithm $\hat{W}_i$ (proposed in [129]) is always used in (3.80) instead of W_i to avoid this actuators saturation::

$$\hat{W}_i = \Xi W_i \Xi \quad \text{with} \quad \Xi = \text{diag}(\xi_1, \xi_2, ..., \xi_8)$$

$$\xi_i = \begin{cases} (1+\eta_1)\dfrac{u_i^*}{\eta_2 u_{imax}^*}, & u_i^* > \eta_2 u_{imax}^* \\ 1, \quad \eta_2 u_{imin}^* \leq u_i^* 0 \leq \eta_2 u_{imax}^* \end{cases} \tag{5.28}$$

where η_1 and η_2 are small positive values with $0 < \eta_2 \leq 1$.

5.1.2 AFTC based on an offline control mixing

Since the coaxial octorotor configuration is an over-actuated system, there exists a finite number of solutions to the control allocation problem in equation (3.76). In [129], a static reallocation for each motors failures case is obtained by resolving the optimization problem (3.79) for each fault situation.

By considering that at least four motors are healthy, the number of complete failures combinations is $N_f = \sum_{i=1}^{4} C_4^i = 162$.

However, due to the symmetry in the coaxial octorotor's configuration, the number of dissimilar combinations is reduced to 27.

In this FTC scheme, the same baseline control law (5.5) is used in all cases, i.e., in nominal and faulty situations. However, as mentioned in [129], after each error detection, a new set of control gains depending of the new system's configuration must be associated to the control law. These gains must be determined empirically for every one of the erroneous configuration that we want to consider. For complete motors failures, the number of dissimilar configurations can be reduced to 27 as previously stated. However, if we want to take into account partial motors failures, this number can grow to hundreds.

5.1.2.1 Fault-Free Mode

In nominal behavior, the expression of the i-th motor speed ω_i is given by (5.29):

$$\omega_i = \sqrt{\frac{1}{8}.\left(\frac{u_f}{\alpha_{ij}.K_f.S} \pm \frac{\tau_\phi}{\alpha_{ij}.K_f.S.d\frac{\sqrt{2}}{2}} \pm \frac{\tau_\theta}{\alpha_{ij}.K_f.S.d\frac{\sqrt{2}}{2}} \pm \frac{\tau_\psi}{K_t}\right)} \tag{5.29}$$

In fact, the above expression of ω_i represents the solution of the equation (3.76) by taking $L = I_{8\times 8}$.

Failed Motors	Reallocated inputs
1 & 2	$F_i' = 0\ i = 5,6$ $F_i' = 2F_i\ i = 3,4,7,8$
1 & 3	$F_i' = 2F_i\ i = 2,4$ $F_i' = F_i\ i = 5,6,7,8$
1 & 4	$F_i' = 2F_i\ i = 2,3,5,8$ $F_i' = 0\ i = 6,7$
1 & 5	$F_i' = F_i\ i = 2,3,6,7$ $F_i' = 2F_i\ i = 4,8$
1 & 6	$F_i' = \frac{4}{3}F_i\ i = 2,3,4,5,7,8$
1 & 7	$F_i' = 2F_i\ i = 2,8$ $F_i' = F_i\ i = 3,4,5,6$
1 & 8	$F_i' = 2F_i\ i = 2,4,5,7$ $F_i' = 0\ i = 3,6$

Table 5.2 – Static reallocation in case of two motors failures

Failed Motors	Reallocated inputs
1 & 2 & (3 or 4 or 7 or 8)	—
1 & 2 & (5 or 6)	$F_i' = 2F_i\ i = 3,4,7,8$ $F_i' = 0\ i = 6$ or 5
1 & 3 & 4	—
1 & 3 & 5	$F_i' = 1.5F_i\ i = 2,6,8$ $F_i' = 2.5F_i\ i = 4$ $F_i' = F_i\ i = 7$
1 & 3 & (6 or 8)	$F_i' = 2F_i\ i = 2,4,5,7$ $F_i' = 0\ i = 8$ or 6
1 & 3 & 7	$F_i' = 2.5F_i\ i = 2$ $F_i' = 1.5F_i\ i = 4,6,8$ $F_i' = F_i\ i = 5$
1 & 4 & 5	—
1 & 4 & (6 or 7)	$F_i' = 2F_i\ i = 2,3,5,8$ $F_i' = 0\ i = 7$ or 6
1 & 5 & 6	$F_i' = 2F_i\ i = 3,4,7,8$ $F_i' = 0\ i = 2$
1 & 5 & 7	$F_i' = 2.5F_i\ i = 8$ $F_i' = 1.5F_i\ i = 2,4,6$ $F_i' = 1F_i\ i = 3$
1 & 6 & 7	$F_i' = 2F_i\ i = 2,3,5,8$ $F_i' = 0\ i = 4$
1 & 6 & 8	$F_i' = 2F_i\ i = 2,4,5,7$ $F_i' = 0\ i = 3$
1 & 7 & 8	—

Table 5.3 – Static reallocation in case of three motors failures

5.1.2.2 One complete failure

To deal with the case of one motor's complete failure, the solution proposed in [129] from the resolution of 3.79 (with one $l_i = 0$) is to reduce the power of its dual to the half, and to increase the powers of the three upper remaining actuators by a factor of 1.5. For example considering a complete failure of motor 1 ($l_1 = 0$), we would have:

$$\begin{aligned} F_6^{'} &= 0.5F_6 \\ F_i^{'} &= 1.5F_i \quad i = 2,4,8 \\ F_i^{'} &= F_i \qquad i = 3,5,7 \end{aligned} \tag{5.30}$$

Failed Motors	Reallocated inputs
1 & 2 & 5 & 6	$F_i' = 2F_i \; i = 3,4,7,8$
1 & 3 & 5 & 7	$F_i' = 2F_i \; i = 2,4,6,8$
1 & 4 & 6 & 7	$F_i' = 2F_i \; i = 2,3,5,8$
2 & 3 & 5 & 8	$F_i' = 2F_i \; i = 1,4,6,7$
2 & 4 & 5 & 7	$F_i' = 2F_i \; i = 1,3,6,8$
2 & 4 & 6 & 8	$F_i' = 2F_i \; i = 1,3,5,7$
3 & 4 & 7 & 8	$F_i' = 2F_i \; i = 1,2,5,6$

Table 5.4 – Static reallocation in case of four motors failures

5.1.2.3 Two, three and four complete motors failures

As mentioned before, in case of multiple complete failures, many solutions of the optimization problem (3.79) are symmetrically equivalent since the frame configuration of the coaxial octorotor is symmetric. So, without loss of generality, only the cases where motors 1 and $i = 2,3,...,8$ fail are presented in Tables 5.2. In the same way, Tables 5.3 and 5.4 present only cases which are not symmetrically equivalent. Note that a dashed line indicates that no solution was found with the associated combination [129].

5.1.3 Adaptive sliding mode control allocation (ASMCA)

In this robust approach, an adaptive control allocation and re-allocation strategy [152] is proposed to redistribute control signals among healthy motors. This ASMCA is employed to maintain the overall system performance, providing tolerance to motors faults but also robustness to external perturbations that are matched uncertainties.

The proposed control law is as follows:

$$u_i = \hat{\Upsilon}_i(\dot{x}_{2i}^d - \lambda_i \dot{e}_i - k_i e_i - f_i) - \hat{\Upsilon}_i K_i * \text{sat}(s_i/\varepsilon_i) \tag{5.31}$$

The parameter $\hat{\Upsilon}_i$ is adaptively adjusted online when there is an error between the baseline controller u_i and the desired one u_i^d in the case of actuator faults or external perturbations. The online update scheme of the parameter $\hat{\Upsilon}_i$ is given by:

$$\dot{\hat{\Upsilon}}_i = (-\dot{x}_{2i}^d + \lambda_i \dot{e}_i + k_i e_i + f_i) + K_i * \text{sat}(s_i/\varepsilon_i) s_{\Delta_i} \tag{5.32}$$

where $s_{\Delta_i} = s_i - \varepsilon \, \text{sat}(s_i/\varepsilon)$. Further information about this algorithm are given in [152].

The main steps of the adaptive allocation are conducted as follows:

1. When there exists an error between the actual baseline controller u_i and the desired controller u_{id}, which means that the position and the velocity of the system are different from the desired values, thus the parameter $\hat{\Upsilon}_i$ is updated following the equation (5.32) to maintain the system performance.

2. The control effectiveness matrix B_i is updated accordingly through a defined relationship $\hat{B}_i = \hat{\Upsilon}_i * T_i$, where the relationship between g_i from equations (3.65) to (3.68) and B_i is bounded by $T_i = g_i B_i$.

3. The control signals are redistributed to the healthy motors through the relationship $u^* = W_i \hat{B}^T (\hat{B} W_i \hat{B}^T)^{-1} u$

It is proved in [152] that for a nonlinear system with bounded disturbance, the sliding motion will be achieved and maintained by applying the feedback control law (5.31) and the online adaptation scheme (5.32) and by an appropriate choice of the discontinuous gains K_i.

It is worth noting that this controller is based on a first order sliding mode with only 3 gains to tune for each input: λ_i, k_i and K_i. However, if we want to use an adaptive controller based on higher order sliding mode (for example one able to tolerate wind forces), the tuning process becomes more complicated since we will get several more control gains in the expression of the controller.

5.2 Indoor experimental Validation

In this section, we demonstrate the performance of our proposed STSMC. Also, we compare these performances with those of the other two FTC schemes described in the previous section in real indoor experiments. In these experiments, four successive fault injections are used to simulate the failures of top actuators of the coaxial octorotor (actuators 6, 2, 4 and 8) by sending stop commands at desired times.

5.2.1 Experimental platform

The experimental coaxial octorotor is shown the $Modulo - X_8$ presented in section 2.4. The maximum values of thrust force and moment generated by the actuators are given in Table 5.5. The other parameters were previously given in Table 2.1.

Symbole	Parameter	Value	Unit
F_i^{max}	Max forces	5	N
τ_ϕ^{max}, τ_θ^{max}	Max torque	2	Nm
τ_ψ^{max}	Max torque	0.08	Nm

Table 5.5 – Coaxial octorotor parameters

Gain	No fault	FI1	FI2	FI3	FI4
λ_1	3	3.3	3.7	4	4
λ_2	5	5.2	5.5	6	6
λ_3	5	5.2	5.5	6	6
λ_4	4	4.2	4.5	5	5
k_1	3	3.3	3.6	4.1	4.1
k_2	4	4.4	4.8	5.2	5.2
k_3	4	4.4	4.8	5.2	5.2
k_4	5	5.2	5.6	6.2	6.2
K_1	6	6.4	6.8	7.3	7.3
K_2	5	5.2	5.6	6.1	6.1
K_3	5	5.2	5.6	6.1	6.1
K_4	6	6.4	6.8	7.3	7.3

Table 5.6 – Gains values for each recovery configuration for the Multiplexing method

In order to compare the FTC methods, we use the same initial control parameters for all FTC control schemes, i.e. in (5.18) and (5.31):

$$\begin{aligned} \lambda_1 = 3, \quad \lambda_2 = 5, \quad \lambda_3 = 5, \quad \lambda_4 = 4 \\ k_1 = 3, \quad k_2 = 4, \quad k_3 = 4, \quad k_4 = 5 \\ K_1 = 6, \quad K_2 = 5, \quad K_3 = 5, \quad K_4 = 6 \\ \varepsilon = 0.15 \end{aligned} \tag{5.33}$$

Note that for the reconfigurable control mixing technique, we assign new control parameters after each failure injection to meet the new condition of the system. These gains for each recovery configuration are determined experimentally and are given in Table 5.6. On the other hand, for the STSMC and ASMCA, the initial control parameters (5.33) are adjusted online with respect to equations (5.18) and (5.31) respectively.

5.2.2 Fault detection and isolation (FDI) using current sensors

As mentioned before, the system recovery in AFTCs is depending upon the occurring errors in the system being detected and identified. Therefore, a fault detection and isolation (FDI) unit is required to detect the failed component and to estimate the fault parameters. Fault detection and isolation techniques can be classified into two categories, namely Model-based FDI and Model-free FDI.

In Model-based FDI, the detection, isolation and identification of faults are done by comparing the output of the system available measurements with *a priori* information represented by the dynamics equations of the system. In Model-free FDI, the detection relies only on real-time or historical data collected from the sensors and measurements in order to detect and isolate failed components without the need of any information about the system dynamics.

In our experiments, the FDI is supposed to be carried out by comparing each actuator's current to the current that it should theoretically have given the desired command (a Model

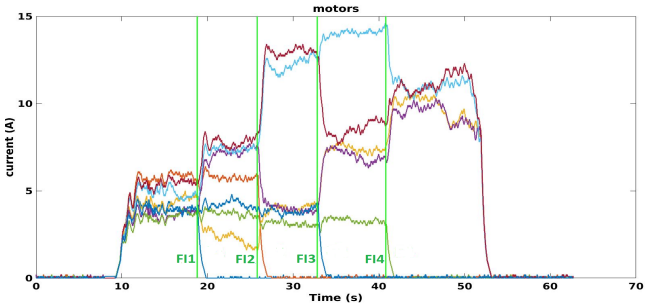


Figure 5.1 – Motors currents during real hover flight and successive failures injections (FI)

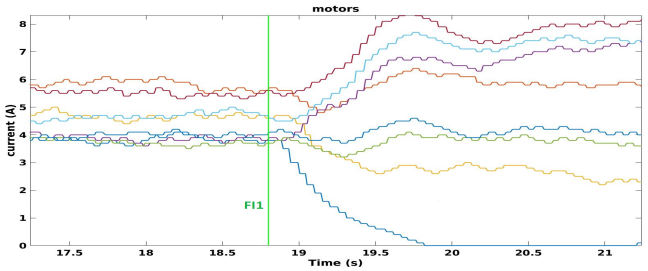


Figure 5.2 – Zoomed view of motors currents during real hover flight and successive failures injections (FI)

Based FDI). This technique was successfully tested in [128], but we only simulated it in our experiments in order to focus on the reconfiguration problem of the FTC.

Fig. 5.1 and 5.2 illustrate the motor currents during a real hover flight. Note that after the injection of the first failure, the current measurements take about one second before converging to zero. We chose in our experiments to fix the simulated error detection time at 0.5s as we consider from this data that an 80% decrease of motor current could be correlated to the complete failure of the actuator.

5.2.3 Experimental Results

To compare the three FTC schemes, two indoor scenarios are considered:

- The coaxial octorotor is required to do a stable flight at 1 meter altitude.
- The octorotor is required to follow a squared trajectory of 2 meters sides at the same altitude of 1 meter.

For each FTC scheme, we realized 10 experimental tests of each flight, thus a total of 60 flight tests were carried out. We needed to do this number of experiments as they are not completely reproductible because of the complexity of the system. Ten experiments per method allow us to have a good representativity while being feasible under our resources and time constraints.

To study this number of experimentations, we plot the results using the box and whisker technique, since it is able to summarize a set of data and is ideal for comparing distributions. In this representation the box lower and upper sides represent the two middle quartiles of data measurements (between 25% and 75%), while the whiskers refer to the

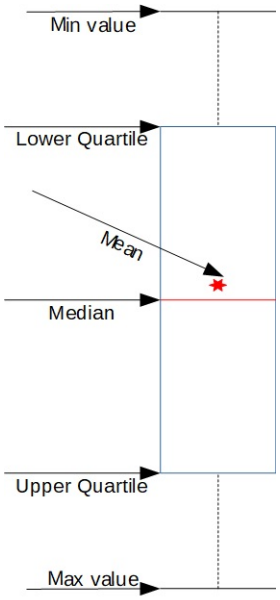


Figure 5.3 – Box and whisker plot

lower and higher quartile (respectively, between 0% and 25%, and between 75% and 100%). Finally, the mean and the median of the observations are represented respectively by a star and an horizontal line inside the box. The Figure 5.3 shows an example of box and whiskers with all the cited properties.

5.2.3.1 Hovering flight

To characterize the performance of the three FTC schemes in handling motor failures, first we performed 10 trials of the hovering scenario using each method (STSMC, Multiplexing, Adaptive). In each experiment, the top motors (successively 6, 2, 4 and 8) are turned off from the ground station at the same times $t_1 = 18.28s$, $t_2 = 25.8s$, $t_3 = 32.8s$ and $t_4 = 40.8s$. Position data of the different trials are collected between $t = 17s$ and $t = 45s$ right before the first injection, and after the last one. The three controllers were equally stable at the beginning of the data collection. All the data are illustrated in Fig. 5.4, Fig. 5.5 and Fig. 5.6. To simplify the analysis of the performance, we calculated the average positions of the trials for each FTC scheme, and we illustrate them in Fig. 5.7, tables 5.7 and 5.8. By comparing these results, it can be seen that in all cases the altitude errors are less important than the (x, y) position errors. This is because all the eight actuators contribute to the height control in the same direction and the loss of one motor is a loss of 1/8 of the total number of actuators in this direction. However, in the roll and pitch control we have 4 actuators generating positive orientations and the 4 remaining actuators generating negative orientations, and the loss of one motor is then a loss of 1/4 of the total number of actuators contributing to the same torque. The loss of one motor has thus more influence on the (x,y) position than on the altitude.

The altitude tracking performance of the trials averages is shown in Fig. 5.8.

5.2.3.2 Trajectory Tracking flight

To characterize the performance of the three FTC schemes in handling motor failures when following trajectories, we realized another flight scenario, where the coaxial octorotor has to follow a squared path starting from the initial position ($x = 0$, $y = 0$) as shown in Fig. 5.13. This is the case of a trajectory tracking problem presented in section 3.2, where the desired positions and velocities are time-dependent and the system is restricted to follow the squared reference and reach successively the ABCDA corners of the rectangle as shown in Fig. 5.13. Again, we performed 10 trials using each FTC scheme, where the failures to the top motors (successively 6, 2, 4 and 8) are injected in all experiments at the same times $t_1 = 18.28s$, $t_2 = 25.8s$, $t_3 = 32.8s$ and $t_4 = 40.8s$. Data on all experiments are illustrated in Fig. 5.9, Fig. 5.10 and Fig. 5.11, and the average positions and errors information are presented respectively in Fig. 5.12, Tables 5.9 and 5.10.

5.3 Discussion

In this section, we will discuss and compare the different behaviors and aspects of the three FTC schemes considering four aspects: performance, development costs, computation time and health monitoring.

5.3.1 Performance

All three FTCs give good results and successfully tolerate the four successive faults. However, it can be seen on Fig. 5.8 and 5.13 that compared to the Adaptive method, the Multiplexing and the STSMC have a smaller response time in compensating for the effects of the injected faults. This is probably due to the fact that the adaptation process of the adaptive method needs some time to converge to the required control parameters. Also, from Table 5.7, 5.8, 5.9 and 5.10, it can be shown that the lowest errors are recorded when using the Multiplexing method. This is probably due to its parameters being obtained by resolving an optimization problem, while the SMTC gains bounds the error (see equation (5.18)) and the adaptive gains are based on a comparison with noisy sensors outputs.

5.3.2 Development cost

The Multiplexing FTC method is very costly to develop since it requires to determine the control law's gains for every possible faults. This cost becomes even greater if we take into account partial failures in addition to complete ones. On the other hand, the STSMC and the Adaptive method can both be used to deal with all possible faulty scenarios without more development effort. Nevertheless, the STSMC still requires more knowledge about the model of the system, since a thruster model identification must be done to determine the thrust and drag coefficients of the actuator/propeller system in order to obtain the tuning law.

However, note that if we want to use an adaptive robust sliding mode controller to deal with unmatched uncertainties like wind forces in x and y direction (as seen in chapter 4), it will have to be based on second order sliding mode algorithm where the tuning process

of the gain becomes more complex since the number of gains increases and these gains need to be codependent to guarantee the stability of the system.

5.3.3 Computation time

The averages of the computation time of each FTC during an experiment (from $t = 15s$ to $t = 45s$) of the baseline controller for each FTC scheme are given in Table 5.11. It shows that the STSMC and the Adaptive methods are more expensive than the Multiplexing method in terms of computational complexity by a ratio of 1.5. This is certainly because they involve online computation of the controller parameters while the Multiplexing method just switch from one control law to another.

5.3.4 Health monitoring

An advantage of using active fault tolerance schemes (Multiplexing and STSMC) is that we can know if the system is getting close to its fault tolerance limit since we are monitoring the system health. This is not possible in the case of the Adaptative method, and therefore it may not be as reliable in critical cases. Nevertheless, the Adaptive method also adds more robustness to the system against external disturbances and does not need a FDI mechanism. Note however than an added FDI mechanism to the Adaptative method, even if not required, would allow to monitor the system's health in the same way than the two other methods.

5.4 Conclusion

Based on the experimental results we obtained, it can be concluded that the proposed STSMC, the Multiplexing, and the Adaptive fault tolerant control strategies all allow a coaxial octorotor to maintain stability after losing up to four motors.

Although the Multiplexing FTC scored the lowest position errors and time complexity, it is not efficient in development cost as every possible fault scenario must be studied both theoretically and experimentally to determine respectively the multiplexing parameters of the control law and its gains. Meanwhile, the proposed STSMC can be considered as a good intermediate alternative to both strategies, since it shows better performance and faster response time compared to the Adaptive method and allows to monitor the the system's health with almost no added development cost. It can also be easily implemented on a second order sliding mode controller to allow robustness against wind disturbances, unlike the adaptive method that requires complex gains tuning.

FTC	RMS e_x (m)	RMS e_y (m)	RMS e_z (m)
STSMC	0.0835	0.0777	0.0453
Multiplexing	0.0621	0.0707	0.0448
Adaptive	0.0975	0.0882	0.0614

Table 5.7 – hovering flight Root Mean Square errors

FTC	e_{xmax} (m)	e_{ymax} (m)	e_{zmax} (m)
STSMC	0.222	0.151	0.143
Multiplexing	0.204	0.139	0.135
Adaptive	0.213	0.161	0.175

Table 5.8 – hovering flight Max errors

FTC	RMS e_x (m)	RMS e_y (m)	RMS e_z (m)
STSMC	0.0882	0.0909	0.0553
Multiplexing	0.0723	0.0798	0.049
Adaptive	0.0950	0.1009	0.0616

Table 5.9 – Square Trajectory flight Root Mean Square errors

FTC	e_{xmax} (m)	e_{ymax} (m)	e_{zmax} (m)
STSMC	0.224	0.249	0.155
Multiplexing	0.218	0.239	0.141
Adaptive	0.281	0.238	0.180

Table 5.10 – Square Trajectory flight Max errors

FTC	CPU clock (s)
STSMC	0.110196
Multiplexing	0.075216
Adaptive	0.115344

Table 5.11 – Time complexity of each FTC method between $t_4 = 17s$ and $t_4 = 45s$ during an experiment.

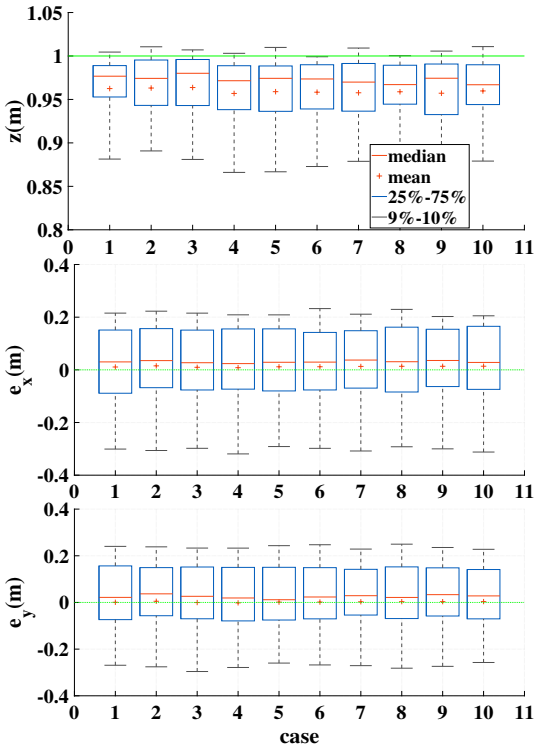


Figure 5.4 – Box and whisker plot for 10 trials of the Adaptive method after four successive failures in hover flight, showing the altitude z(m), and position errors e_x(m) and e_y(m) between t=17s and t=45s.

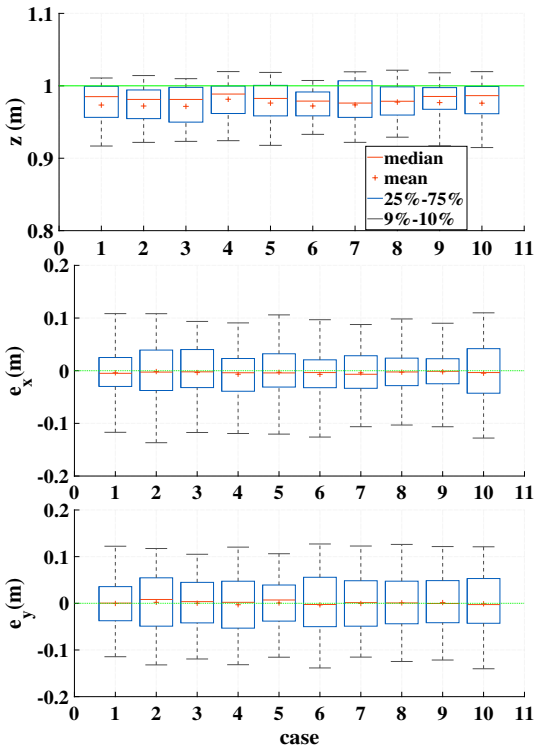


Figure 5.5 – Box and whisker plot for 10 trials of the Multiplexing method after four successive failures in hover flight, showing the altitude z(m), and position errors e_x(m) and e_y(m) between t=17s and t=45s.

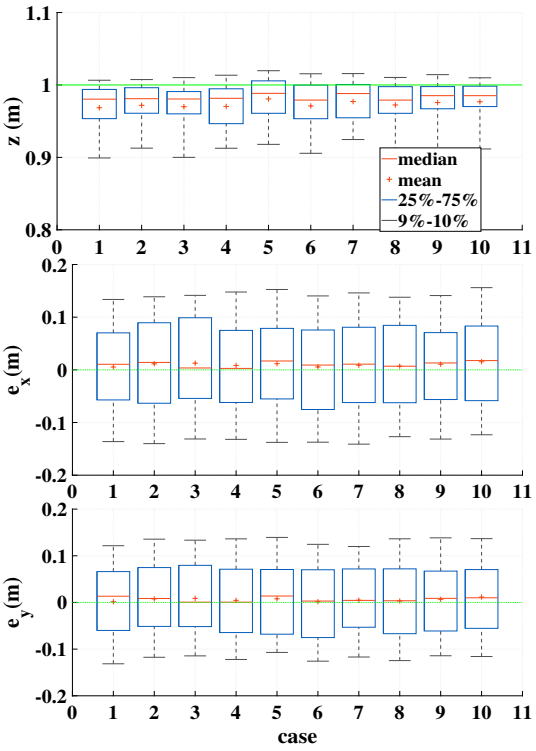


Figure 5.6 – Box and whisker plot for 10 trials of the STSMC method after four successive failures in hover flight, showing the altitude z(m), and positions x(m) and y(m) between t=17s and t=45s.

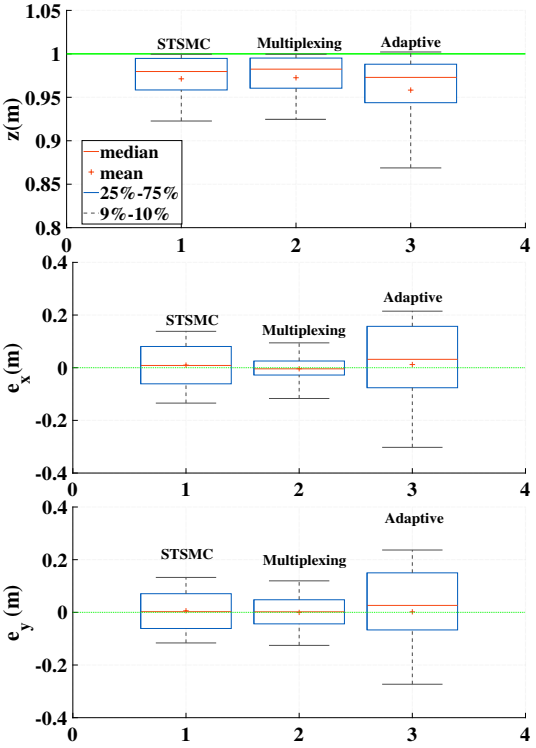


Figure 5.7 – Box and whisker plot for means of each FTC scheme trials after four successive failures in hover flight, showing the altitude z(m), and positions x(m) and y(m) between t=17s and t=45s.

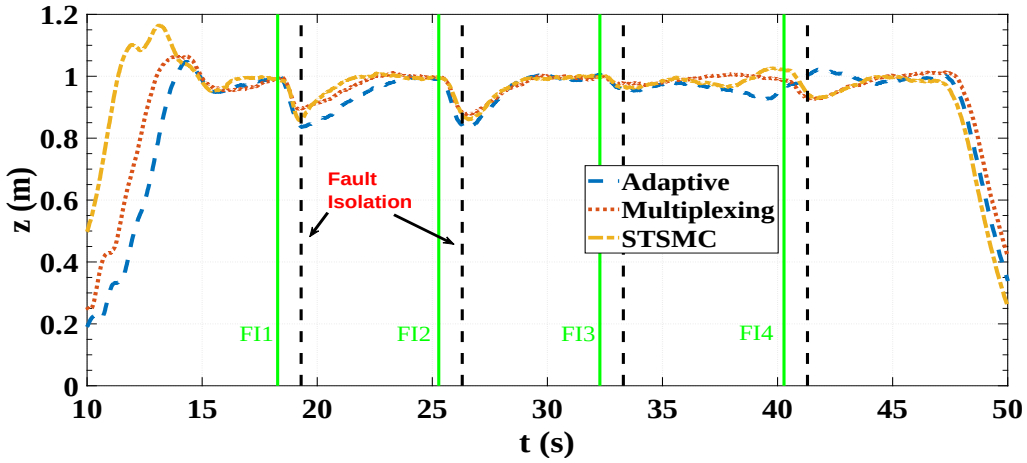


Figure 5.8 – Behavior of 10 trials means of Adaptive, STSMC, and Multiplexing methods after four successive failures injection to the top motors in hover flight

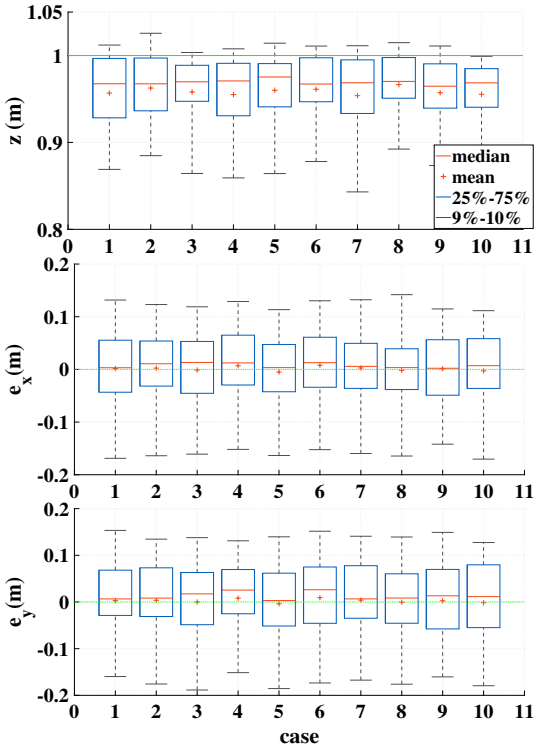


Figure 5.9 – Box and whisker plot for 10 trials of the Adaptive method after four successive failures in square trajectory flight, showing the altitude z(m), and position errors e_x(m) and e_y(m) between t=17s and t=45s.

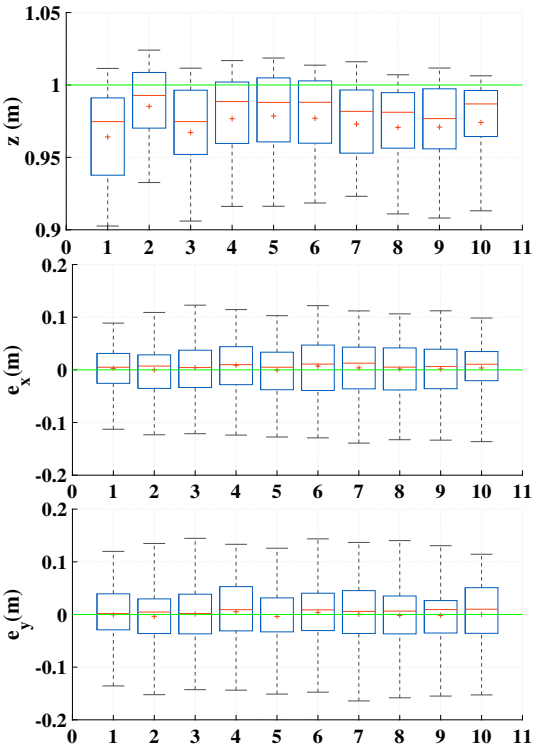


Figure 5.10 – Box and whisker plot for 10 trials of the STSMC method after four successive failures in square trajectory flight, showing the altitude z(m), and position errors e_x(m) and e_y(m) between t=17s and t=45s.

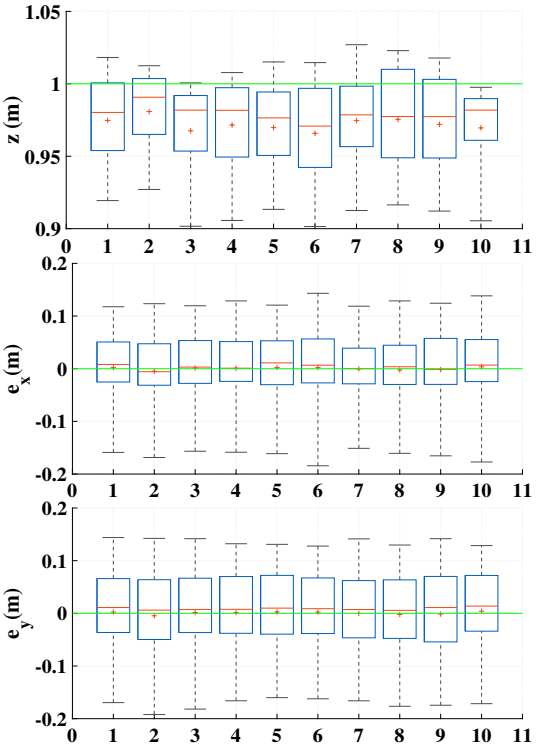


Figure 5.11 – Box and whisker plot for 10 trials of the Multiplexing method after four successive failures in square trajectory flight, showing the altitude z(m), and position errors e_x(m) and e_y(m) between t=17s and t=45s.

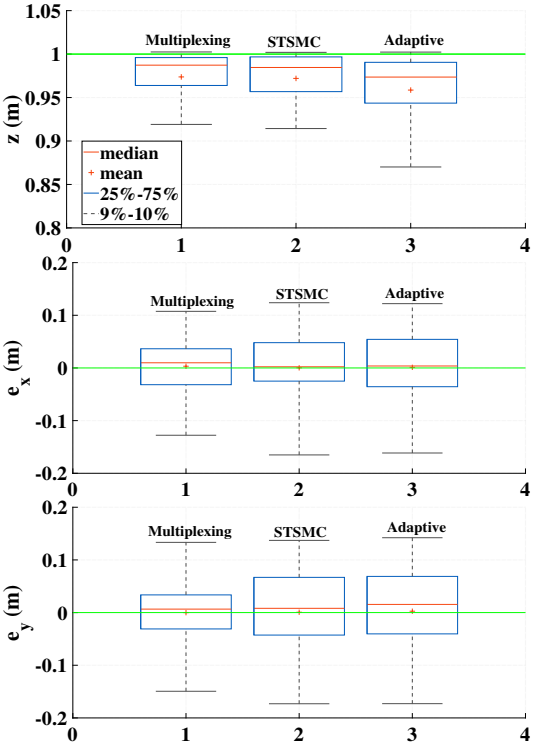


Figure 5.12 – Box and whisker plot for means of each FTC scheme trials after four successive failures in square trajectory flight, showing the altitude z(m), and position errors e_x(m) and e_y(m) between t=17s and t=45s.

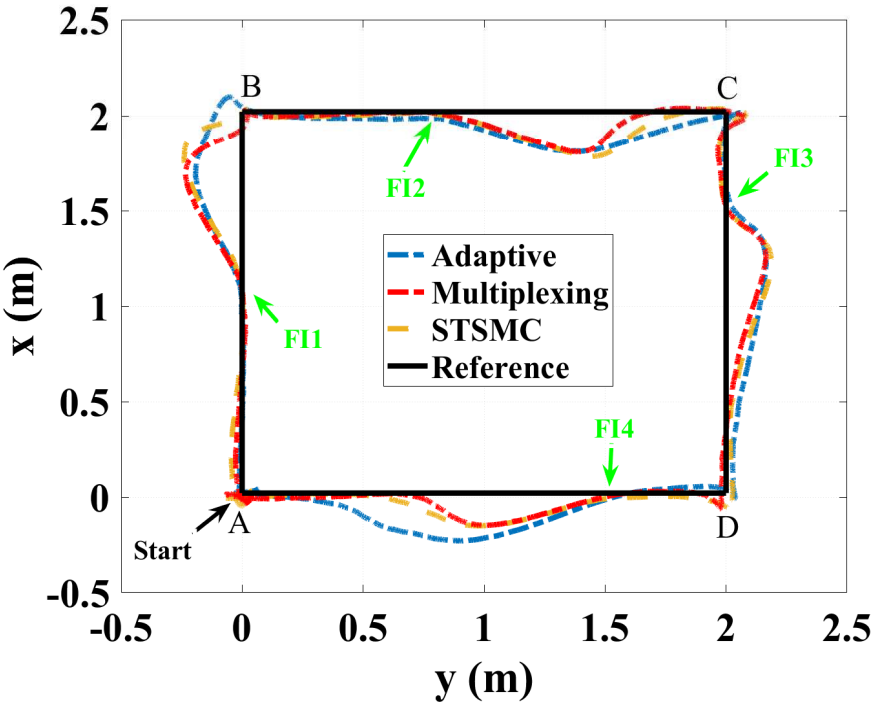


Figure 5.13 – Behavior of 10 trials means of Adaptive, STSMC, and Multiplexing methods after four successive failures injection to the top motors in square trajectory flight in ABCDA direction

Fault tolerance strategy for a quadrotor UAV under sensor and software faults

Contents

In this chapter, we present a fault-tolerance architecture for perception targeting sensors and software faults on an outdoor quadrotor UAV. This architecture extends the duplication-comparison technique described in [19] which is introduced in section 1.8.2.2, with the weighted average voting system proposed in [86]. This architecture uses data fusion with Kalman filters in order to estimate the states (position and orientation) of the UAV. Four main additions have been made between our proposed architecture and the one proposed in [19]: (1) The first difference is that we add an analytical redundancy using the dynamic model of the system (DM). (2) The second difference is that we use a weighted average voter instead of a simple thresholding, which increases the accuracy of the outputs and the error detection process. (3) The third difference is that we have multiple solutions which can be applied to recover a faulty system and this increases the flexibility of our architecture. (4) The fourth difference is that our architecture is experimentally validated in real outdoor environment using a quadrotor.

This chapter is organized as follows: we first describe the different fusion architectures. Then we present the weighted average voting system for a triple modular redundancy detailed in [86], which will also be used in our case as thresholding for error detection. After that, we describe the data fusion architecture used in the Arducopter flight stack. By briefly presenting the steps of the fusion algorithm and discussing its advantages. As the Arducopter flight stack is vulnerable to software and hardware faults, we propose our fault tolerance architecture for data fusion, which uses the voting

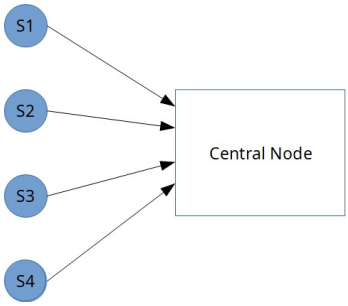


Figure 6.1 – Centralized fusion architecture

system, the redundancy based approach (Duplication/Comparison) described in [19] and an analytical redundancy using the quadrotor's equations of motions. Then, we consider a case study on the quadrotor *Tarot*650 were we show the effectiveness of the proposed strategy through simulations using mission planner and real experiments.

6.1 Fusion architectures

In a data fusion architecture, additional and complementary data are provided by different sensors to a fusion node. This fusion node provides a more meaningful information about the system states can be obtained by combining the received measurement data. Based on the availability and the processing of the measurements, the authors in [21] proposed that the fusion architectures can be divided into two categories, namely: centralized and distributed fusion architectures. However, in this thesis, we propose a third category, decentralized architecture, that the previous authors considered as part of the distributed category.

6.1.1 Centralized fusion architecture

In a centralized fusion architecture, we find a central fusion node which receives directly the raw data from multiple sensors. Then, the central fusion node computes the state estimates and makes decisions as shown in Figure 6.1. Although, a sensor module may pre-process the data before transmitting it to the central node, the term *raw data* is related to the measurements or the pre-processed data without filtering or local fusion. Each sensor provides its measurements to the central system where the data is filtered and fused together. If the data is correctly aligned and consistent, then a theoretical optimal solution to the state's estimation can be obtained using the centralized fusion architecture. However, various issues are raised in this architecture, such as the difficulty to adapt to any change in the environment, the presence of a critical point as the central node (which makes the system particularly vulnerable to failures), and a large computational load on the central node.

6.1.2 Distributed fusion architecture

In a distributed fusion architecture, local state estimates are computed by each sensor node through processing its sensor data. In most applications, the raw information are used to

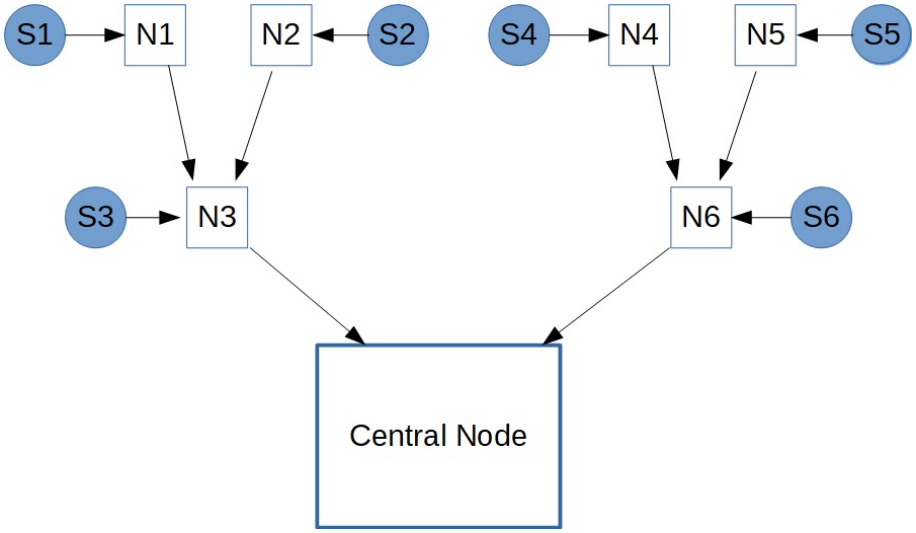


Figure 6.2 – Distributed fusion architecture.

compute the state estimates of some quantity of interest in the form of the mean or the covariance. These estimates are then communicated among sensor nodes to the central node to form a global state estimate as depicted in Figure 6.2. Compared to a centralized architecture, a distributed one solves the problems in a cooperative way. Furthermore, local processing of the data means a lower processing load on each node due to the distribution of the load, a flexibility to changes and possibly some fault tolerance if there are redundant nodes or information.

6.1.3 Decentralized fusion architecture

Another fusion architecture is the decentralized one where nodes operate independently, and share information with each other without any central fusion node. The main difference with the distributed architecture resides in the fact that the decentralized architecture lacks any central node, rather making each node computes some part of the underlying system's states and to cooperate with each other to share a more trusted view of the whole system's states.

6.1.4 Brief comparison between the fusion architectures

In general, the estimation from a centralized system is more precise than from decentralized or distributed ones. However, a decentralized or distributed system is inherently more robust and might be more fault tolerant. In our thesis, we will use this type to implement our fault tolerant fusion architecture: as all the data is collected on a single system (the UAV) most of the advantages of distributed systems would not apply.

6.2 Weighted Average Voting System

As seen in section 1.8.2.2, the fault tolerance architecture for perception proposed in [19] uses redundant sensors blocks to produce diversified fusion outputs, and a voter to detect

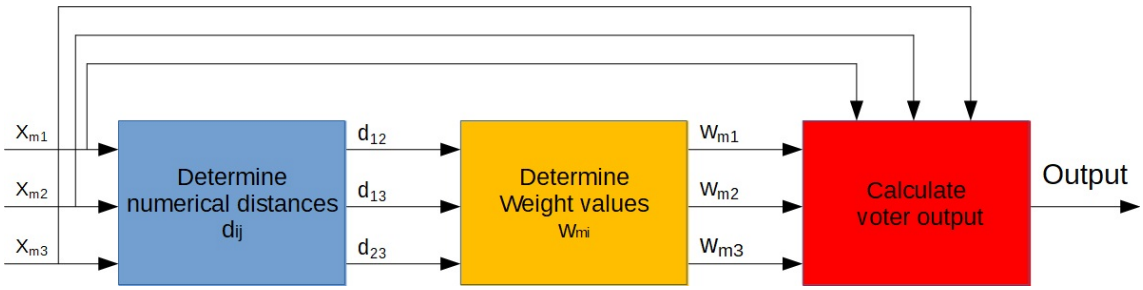


Figure 6.3 – A 3-input weighted average voter.

errors and produce a unique output. However, the voter is a fairly common threshold comparator that averages the output of the healthy sensors blocks and detects errors by a thresholding comparison. We propose to use an updated version of this voter in order to improve the system's behavior in presence of faults prior to the error detection module, by weighting the output of each sensors block with its consistency with the other's.

Considering a redundant system using multiple diversified but functionally identical modules operating in parallel, the weighted average voter [86] gives a weighted mean of values obtained from the redundant modules. Given a set of inputs from three diversified modules x_{m1}, x_{m2}, and x_{m3} for a particular cycle, the weighted average voter determines first a numerical distance of input pairs: $d_{12} = |x_{m1} - x_{m2}|$, $d_{13} = |x_{m1} - x_{m3}|$ and $d_{23} = |x_{m2} - x_{m3}|$. Out of these values, the weighting values of individual inputs, w_{m1}, w_{m2} and w_{m3} are computed: a module's result far from the other modules results would be assigned a low weight. The weight values are then used to calculate a single value as the voter output (Figure 6.3). We detail in the rest of this section how the weights and the final output are obtained.

Voter implementation

In order to determine the consistency of all voter input pairs, the weighted average voter uses the concept of the soft threshold [86]. For any voter input pairs i and j, the agreement indicator s_{ij} is defined as follows:

$$s_{ij} = \begin{cases} 1, & \text{if } d_{ij} \leqslant a \\ (\frac{n}{n-1})(1 - \frac{d_{ij}}{na}), & \text{if } a < d_{ij} \leqslant na \\ 0, & \text{if } d_{ij} \geqslant na \end{cases} \tag{6.1}$$

where d_{ij} is the distance between the input pairs i and j, a is the fixed threshold of the voter, and n is another positive tuneable thresholding parameter.

If the distance d_{ij} of input pairs is less than the threshold a, the agreement indicator is $s_{ij} = 1$. This means that the measurements of the modules i and j are aligned and consistent for the specific application. Oppositely, if the distance d_{ij} of input pairs is more than the threshold $n \times a$, then the agreement indicator becomes $s_{ij} = 0$, which means that the measurements of the modules i and j are inconsistent. For input pairs with a distance between a and $n \times a$, the agreement indicator varies in the range $[0, 1]$ as shown in Figure 6.4.

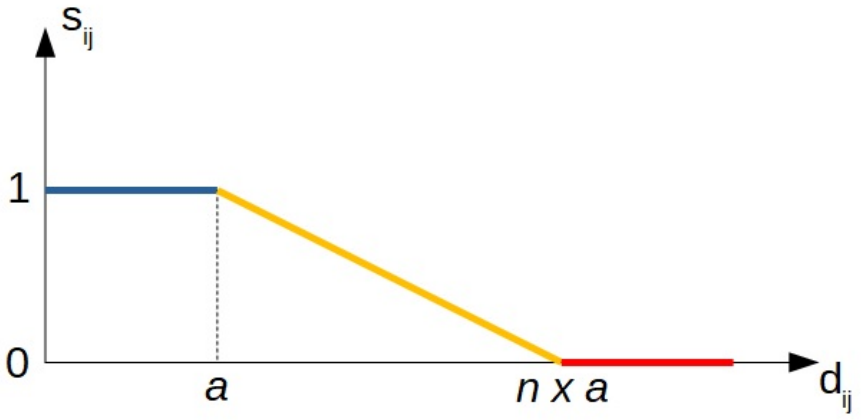


Figure 6.4 – Agreement indicator in function of the distance between input pairs.

After calculating the agreement indicator for each input pairs, the next step is to calculate the weighted values w_{mi} for each module i based on the following equation:

$$w_{mi} = \frac{\sum_{j=1, j\neq i}^{k} s_{ij}}{k-1} \tag{6.2}$$

where k is the number of diversified modules.
The voter output is calculated as follows:

$$y = \frac{\sum_{i=1}^{k} x_{mi} \times w_{mi}}{\sum_{i=1}^{k} w_{mi}} \tag{6.3}$$

The tuneable parameter n has a significant impact on the behavior of the voter. In our case, we will also use the voter as a detection error mechanism, considering that an error is present in a sensor when an agreement indicator reaches zero. Thus, as n increases, the chances of getting undetected errors increases, while when n tends to 1, the voter behaves as a common voter with a fixed hard threshold value a, where the agreement indicator s_{ij} is either 0 or 1, which increases the chances of getting false positives.

6.3 Arducopter fusion architecture

Before presenting our fault tolerant data fusion architecture in the next section, we present here how data fusion is originally done on multicopters using the Arducopter flight stack. The Arducopter flight stack uses an Extended Kalman Filter (EKF) algorithm in order to identify the vehicle's states (position, velocity and angular orientation) using the data stream from multiple sensors (IMU, GPS, compass, barometer...). Data fusion enables to achieve adequate performances and robustness despite the use of many low cost sensors, and provides the following advantages:

- Reduction of sensor noise and errors.
- Complementary sensing modules and data can be combined to achieve robust estimation. For example, the outliers of the GPS measurements are compensated using the more stable measurements of the IMU.

In multirotors running Arducopter, data fusion is used to estimate a vector containing 23 system's state's variables:

- Quaternions $(q_0, q_1, q_2, q_3, q_4)$
- Velocity (North,East,Down)
- Position (North,East,Down)
- Gyrometers bias offsets (X,Y,Z)
- Gyrometers scale factors (X,Y,Z)
- Z acceleration bias
- Earth magnetic field (North,East,Down)
- Body magnetic field (X,Y,Z)

The quaternions is an alternative representation to the Euler angles for representing the attitude of an UAV. The Euler angles (roll, pitch and yaw angles) can be derived from the quaternions. For further information about this topic please refer to [33]. The position and velocity are estimated in the earth frame defined in section 2.2.1 in Chapter 2. For more details about the remaining states, please refer to the Kalman filter section in the official site of the Ardupilot [1].

The fusion algorithm detailed here has been retro-engineered from the filter equations described in the source code of the Arducopter flight stack, more particularly from the github repository [2] of Nov 3, 2016.

Note that, the availability of faster processors such as pixhawk and Navio has enabled more complex mathematical algorithms such as the EKF3 to be implemented to estimate the orientation, velocity and position of the flight vehicle. The sequential fusion methodology in the Ardupilot's EKF can also use measurements from sensors such as an optical flow and range finders in order to assist the navigation process, which we didn't use on our UAV.

We present here a simplified non-mathematical description of the filter:

1. The IMU's angular speeds are integrated to calculate the orientation of the UAV.
2. The IMU's linear accelerations are converted using a rotation matrix from the body frame to the earth frame. In the case of altitude acceleration the gravity is also taken into account.

[1] https://ardupilot.org/dev/docs/ekf2-estimation-system.html

[2] https://github.com/priseborough/InertialNav/blob/master/derivations/RotationVectorAttitudeParameterisation/GenerateNavFilterEquations.m

3. The IMU's converted linear accelerations from step 2 are integrated to calculate the velocity.

4. The velocity obtained from the IMU during step 3 is integrated to calculate the position.

   The process (1 to 4) is referred as the State Prediction process in the Kalman filter. The filter has other inputs (gyro and compass biases...) that change at a slower rate, so they are not taken into account in this process but are used during the correction phase in the next steps.

   If we had a perfect initial and perfect IMU measurement, the filter could stop here and keep repeating steps 1 to 4. However in practice, disturbances such as sensor noises would lead the system to an unstable diverging state. For this reason, the EKF algorithm uses data from other sensors (GPS, optical flow, barometer, etc.) in order to correct the positions predicted by step 4 and to restabilize the system. In the steps 6 to 7 the example of a GPS is given for correction of the horizontal position, however the same principle applies to other measurement types (barometric altitude, GPS velocity, etc)

5. The estimated gyro and accelerometer noises are calculated from the steps 1 to 4 and are then used to estimate the growth in error of the calculated position and orientation. These estimated errors are found in the State Covariance Matrix (P in Figure 1.7).

6. When a GPS measurement is available, the filter calculates the difference between the predicted position from step 4 and the position from the GPS. This difference is called an Innovation ($\hat{S}_k$ in equation (1.10)).

7. The Innovation from step 6, the State Covariance Matrix from step 5, and the GPS measurement error specified by its variances (information given in the specific sensor and usually given by the manufacturer) are combined to calculate a correction to the state's variables obtained from step 1 to step 4. This is referred as a State Correction.

8. Since the correction has been made, the obtained values of the state's variables are more precise. The filter updates the states and the state covariances and returns to step 1.

An illustration of the process is introduced in the Figure 6.5. As shown in this figure, the EKF is a centralized fusion architecture, since the state estimates are computed in a central fusion node which receives the raw data from all the onboard sensors. In order to increase the robustness in case of multiple IMUs, the EKF performs a sanity check on the raw data of the IMUs, where the health of each IMU is checked based on the estimated covariance values of each sensors, then the IMU reporting the best measurement is selected as the primary. It is thus used in steps 1 to 4 for state prediction and data fusion as explained in the algorithm.

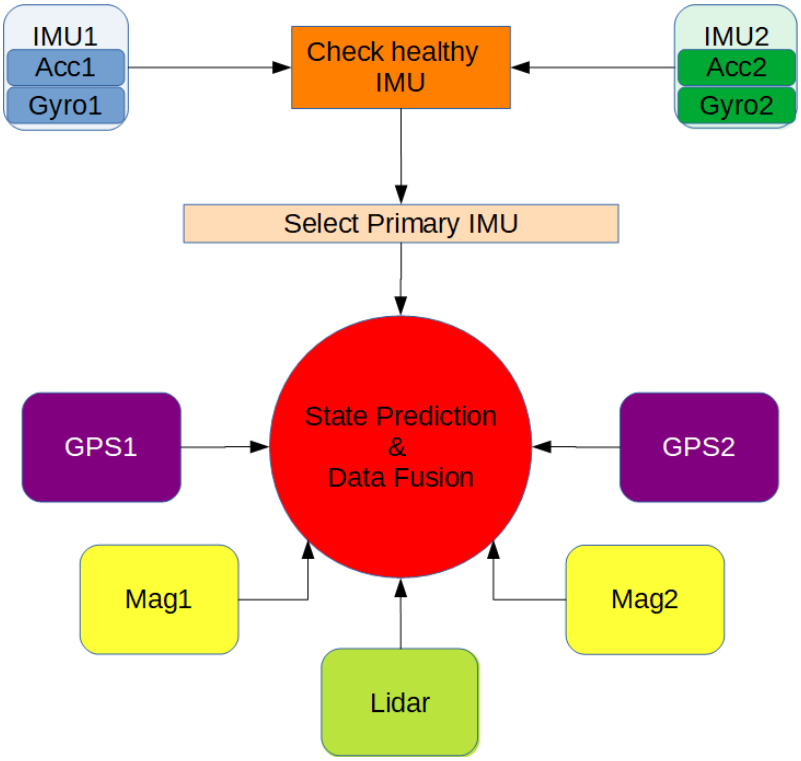


Figure 6.5 – Arducopter data fusion architecture.

6.4 Enhanced data fusion architecture for tolerating sensor and software faults

Data fusion can be used to reduce sensor noise and errors and to achieve and optimize a solution in case of multiple sensors. However, even though data fusion architectures are robust to some temporary outliers (like sensor noises) they can not usually tolerate prolonged sensor faults. They are also vulnerable to software faults, particularly on gains or covariances in the data fusion mechanism.

In this section, we present our proposed data fusion architecture for tolerating sensors and software faults which is based on duplication and comparison and was initially proposed in [19]. We also use a redundant predictive block based on a mathematical model of the UAV for both error detection and diagnosis and the weighted average voting system presented in section 6.2. This architecture will be validated in section 6.7 on a real outdoor UAV: the *Tarot*650 quadrotor.

The proposed fault tolerance architecture is shown in Figure 6.6. This architecture provides detection and recovery services adapted to multi-sensor perception systems to ensure their reliability and the safety of the UAV.

The main differences between our architecture (Figure 6.6) and the architecture proposed in [19] are summarized as follows:

- *Branches*: in our architecture we proposed to add an analytical redundancy using the dynamic model (DM) and the equations of motion of the UAV, whereas in [19], all the branches are formed only using available hardware (sensors).

- *Voter*: in our architecture we used an average weighted voter to calculate the output of the system, whereas in [19], a simple thresholding is used instead.
- *Recovery*: in our architecture, we propose several solutions to recover a faulty system by altering the outputs of the sensor blocks, whereas in [19], the solutions were limited to using the identified healthy branch.
- *Validation*: our architecture is validated in real outdoor flights, whereas the architecture in [19] was only validated in an open loop using data sets from real experiments.

In our generic architecture, we implement two parallel and independent perception branches, each containing a data fusion block (here KF1 and KF2) that estimates the state of the UAV from redundant and diversified sensor blocks. Each sensor block (SB1, SB2, SB3 and SB4) may contain one or several sensors. In Figure 6.6, we consider that the sensor blocks SB1 and SB3 contain functionally similar sensors measuring the same state's variables. Similarly, the sensor blocks SB2 and SB4 are functionally equivalent. Note that more than two independent branches could be used if the required resources and space are available, which ultimately would allow to tolerate more successive faults. In the residuals generator block we calculate the residuals from the redundant sensors data which represents the differences between the output of these redundant sensors. Moreover, the dynamic model module in Figure 6.6 represents the analytic redundancy in our architecture, where we use the equations of motion of the UAV and the control input (thrust and torques) of the motors. The output of all the mentioned modules are then fed into the final module which is called the software/hardware error detection, isolation and recovery module.

This architecture can tolerate one or more successive hardware faults related to sensor blocks (as long as we still have redundancies in functionally equivalent sensors) and allows to tolerate a software fault related to the fusion blocks, under the assumption of no simultaneous sensor's failure.

6.5 Fault detection

We propose in this section an implementation example of our architecture with three parallel branches, two of them executing a data fusion process and using each two sensor blocks similarly to Figure 6.6. The first branch (SB1, SB2, KF1) combines the outputs of sensor blocks SB1 and SB2 within the KF1 fusion block, the second branch (SB3, SB4, KF2) combines the outputs of sensor blocks SB3 and SB4 at the KF2 fusion block, and the third branch is the Dynamic model which uses the equations of motion of the quadrotor UAV. For the sensor blocks (SB1, SB2) and (SB3, SB4), the outputs of each component are compared with its redundant component: the outputs of SB1 are compared with the output of SB3, while the outputs of SB2 are compared with the output of SB4. Also, the outputs of the fusion blocks KF1 and KF2 and of the Dynamic model (DM) are compared and combined using the weighted average voter. The comparison between the fusion blocks allows to detect an error in the system, while the comparisons between the sensor blocks are used to diagnose the detected error. In case of no fault in the system, the

output of the perception system is the same as the output of the voter which represents a combined solution of the three estimations of the state vector. In the following we explain the error detection and identification algorithm. Note that a third sensor branch could be used instead (or additionally) to the Dynamic Model and could allow to tolerate more hardware faults, but would obviously be more costly and limit the payload of the UAV.

Our architecture is made up of three modules (as shown in Figure 6.7). In the following, we define each module the two modules used respectively for error detection and identification:

1. *Voter and error detection*: The error detection is done by computing the values of the agreement indicators s_{ij} of the outputs of the two fusion blocks KF1 and KF2, and of the Dynamic model (DM). The indicators are computed using the equation (6.1), where the indices 1, 2 and 3 represent respectively the outputs of KF1, KF2 and DM. When the indicator s_{12} goes to zero, this implies the occurrence of an error in the system. This error is either due to a software fault of the data fusion blocks KF1 and KF2, or to a hardware fault in one of the sensor blocks. Using these agreement indicators, we can also identify the erroneous branch thanks to the redundancy introduced by the DM. When $s_{12} = 0$, we compare the values of s_{13} and s_{23}. If $s_{13} = min(s_{13}, s_{23})$, and $|s_{13} - s_{23}| > \varepsilon$ (a threshold value delta), then the error is in the first branch. If $s_{23} = min(s_{13}, s_{23})$ and $|s_{13} - s_{23}| > \varepsilon$, the error is in the second branch. Finally, if $|s_{13} - s_{23}| \leqslant \varepsilon$, we can only conclude that the fault does not yet affect the system sufficiently to be identified, and we will continue checking this condition in the next cycles until we identify the erroneous branch.

   In case where the indicators s_{13} and s_{23} go to zero with $s_{12} = 1$, this implies that there is an error in the estimation of the DM. Such error can be due to high uncertainties in the system, which can happen because of unusual environmental conditions or divergence due the model imperfections. In this case, on possible solution is to reset the output of the DM to the average value of the outputs of KF1 and KF2, under the assumption that no other fault impacts the sensors. We could also remove the erroneous DM from the system, but we would then only be able to detect hardware faults, and no longer tolerate them as we would be unable to identify the correct branch.

2. *Software/hardware error identification module*: This module diagnoses the detected error, and identifies whether it is a software fault in the data fusion process or a hardware fault of a sensor block. In practice, the comparison of sensors outputs allow us to determine whether the detected error is hardware or software, and to identify the erroneous sensor in the case of a hardware fault. The residuals are computed as the differences between the outputs of the redundant sensors blocks (SB1/SB3 and SB2/SB4 in our implementation). If the output of a sensor block deviates significantly from its redundant block then the system diagnoses a hardware error on one of these two sensors. More precisely, a hardware error is detected if the value of the residual $\Delta_{SB1,SB3}$ or $\Delta_{SB2,SB4}$ exceeds the values of the fixed thresholds Th_{13} and Th_{24}, which are depending on the application. If the value of all residuals are below their corresponding threshold, a software error is diagnosed either in KF1 or KF2. Note that if these thresholds were too high or too

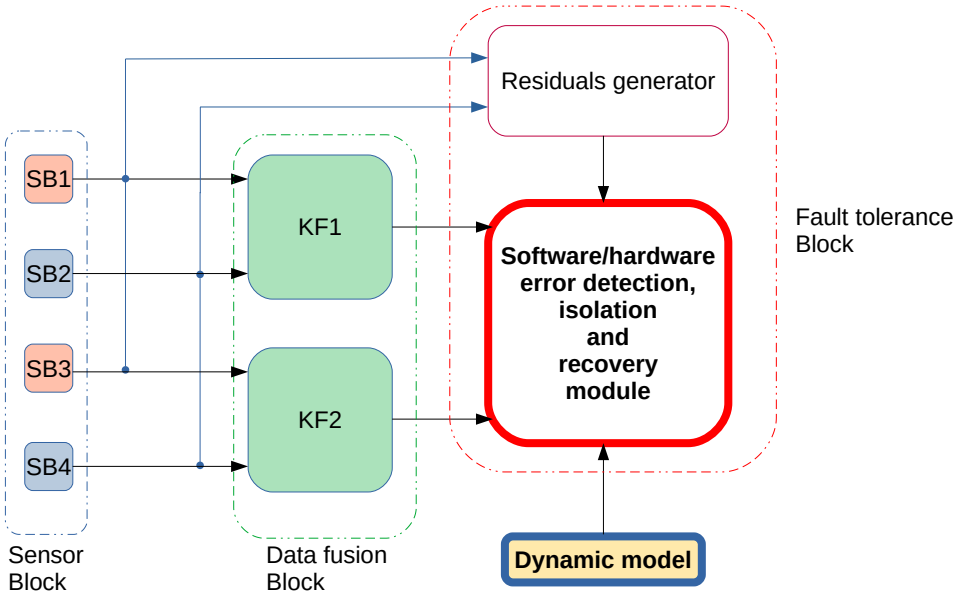


Figure 6.6 – Fault tolerance architecture using Duplication comparison and analytical redundancy

low, this may cause respectively absences of error detection or false positives. The value of these thresholds are determined by taking into account the precision of the onboard sensors. For example, for the low cost GPS l used in our *Tarot*650, a value of 2 meters is acceptable. On the other hand, if the value of the residuals are below the predefined thresholds, we diagnose a software error in one of the data fusion blocks KF1 and KF2.

The operation of the fault tolerance algorithm is summarized in the algorithm 1 and illustrated in Figures 6.8 and 6.9

6.6 Recovery module

Once the error in the system is detected on the voter level and isolated by the identification module, an appropriate solution must be applied accordingly in order to re-stabilize the perception system in case of hardware or software faults. In the following, some of the possible solutions are presented.

6.6.1 Recovery for Hardware Fault

In the case of a hardware fault in component i of KF1, many solutions may be implemented in order to reconfigure the system after the error detection and identification

Algorithm 1: Fault detection and identification algorithm

Data: $KF1, KF2$: data fusion output
$SB1, SB2, SB3, SB4$: sensor blocks outputs
Constant: $\varepsilon, Th_{24}, Th_{13}$

```
1  begin
2      Calculate s12, s13, s23;
3      Calculate Δ_SB1,SB3 and Δ_SB2,SB4;
4      if s12 = 0 then
5          if |s13 − s23| > ε then
6              if Δ_SB1,SB3 > Th13 then
7                  if s13 = min(s13, s23) then
8                      /* error in SB1, apply adapted hardware recovery mechanism
                          */
9                  else
10                     /*error in SB3, apply adapted hardware recovery mechanism
                          */
11             else if Δ_SB2,SB4 > Th24 then
12                 if |s13 − s23| > ε then
13                     if s13 = min(s13, s23) then
14                         /* error in SB2, apply adapted hardware recovery
                              mechanism */
15                     else
16                         /*error in SB4, apply adapted hardware recovery
                              mechanism */
17             else if s13 = min(s13, s23) then
18                 /* error in KF1, apply adapted software recovery mechanism */
19             else
20                 /* error in KF2, apply adapted software recovery mechanism */
21         else
22             /* cannot identify the faulty branch, keep running without
                  modifications */
23     else
24         /* No error detection, system healthy */
```

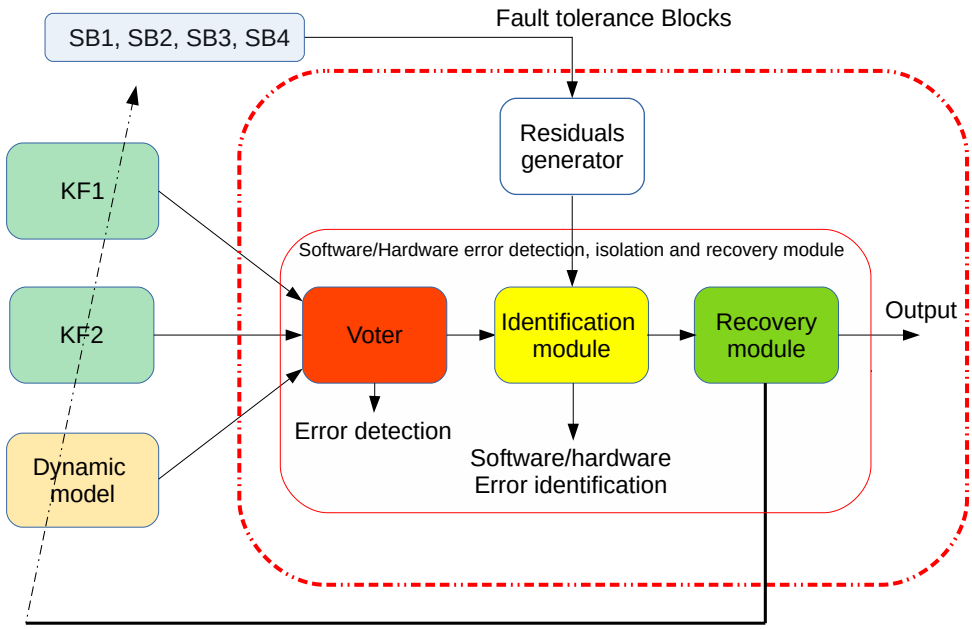


Figure 6.7 – Software/hardware error detection, isolation and recovery module

process. However, we identify two main solutions as follows:

- **Sol 1**: the first solution consists in removing the component i from KF1 and replacing it with the equivalent component j from KF2. The newly defined block KF3 (KF1 with component j) is then integrated into the voting mechanism along with KF2 and DM. Henceforth, using this technique, it is possible to tolerate all software faults and hardware faults except on component j which cannot be even detected anymore. This is because if the component j become faulty, it will affect KF2 and KF3 simultaneously, and by following the diagnostic logic of 1, we can see that a fault in DM will be detected because it's equivalent output will defer from the coherent estimations of KF2 and KF3 and a false alarm will be reported by the error detection process.

- **Sol 2**: the second solution consists in removing all the branch containing KF1 from the architecture and using only the remaining healthy branches KF2 and DM. Moreover, the healthy components of KF1 as backup resources and keep comparing their output with those of KF2. Henceforth, it is possible to detect all possible faults although we can only tolerate the errors on the healthy components only (all components except component j).

- **Sol 3**: the third solution consists in removing all the branch containing KF1 and replacing it by a virtual estimation branch KF3 where the output of KF3 is the average of the outputs of KF2 and DM. This may increase the precision of

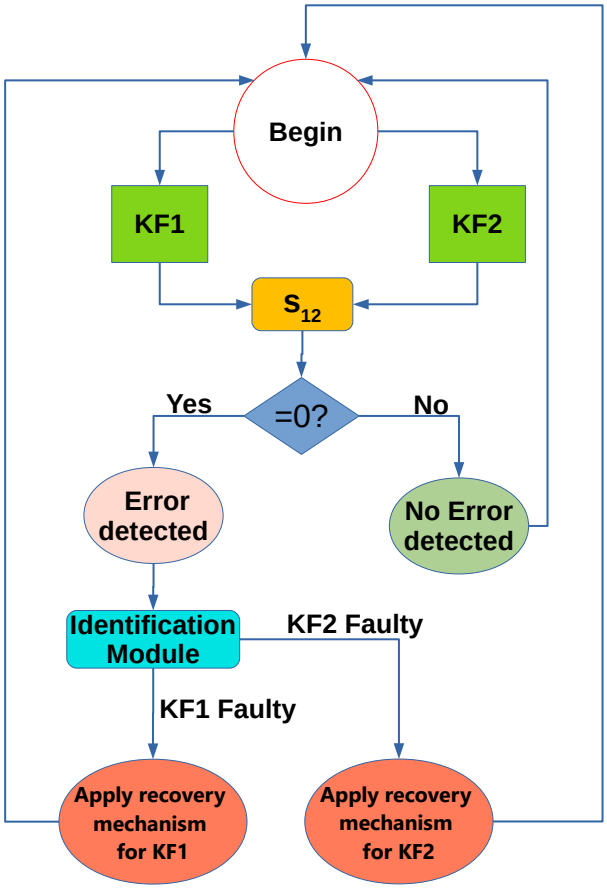


Figure 6.8 – Proposed data fusion architecture for tolerating sensors and software faults

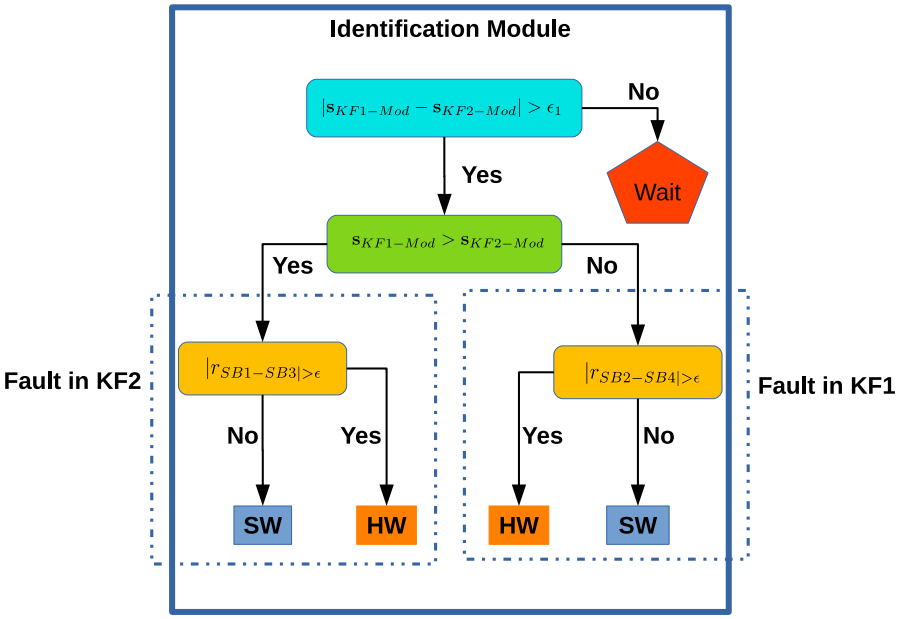


Figure 6.9 – Proposed data fusion architecture for tolerating sensors and software faults

the estimation in case of increasing uncertainties in the DM calculations due to unmodeled effects. The same properties of sol 2 are available here, however the detection time become greater in this case after a new fault in the system.

Note that if the hardware faults is in KF2, the same solutions mentioned above can be applied by replacing KF1 by KF2 and vice-versa. Once the appropriate solution for the intended application is selected, the recovery module could be easily designed accordingly.

6.6.2 Recovery for Software Fault

In the case of software fault in KF1, again many solutions may be proposed in order to reconfigure the system after the error detection and identification process. However, we identify two main solutions as follows:

- **Sol 1**: the first solution consists in initializing a new data fusion block KF3 with the current states values of the healthy block KF2. Also, the same code section of the KF2 is used in order to perform the estimations in KF3, since no software fault is yet activated in KF2. Henceforth, using this technique, it is possible to detect and tolerate all hardware faults in the system although software faults cannot be guaranteed to be detected anymore. This is because a new software fault may affect the shared code section between KF2 and KF3 and thus leads to a simultaneously distortion in both estimations, and by following the diagnostic logic of 1, we can see that a fault in DM will be detected because it's output will defer from the coherent estimations of KF2 and KF3 and a false alarm will be reported by the error detection module.

- **Sol 2**: the second solution consists in removing all the branch containing KF1 from the architecture and using only the remaining healthy branches KF2 and DM.

Moreover, the healthy components of KF1 can be kept as backup resources to keep comparing their output with those of KF2. Henceforth, it is possible to detect and tolerate all the harware faults in the system, however we can only detect software faults without the ability to tolerate them. It is also possible to identify a virtual estimation branch KF3 by using the average of the outputs of KF2 and DM as the output of KF3. This may increase the precision of the estimation in case of increasing uncertainties in the DM calculations due to unmodeled effects.

- **Sol 3**: the third solution consists in initializing from the beginning a backup data fusion process in parallel. The added data fusion core in the system is only performing mathematical calculations without being integrated into the architecture when running normally. Once a software fault in KF1 is detected, the data fusion core in the memory is replaced by the backup core using the same components. Using this solution, all the detecting and tolerance capacities in the system are retained after the first software fault.

Again once the appropriate solution for the intended application is selected, the recovery module could be easily designed accordingly.

6.7 Validation

To evaluate the proposed architecture for tolerating sensors and software faults developed during this work, we have conducted the following experiments:

- A real outdoor flight test with hardware fault injection (additive fault of 5m in the x and y direction) on the GPS1 during a U-path trajectory tracking
- A real outdoor flight test with hardware fault injection (additive fault of -1m) on the Lidar used in KF1 during a hovering at 3m altitude
- A simulation of a flight with software altitude fault injection in KF1 during a hovering at 3m altitude
- A simulation of a flight with software x, y fault injection in KF1 during a U-path trajectory tracking

The real experiments were conducted using the Tarot 650 which is a commercial hobby type quadrotor UAV. The implemented UAV using this frame is shown in Figure 6.10. The Tarot 650 is equipped with the Hex Cube Black (FMUv3). This is an updated version of the Pixhawk controller which is an open-source and open-hardware autopilot able to run the Arducopter flight stack used to build our software. This Cube has a 32-bit ARM Cortex M4 core processor with FPU with 168 Mhz/256 KB RAM/2 MB Flash and a 32-bit failsafe co-processor. It also includes three redundant IMUs and two redundant barometers. The Cube has been equipped with additional sensors, namely the Lidar Lite v3, two GNSS modules and an RTK module used as the reference for the position measurements to evaluate our experimental results.

Figure 6.10 – Experimental Tarot 650 quadrotor

To our best knowledge, the model parameters of the TAROT 650 were not identified elsewhere in the literature, and we couldn't obtain these information from the manufacturers. Therefore, we had to estimate these constants using mesurement tool (numerical balance for mass, and a meter for length) and analytical expressions for inertia values. The Tarot 650 parameters are given in Table 6.1.

m	Mass of the vehicle	$1.7\,kg$
l	Length of the arm	$0.23\,m$
I_{xx}, I_{yy}	Inertia	$3.38 * 10^{-2}\,Kg.m^2$
I_{zz}	Inertia	$2.25 * 10^{-2}\,Kg.m^2$

Table 6.1 – The TAROT 650 model's parameters

All the simulations and experiment configurations were done using a ground control station (Mission Planner) running the Arducopter flight stack. Note that the control law used to control the UAV is a smooth second order sliding mode control law based on the super-twisting algorithm proposed in [64]. It has been proven that this control law is capable of stabilizing the UAV despite the existence of external perturbations due to wind which increases the reliability of the sates estimations of the dynamic model.

6.7.1 Implementation of the fault tolerance architecture

The details of the implemented architecture are shown in Figure 6.11. In this application we implement three parallel branches (B1, B2 and B3).

The first two branches B1 and B2 are the fusion blocks, each containing two sensors blocks: SB1 and SB2 in B1, and SB3 and SB4 in B2. The first sensors blocks SB1 and SB3 are the IMU blocks (IMU1 and IMU2), they are used to estimate the prediction of the state vector. Each IMU contains the following sensors:

- A gyroscope (Gyro1 and Gyro2): it is used to measure the angular velocity of the UAV in the body frame. The integration of this measure gives the attitude angles (roll, pitch and yaw) of the UAV.

- An accelerometer (Acc1 and Acc2): it is used to measure the linear accelerations of the UAV. The integration of this measure gives the linear velocity, and its double integration gives the position of the UAV.

Each of the other two blocks SB2 and SB4 contains the following sensors:

- A GPS (GPS1 and GPS2): it is used to measure the absolute position of the UAV.

- A magnometer (Mag1 and Mag2): it is used to measure the heading (yaw) of the UAV.

- A lidar (Lidar1): it is used to measure the altitude of the UAV.

Note that we only use one lidar in our experimental quadrotor. Of course we would need another redundant lidar to tolerate faults on this sensor according to our architecture, but we did not have the the second one on the UAV at the time of the experiments. In fact, we intended first to use a barometer as a diversified sensor to the lidar, but its performances were too poor in practice to use it.

A data fusion block combines the outputs of these sensors blocks in a Kalman filter (KF1 for SB1 and SB2, and KF2 for SB3 and SB4). The sensors blocks SB2 and SB4 are used in the correction step of the Kalman filter process, while the sensors blocks SB1 and SB3 are used for the prediction step of the Kalman filter process. The third branch B3 is the Dynamic Model block. In this block the state vector of the UAV is estimated using the dynamic model of the vehicle and the relationships between the PWM inputs and the generated thrust and torques that we identified on our UAV as follows:

$$F_i = (-1.4736u_{pwm}^3 + 11.0691u_{pwm}^2 - 16.7074u_{pwm} + 7.3007)/100 \tag{6.4}$$

$$\tau_i = (-0.0905u_{pwm}^3 + 0.4771u_{pwm}^2 - 0.679u_{pwm} + 0.3045)/100 \tag{6.5}$$

The dynamic model that we use in this chapter is the model from (2.24), since the experimental validations are done in an outdoor environment where wind perturbations affect the UAV.

Finally, in order to test our architecture in real outdoor experiments, we need to choose all the values of the thresholds used in the Residuals Generator block and the minimum and maximum threshold's parameters used in the voter module. The values that we used were determined after conducting several real outdoor experiments in normal conditions and are chosen empirically in order to prevent the occurrence of false alarms and undetected errors. They are given in Tables 6.2 and 6.3. For example, in case of hardware fault in one of the GPS modules, the residuals between the measured distances d_1 and d_2 measured respectively by GPS1 and GPS2 will exceed the fixed threshold TH_{23} which can be found equal to 2 m from Table 6.3.

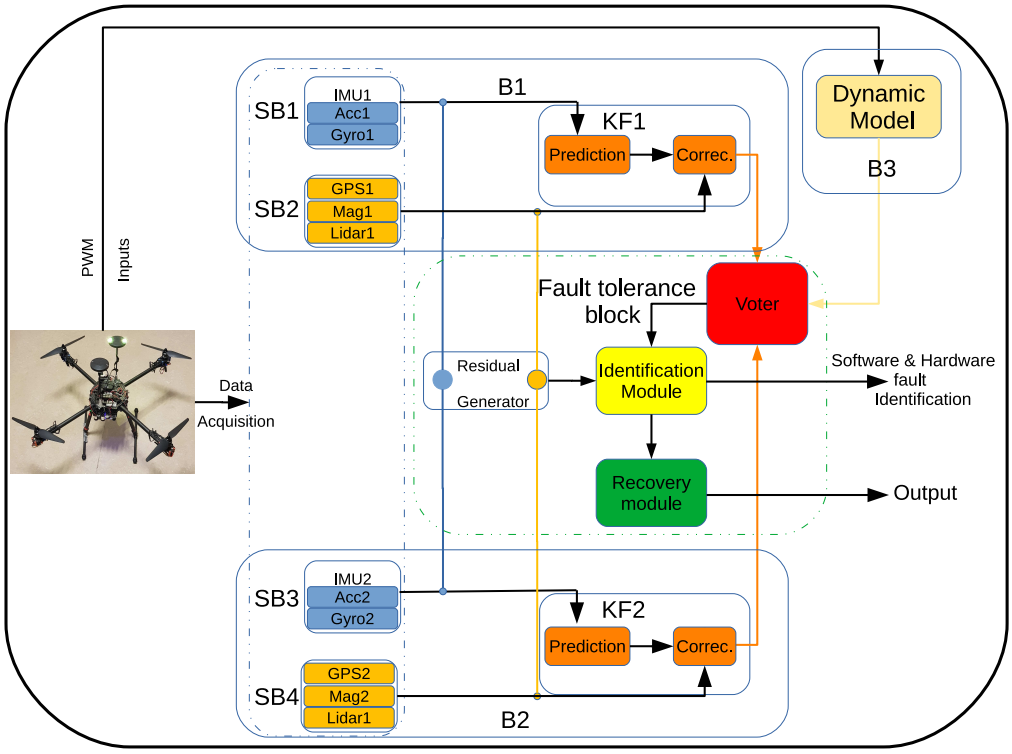


Figure 6.11 – Proposed data fusion architecture for tolerating sensors and software faults

Estimated State	a	n
x	0.5	6
y	0.5	6
z	0.5	2
ϕ	1	3
θ	1	3
ψ	2	3

Table 6.2 – Tuneable parameters values for selecting the thresholds of the voter

Sensor	Measured state	Threshold	Unit
GPS	d	2	m
Magnetometer	ψ	6	deg
Lidar	z	1	m
IMU	$\ddot{x}, \ddot{y}$	0.01	m/s^2
Gyro	$\dot{\phi}, \dot{\theta}$	0.1	deg/s

Table 6.3 – Thresholds values used in the Residual Generator block

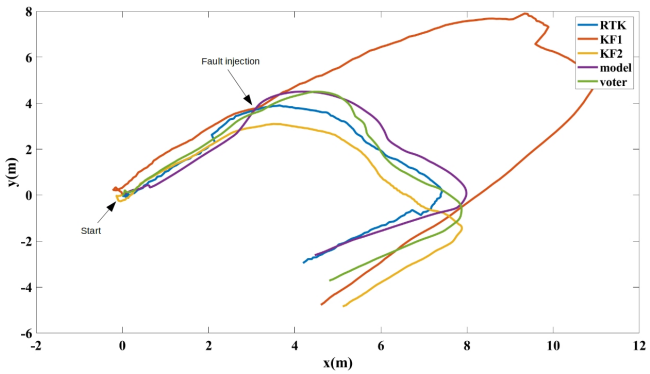


Figure 6.12 – GPS additive fault: positions estimated by the two Kalman filters, the dynamic model, the voter, and the ground truth given by a GPS RTK

6.7.2 Additive fault on GPS1

In this experiment, an additive fault of 5m in the x and y direction has been injected in the first GPS (GPS1). The UAV is required to follow a U-path trajectory starting from the initial point $(x = 0, y = 0)$ to the destination point $(x = 4, y = -3)$ as shown in Figure 6.12. This is the case of a trajectory tracking case, where the desired positions and velocities are time-dependent and the system is restricted to follow the U-path reference. Note that the GPS RTK is chosen as ground truth because of its precision (1cm) when receiving enough satellites data (four or more in our case).

The injected fault simulates an external fault such as a jump in the position provided by the GPS due to bounces from one of the satellite signals. This fault is generally not permanent, but it can occur for a significant period of time. Note that using the same type of GPS would usually not tolerate this fault, as it has a common cause. However, using diversified GPSs (such as Galileo, Glonass, or USA's GPS) would allow to tolerate faults due to a lack of satellites visibility in the constellation or signal rebounds. However some common cause faults, such as signal obtrusions due to a tunnel or a dense forest, would still be impossible to tolerate.

Between the instants $t_{inj} = 7s$ and $t_{inj_{end}} = 15.8s$, we added a 5 meter jump on the x_{GPS1} and y_{GPS1} components, as described in (6.6).

$$\begin{aligned} &x_{GPS1}(t_k) = x_{GPS1}(t_{inj}) + 5 \\ &y_{GPS1}(t_k) = y_{GPS1}(t_{inj}) + 5 \\ &\text{for } t_k \text{ such as } t_{inj_{end}} \geq t_k \geq t_{inj} \end{aligned} \tag{6.6}$$

Figure 6.12 shows the UAV positions given by the different localization systems during the first experiments: the ground truth as the output of the GPS RTK, the output of the branches B1 and B2 as the Kalman filter blocks KF1 and KF2, the output of the branch B3 given by the dynamic model, and the output of the perception component as the result of the voter on the three branches. The additive fault injected to the system causes the position estimated by KF1 to deviates significantly from all the other systems.

Figure 6.13 presents the agreement indicators of our voter during the experiment. We can see that after the fault injection at $t = 7s$ the agreement indicator between the two

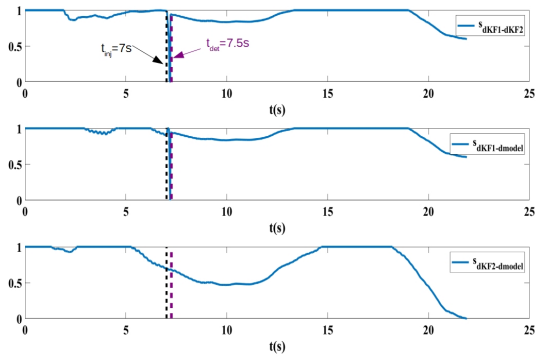


Figure 6.13 – GPS additive fault: the agreement indicators of the Euclidean distance of the first and second Kalman filters and the model

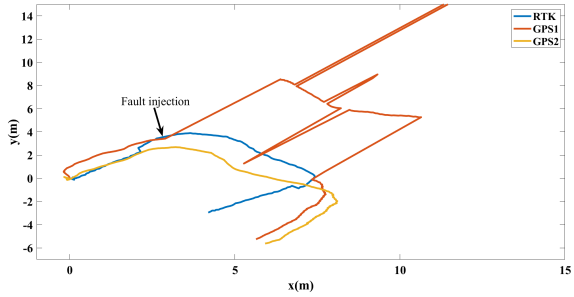


Figure 6.14 – GPS additive fault: positions estimated by the two GPS, the dynamic model, the rtk and the voter

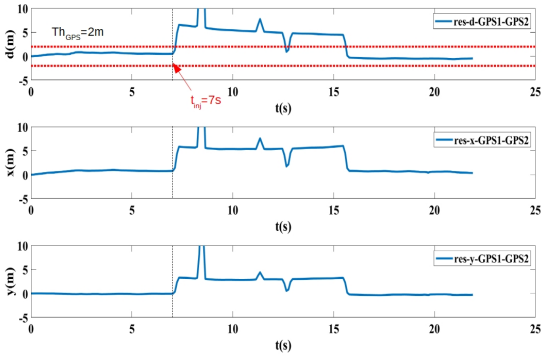


Figure 6.15 – GPS additive fault: The residues of the x, y, d components between the first GPS and the second GPS

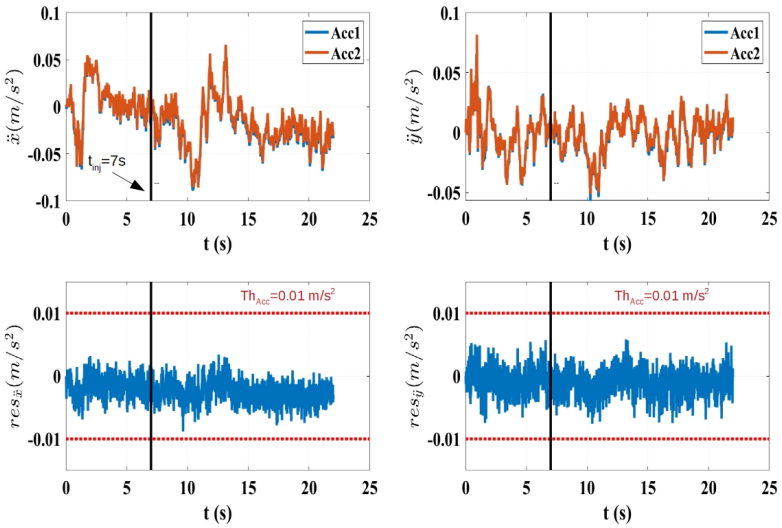


Figure 6.16 – GPS additive fault: accelerations estimates by the two Accelerometers and their corresponding residues

Kalman filter blocks $s_{dKF1-dKF2}$ starts to fall down from $s_{dKF1-dKF2} = 1$ and reaches $s_{dKF1-dKF2} = 0$ after 0.5s at $t_{det} = 7.5s$. A similar behavior can be seen for the agreement indicator between KF1 and the dynamic model. The agreement indicator between KF2 and the model stays much higher during this period, although it drops from 1 to 0.5. This is due to the value of KF1 affecting the output of the voter until the error detection, and thus affecting the dynamic model that uses the voter's output to calculate its next position.

Following Algorithm 1, when the agreement indicator $s_{dKF1-dKF2}$ equals 0, we check which one is the more consistent to the dynamic model. Figure 6.13 clearly shows from the agreement indicators pertaining to the dynamic model that the erroneous one is KF1. Thus, the erroneous branch is B1 which contains KF1.

After identifying the erroneous branch, the next step is to identify if the error is due to a hardware or software fault. Thus we compare the outputs of the functionaly equivalent sensors. Figure 6.14 shows the measured positions by GPS1 and GPS2, and for reference by the ground truth. As shown in this figure, the positions of GPS1 and GPS2 are highly different after the fault injection. Indeed, in Figure 6.15, we can see the residuals of the measured x, y, d components by the two GPS. In particular, the distance d between the two GPS exceeds the $TH_{GPS} = 2m$ threshold at $t_{det} = 7.5s$, thus a hardware fault is reported at $t_{det} = 7.5s$. Note that after the fault injection on GPS1, some outliers appears in the measurements of this sensor. In our opinion, we think that this is due to some corrections which are done automatically in the Ardupilot since the autopilot is reporting unusual output of the sensor which is not supposed to be unhealthy.

Since the detected error in our architecture involves the positions estimates, the outputs of the Acc1 and Acc2 are also compared since they are also used to predict the positions estimates in the Kalman Filter and could be another source of the error. Figure 6.16 shows the measured accelerations $\ddot{x}$ and $\ddot{y}$ measured by Acc1 and Acc2 and their corresponding residues. It is clear that the values of the residues do not exceed the value of the predefined threshold $Th_{Acc} = 0.01m/s^2$, thus the two Acc are reported as healthy

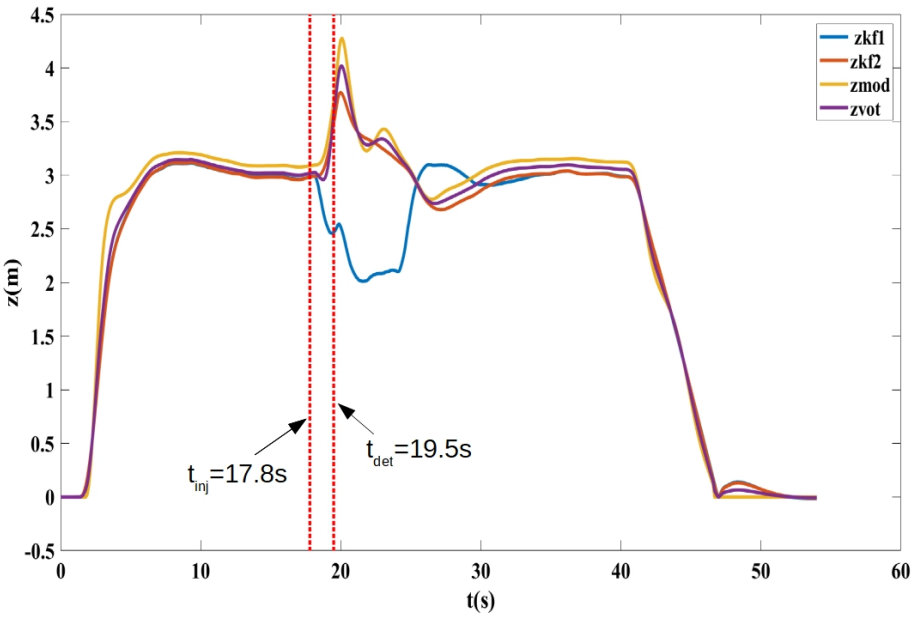


Figure 6.17 – Lidar additive fault: The estimated altitudes by the two Kalman filters, the dynamic model and the voter

sensors. Since we have already identified the first branch as the erroneous branch in the previous step, we can conclude that the erroneous sensor is the first GPS (GPS1) in KF1.

Once the GPS1 is detected as the faulty sensor, the first branch is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by KF3 is equal to the average of the position estimated by KF2 and the Dynamic model. This recovery corresponds to the solution 3 in section 6.6.1. This explains why the agreement indicators $s_{dKF1-dKF2}$ and $s_{dKF1-dmodel}$ increases quickly after the identification of the fault in Figure 6.13, since KF1 has been recovered. Note that as we do not directly use the redundant sensors of SB1 and SB3, we can no longer directly tolerate another fault. However, we can still compare the outputs of the IMU, the lidars and the magnometers to detect hardware faults. We could also have used the other recovery mechanism proposed in 6.6.1, by using GPS2 instead of GPS1 in KF1.

6.7.3 Additive fault on Lidar1

In this experiment, the quadrotor is required to perform a hovering flight at a 3 meters altitude. In this experiment, the injected fault consists in adding a value of -1 meter to the Lidar1 output used in KF1. Note that because of time constraints, as previously stated in section 6.7.1, we only have one lidar on our UAV, which outputs are sent to both SB2 and SB4. To simulate a fault on a single Lidar, we thus only inject the fault on the data received by SB2. There is no need for an additional ground truth in this experiment since the precision of the Lidar Lite v3 equipped to the drone is below 15 cm, thus we consider it as a ground truth. The estimated altitudes by the first and second Kalman filters (KF1 and KF2), the dynamic model and the voter are also depicted in Figure 6.17. As can be

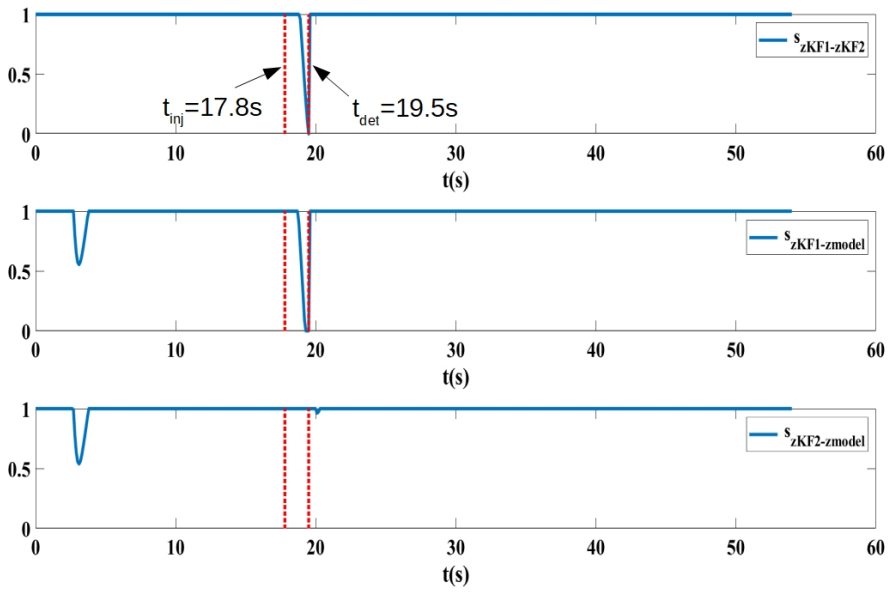


Figure 6.18 – Lidar additive fault: The agreement indicators of the altitude of the first and second Kalman filters and the model

seen, the additive fault injected to the system causes the estimated altitude by KF1 to deviate from all the other systems.

In Figure 6.18, it can be noticed that after the fault injection at $t_{inj} = 17.8s$ the agreement indicator of the altitude between the two Kalman filter blocks $s_{zKF1-zKF2}$ starts to fall down from $s_{zKF1-zKF2} = 1$ and reaches $s_{zKF1-zKF2} = 0$ after 1.7s at $t_{det} = 19.5s$. As in the previous experiment, the agreement indicator between KF1 and the model behaves in the same way. However, here the agreement indicator between KF2 and the model stays at the maximum value of 1, probably because of a more conservative threshold value than for the position, thanks to the better precision of the lidar compared to the GPS.

As described in algorithm 1, an error has been detected due to the value of $s_{zKF1-zKF2}$. Figure 6.18 shows that the two agreement indicator on the model points to an error in the branch B1.

After identifying the erroneous branch, the next step is to diagnose if the error is due to a hardware or a software fault. Thus we compare the outputs of the Lidar used in KF1 and KF2 which are considered independent. Figure 6.19 shows the measured altitudes of the Lidar $zr1$ and $zr2$ in KF1 and KF2 respectively, and their residues. We can see that the residuals of the $zr1$ and $zr2$ exceeds the $TH_{rg} = 1m$ threshold at $t_{det} = 19.5s$, thus a hardware fault is reported at $t_{det} = 19.5s$. We do not need to compare other sensors results here, as the lidar is the only sensor determining the altitude in our data fusion. Since we have already identified that the first branch is the erroneous branch from the previous step, we can conclude that the faulty sensor is the $zr1$ in KF1.

Once $zr1$ is detected as the faulty sensor, the first branch is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by

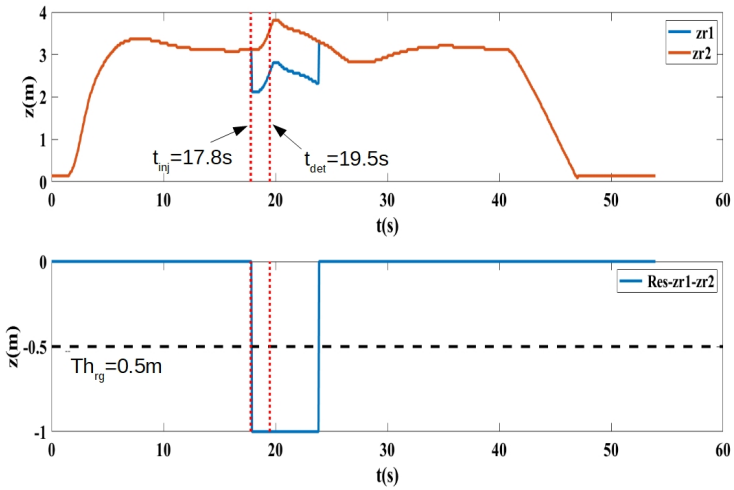


Figure 6.19 – Lidar additive fault: The measured altitude and the residues of the Lidar's output used in KF1 and KF2

KF3 is equal to the average of the altitude estimated by KF1 and the Dynamic model. This corresponds to solution 3 in section 6.6.1. This explains why the agreement indicators $s_{zKF1-zKF2}$ and $s_{zKF1-zmodel}$ increases quickly after the identification of the fault in Figure 6.18, since they no longer describe the difference between the faulty branch and the other branches in the system. As in the previous experiment, this solution has the flaws described in 6.6.1, and could have been replaced by the other method presented in the same section.

6.7.4 Software altitude fault

To evaluate the proposed architecture for tolerating software faults developed during this work, we present in this section the simulation results of a software fault injection described in the following.

In order to validate our architecture against a software fault in the altitude estimation, the altitude covariance term P_z is forced to a negative value of -0.1 starting from the beginning of the flight simulation described in the following. In this simulation, the quadrotor is required to perform a hovering flight at a 3 meters altitude. The estimated altitudes by the first and second Kalman filters (KF1 and KF2), the dynamic model and the voter are depicted in Figure 6.20. It can be noted that KF1 still gives an acceptable behavior for more than 15 seconds even if the software fault is injected before the takeoff. Indeed, development software faults are always present in the system.

This injected fault can corresponds to two real faults. First a development error in the value of this term. This can corresponds to a programming error (an incorrect value due to some mistakes from the programmers) or a design error. Indeed, the matrices P and Q in Kalman filters have values that are not easy to determined and can thus be designed incorrectly. Second, it can simulate the propagation of other software errors during operation. Consider that the the effect of rounding errors on the state estimation

can be accounted for by calculation errors in the error covariance matrix P during the data fusion mechanism in the Kalman filter. The longer the Kalman filter has been running and the higher the iteration rate, the greater the distortion of the matrix become. The diagonal terms of the covariance matrix represents the estimation uncertainties relative to each state estimation and should always be positive in order to guarantee a stable estimation during the data fusion positive. However, distortions due to other software errors in the system could lead, the estimated covariance terms to become negative as in our injected fault, which will cause divergence in the estimation process.

In Figure 6.21, we can see that the agreement indicator of the altitude between the two Kalman filter blocks $s_{zKF1-zKF2}$ starts to fall down from $s_{zKF1-zKF2} = 1$ and reaches $s_{zKF1-zKF2} = 0$ after 1.7s at $t_{det} = 19s$. As in the previous experiments, the agreement indicator between KF1 and the model behaves in the same way. However, here the agreement indicator between KF2 and the model stays at the maximum value of 1, probably because of a more conservative threshold value than for the position due to the reduced uncertainties in the simulation environment. These results are consistent with the values of the residues between the branches presented in Figure 6.22, where only the residues of KF1 exceed the predefined threshold value of 0.5 m, thus indicating a fault in the first branch.

As described in algorithm 1, an error has been detected due to the value of $s_{zKF1-zKF2}$. Figure 6.21 shows that the two agreement indicator on the model points to an error in the branch B1.

After identifying the erroneous branch, the next step is to identify if the error is due to a hardware or software fault. Thus we compare the outputs of the functionaly equivalent sensors. Figure 6.23 shows the measured altitudes by the first and the second Lidar. As shown in this figure, the altitude measurements of both Lidar show consistency during all the flight time. Since we do not need to compare other sensor results here, as the Lidar is the only sensor used to measure the altitude, this leads to the conclusion that no hardware fault is present in the system. Thus, a software fault is confirmed in KF1.

Once KF1 is detected as the faulty block, it is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by KF3 is equal to the average of the altitude estimated by KF1 and the dynamic model. This is another alternative solution for those presented in section 6.6.2. This explains why the agreement indicators $s_{zKF1-zKF2}$ and $s_{zKF1-zmodel}$ increases quickly after the identification of the fault in Figure 6.21, since they no longer describe the difference between the faulty branch and the other branches in the system. As in the previous experiment, this solution has the flaws described in 6.6.2, and could have been replaced by the other method presented in the same section.

6.7.5 Software position fault

In order to validate our architecture against a software fault in the position estimation,the position covariances terms P_x and P_y are forced to a small negative value of -0.01 starting from the beginning of the flight simulation described in the following. In this simulation, The UAV is required to follow a U-path trajectory starting from the initial point ($x = 0, y = 0$) to the destination point ($x = 26, y = 5$). The estimated positions by the first and second Kalman filters (KF1 and KF2), the dynamic model and the voter are depicted in

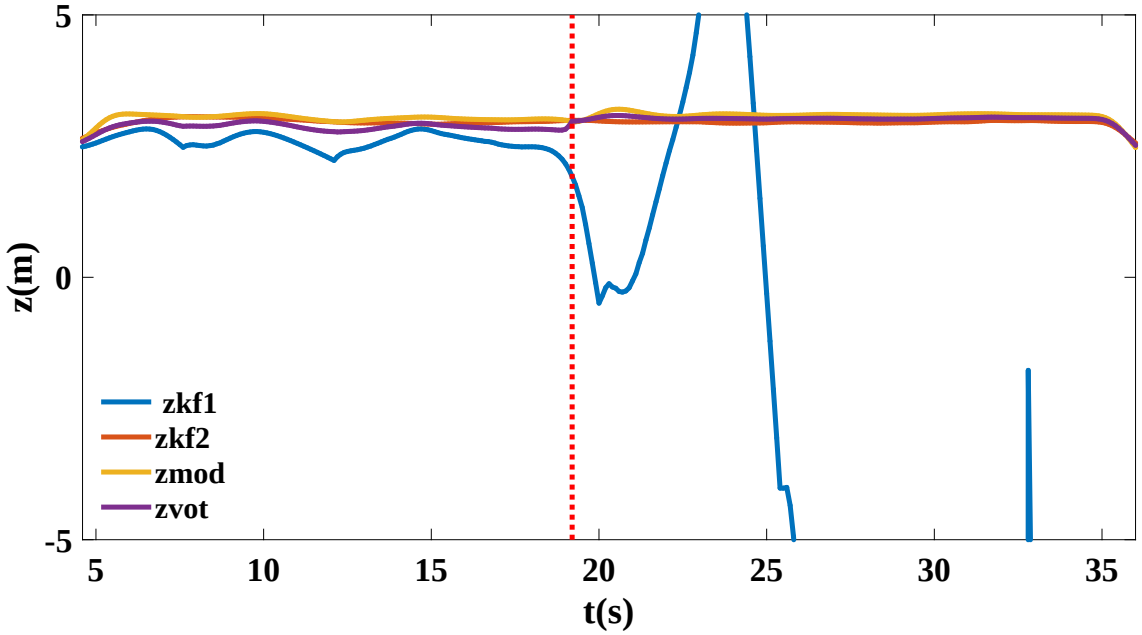


Figure 6.20 – Software fault: altitude estimated by the two Kalman filters, the dynamic model and the voter

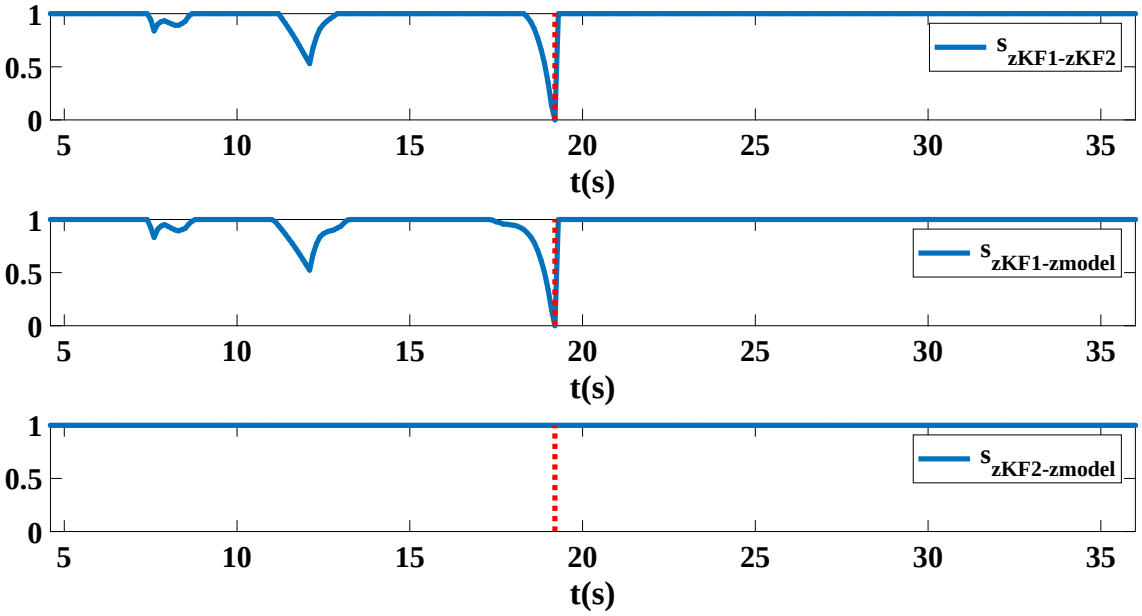


Figure 6.21 – Software fault: the agreement indicators of the altitude of the first and second Kalman filters and the model

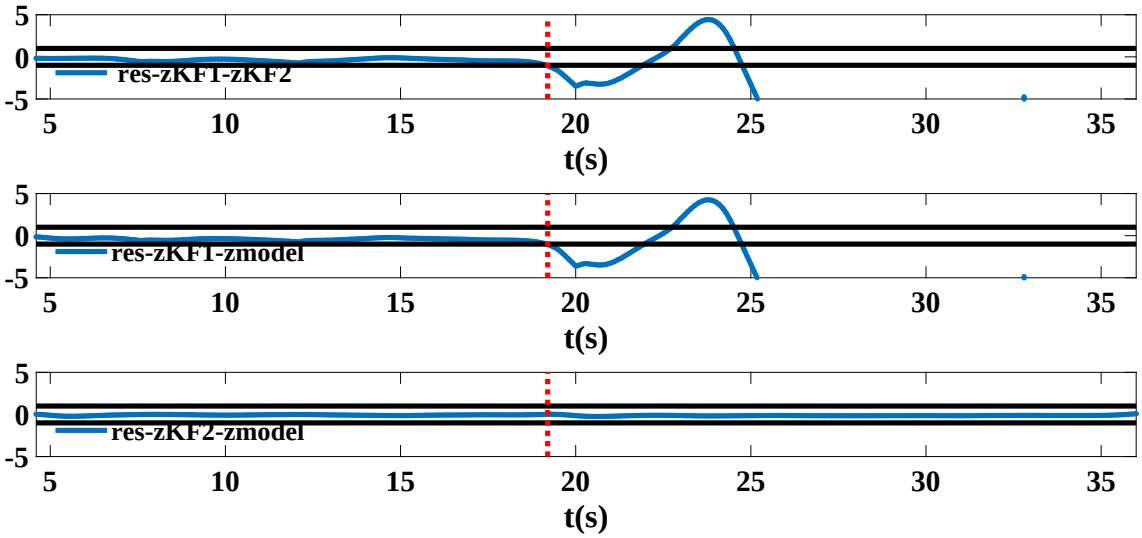


Figure 6.22 – Software fault: the residues of the altitude estimations between KF1; KF2 and the DM

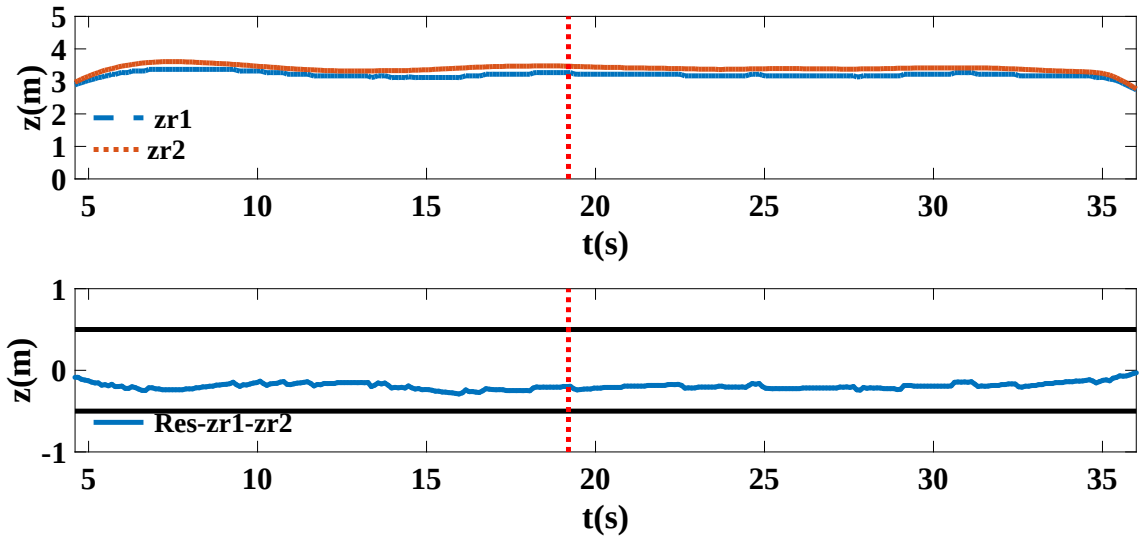


Figure 6.23 – Software fault: The residues of the z components between the first simulated Lidar and the second simulated Lidar

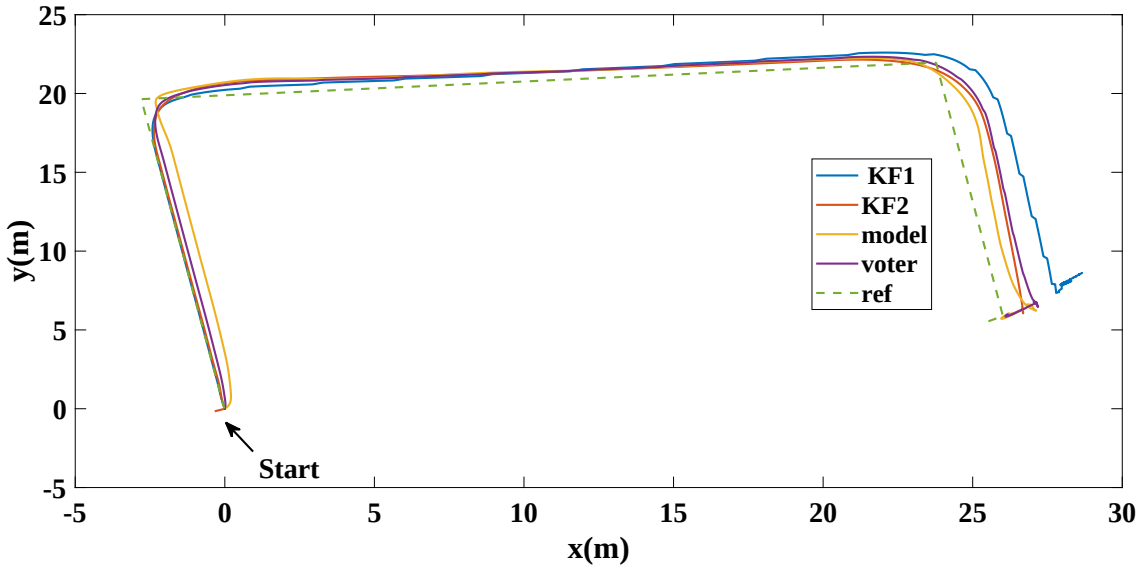


Figure 6.24 – Software fault: positions estimated by the two Kalman fi lters, the dynamic model, the voter

Figure 6.24. It can be noted that KF1 still gives an acceptable behavior for more than 32.5 seconds even if the software fault is injected before the takeoff. Indeed, development software faults are always present in the system.

In Figure 6.26, we can see that the agreement indicator of the Euclidean distance between the two Kalman filter blocks $s_{dKF1-dKF2}$ starts to fall down from $s_{dKF1-dKF2} = 1$ and reaches $s_{dKF1-dKF2} = 0$ after 18s at $t_{det} = 32.5s$. As in the previous experiments, the agreement indicator between KF1 and the model behaves in the same way. However, here the agreement indicator between KF2 and the model stays at the maximum value of 1, probably because of a more conservative threshold value than for the position due to the reduced uncertainties in the simulation environment.

As described in algorithm 1, an error has been detected due to the value of $s_{dKF1-dKF2}$. Figure 6.26 shows that the two agreement indicator on the model points to an error in the branch B1.

After identifying the erroneous branch, the next step is to identify if the error is due to a hardware or software fault. Thus we compare the outputs of the functionaly equivalent sensors. The Figures 6.25 and 6.27 shows that the sensors contributing in the estimation of the position (Acc1 & Acc2 for the prediction step and GPS1 & GPS2 for the correction step in the Data fusion process) show consistency during all the flight time. Thus, a software fault is confirmed in KF1.

Once KF1 is detected as the faulty block, it is immediately removed from the system and a new localization system KF3 is defined, where the position estimated by KF3 is equal to the average of the altitude estimated by KF1 and the dynamic model. This explains why the agreement indicators $s_{dKF1-dKF2}$ and $s_{dKF1-dmodel}$ increases quickly after the identification of the fault in Figure 6.26, since they no longer describe the difference between the faulty branch and the other branches in the system.

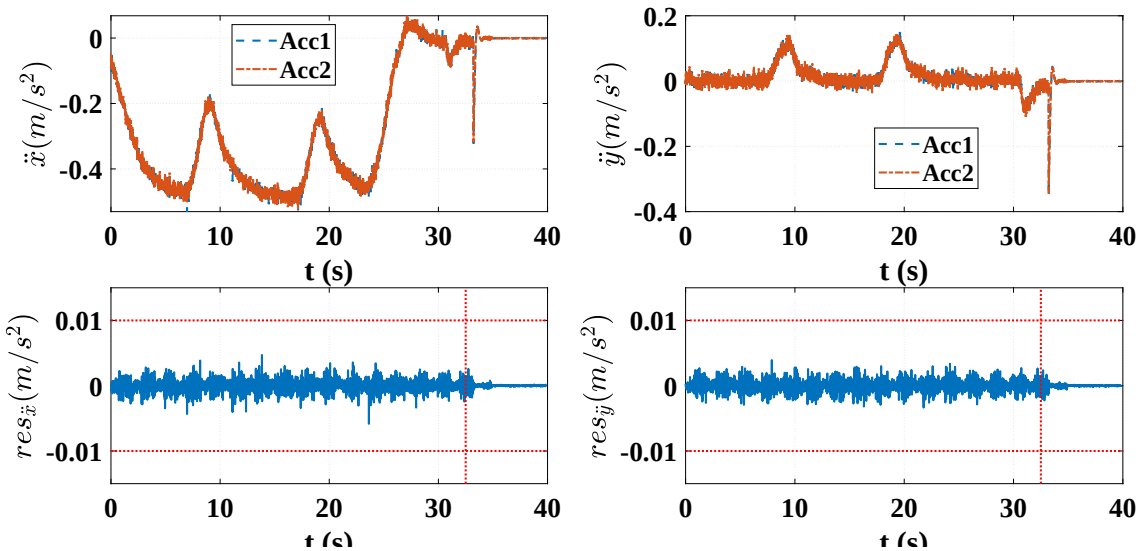


Figure 6.25 – Software fault: accelerations estimates by the two Accelerometers and their corresponding residues

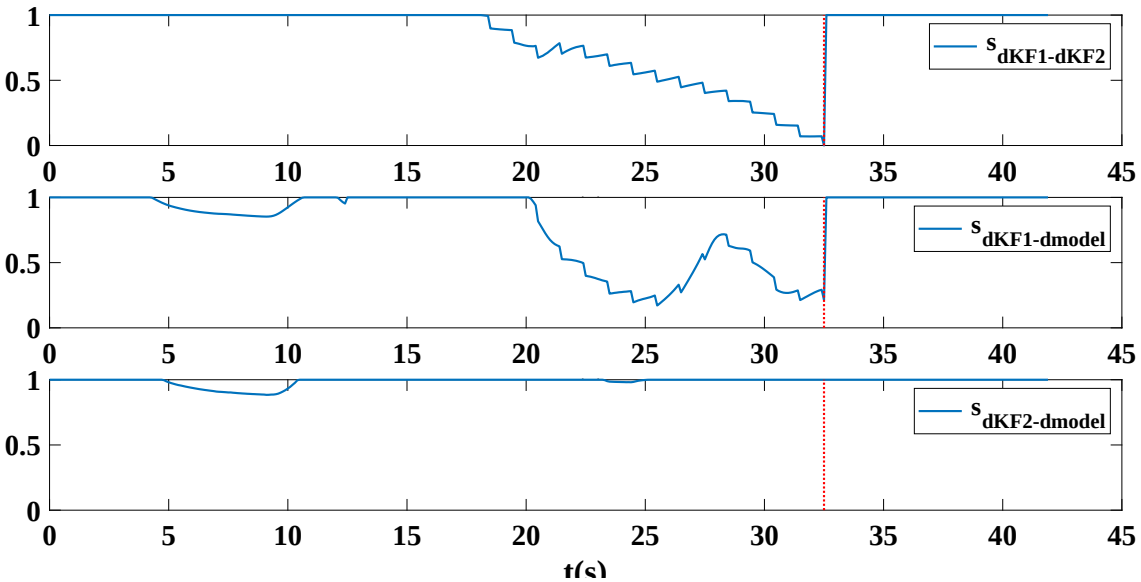


Figure 6.26 – Software fault: the agreement indicators of the Euclidean distance of the first and second Kalman filters and the DM

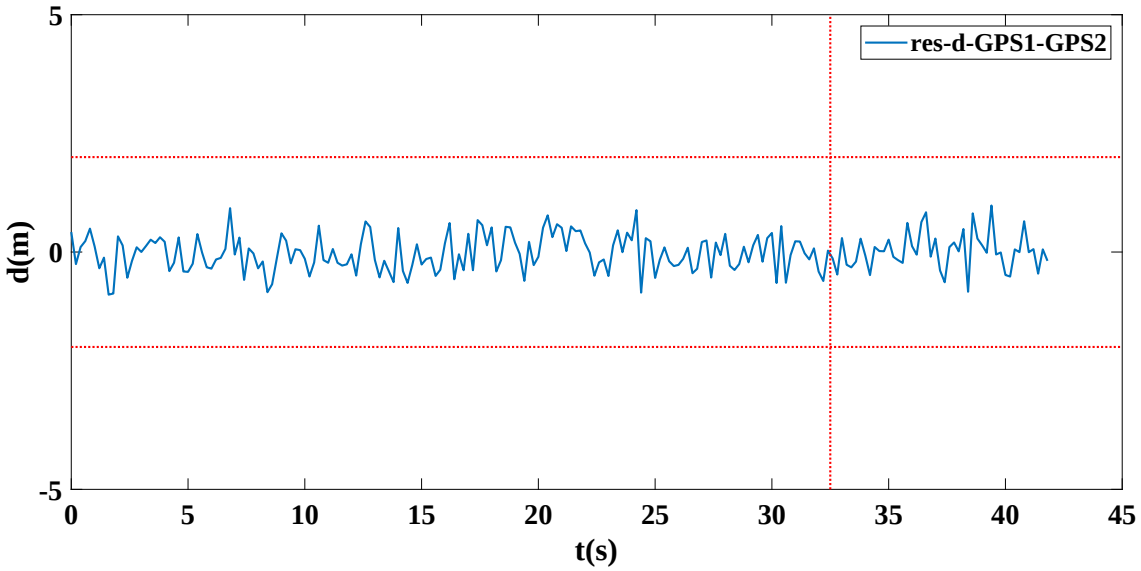


Figure 6.27 – Software fault: The residues of the Euclidean distance measured by the first GPS and the second GPS

6.8 Conclusion

In this paper, we have presented our proposed data fusion architecture for tolerating sensors and software faults which is based on duplication and comparison and was initially proposed in [19]. Four main differences exist between our proposed architecture and the one proposed in [19]: (1) The first difference is that we propose to add an analytical redundancy using the dynamic model (DM). (2) The second difference is that we use a weighted average voter instead of a simple thresholding, which increases the accuracy of the outputs and the error detection process. (3) The third difference is that we have multiple solutions which can be applied to recover a faulty system, increasing the flexibility of our architecture. (4) The fourth difference is that our architecture is experimentally validated in real outdoor environment using a quadrotor. The experiments show that our architecture is able to deal with hardware faults during real flights, and software faults during simulations.

For perspectives, such redundancy in the proposed architecture could also be exploited to tolerate faults in the dynamic model block using the diversity of data fusion. If the data of diversified sensors measuring a state are consistent and the estimation of the same state using the dynamic model is not coherent with the sensors data, thus we need to reset the output of the dynamic model to one of the sensors measurements or to their average. Such scenario can occurs when the uncertainties on the equation of motions become significant due to external perturbations or aggressive maneuvers. It is also interesting to test the effectiveness of the architecture regarding successive and simultaneous hardware or software or mixed faults in the system. This way the proposed solutions for the recovery may be analyzed further more in terms of robustness and sensitivity to different conditions of faulty situations. Also, another interesting topic is to apply this architecture to different robotics systems such as rovers and plans where other types of hardware and software faults can be tested.

Conclusions and Outlook

In this final section, we first summarize the contributions concerning the wind force compensation strategy for a quadrotor, the fault-tolerant control strategy for a coaxial octorotor regarding motor failures based on a self-tuning sliding mode control (STSMC) and the fault-tolerant architecture for perception targeting sensors and software faults based on duplication-comparison and a weighted average voting system. We then suggest possible improvements upon our work and topics for future studies.

Conclusions

Many research studies on robustness against external wind perturbations and fault-tolerant control of unmanned aerial vehicles have been developed in the literature. However, only some of them are applied to real UAV systems during flights. Moreover, most researchers working on UAVs assume that robust controllers are sufficient to deal with wind perturbations, which we found not true from our experiments, apart from the adaptive technique. However this adaptive technique requires second order control laws to tolerate wind perturbations as matched uncertainties, and these control laws are hard to design and to tune. Also, in order to maintain complete control on the UAV in the case of one or more motor failures, to our knowledge, the only solution is to consider multirotors with redundant actuators, which adds the possibility of tolerating multiple failures without losing complete controllability unlike the case of a quadrotor. However, due to the increasing number of sensors and the underlying data fusion mechanisms in UAVs, the risk of software and hardware faults increases, in terms of sensor failures and processing failures. Hence, there is a need to apply a fault tolerant strategy to overcome these issues and detect any failures, and to ensure more reliable performance outcomes with respect to autonomous systems.

In this thesis, we present the design and implementation of a wind force compensation strategy for a quadrotor. This strategy relies on a second-order sliding mode controller based on the super twisting algorithm (STA) with a nonlinear observer. The STA controller ensures robustness to matched external disturbances and time varying, parametric and nonlinear uncertainties. Also, to extend the robustness of the controller against wind perturbation, we propose the integration of a wind observer in the closed-loop system. This estimation allows a better monitoring of the system's status than passive robustness, providing the opportunity for a recovery procedure such as an emergency landing when the external perturbations become too strong for the system. The effectiveness of the proposed strategy is compared to an adaptive gain controller through simulations and validated in real experiments on the *DJIS*500 quadrotor.

Regarding motor failures, three fault-tolerant control (FTC) strategies for a coaxial octorotor UAV were presented. The first FTC is based on a control mixing (Multiplexing) strategy which consists of a set of control laws designed offline, each one is dedicated to a specific fault situation. In the second FTC, a robust adaptive sliding mode control allocation is presented, where the control gains of the controller are adjusted online in order to redistribute the control signals among the healthy motors in order to stabilize the overall system. The third FTC strategy is a new strategy proposed in this thesis, which is based on a self-tuning sliding mode control (STSMC) where the control gains are readjusted from the detected error to maintain the stability of the system. Multiple indoor experiments on an octorotor UAV are conducted to show and compare the effectiveness and the behavior of each FTC scheme after the injection of successive faults. Based on the experimental results we obtained, it can be concluded that the proposed STSMC, the Multiplexing, and the Adaptive fault tolerant control strategies all allow a coaxial octorotor to maintain stability after losing up to four motors. Although the Multiplexing FTC scored the lowest position errors and time complexity, it is not efficient in development cost as every possible fault scenario must be studied both theoretically and experimentally to determine respectively the multiplexing parameters of the control law and its gains. Meanwhile, the proposed STSMC can be considered as a good intermediate alternative to both strategies, since it shows better performance and faster response time compared to the Adaptive method and allows to monitor the system's health with almost no added development cost. It can also be easily implemented to allow robustness against wind disturbances, unlike the adaptive method that requires complex gains tuning.

Finally, since the open source Arducopter flight stack is vulnerable to software and hardware faults, we present our proposed fault tolerance architecture for data fusion, which uses a voting system, a redundancy based approach (Duplication/Comparison) and an analytical redundancy using the equations of motions. Then, we present the fault detection, identification and recovery services proposed in our architecture, and we present a case study on the quadrotor *Tarot*650. Our proposed data fusion architecture for tolerating sensors and software faults is based on duplication and comparison and was initially proposed in [19]. We also use a redundant predictive block based on a mathematical model of the UAV for both error detection and diagnostic and the weighted average voting system. Four main additions have been made between our proposed architecture and the one proposed in [19]: (1) The first difference is that we add an analytical redundancy using the dynamic model of the system (DM). (2) The second difference is that we use a weighted average voter instead of a simple thresholding, which increases the accuracy of the outputs and the error detection process. (3) The third difference is that we have multiple solutions which can be applied to recover a faulty system and this increases the flexibility of our architecture. (4) The fourth difference is that our architecture is experimentally validated in real outdoor environment using a quadrotor. The experiments shows that our architecture is able to deal with hardware faults during real flights, however it would be also interesting to test the effectiveness of this architecture in handling software faults as well. The effectiveness of the proposed strategy is shown through real outdoor experiments using the *Tarot*650.

Perspectives

In the following, we present some potential future research prospects in continuation of this thesis:

- **Wind compensation**: In our work regarding wind-tolerant strategy, we compared our strategy with an adaptative controller through simulations. In both cases, our proposed method gave better results although the adaptive method also gives very good results in simulations. It is however very hard to implement in real experiments as it has numerous interdependent gains to tune. It would be interesting to compare the behavior of both methods in real experimental flights.

  Also, supplementary tests and improvements are needed to completely validate this method: we only tested its effectiveness against moderate wind (up to 0.2 N), and we need to experiment in more aggressive environment where wind force can reach up to several newtons.

- **Fault tolerance under motors failures**: It would be interesting to consider partial faults and simultaneous faults/failures to conduct better analyses and to study more extensively the advantages of each FTC strategy. Conducting the same experiments outdoor by adding the wind compensation strategy would be very interesting. Also, the baseline sliding mode controller could be replaced by the finite-time sliding mode controller in the FTCs strategies in order to study how it improves the system's stability during recovery.

- **Fault tolerance under sensor and software faults**: The number of experiments we have carried out are not representative of all the faults that our system may encounter, although we believe that the injected faults represents realistic faults that our localization system may face. Moreover, our execution context is only one example of a data fusion application in the robotics field, it would be interesting to adapt our fault tolerance mechanisms for other types of applications such as obstacle detection and object tracking.

Bibliography

[1] A.Freddi, S. Longhi, and A. Monteri. Actuator fault detection system for a mini-quadrotor. In *International Symposuim on Industrial Electronics (ISIE)*, pages 2055–2060, July 2010.

[2] A Pedro Aguiar and Joao P Hespanha. Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty. *IEEE transactions on automatic control*, 52(8):1362–1379, 2007.

[3] H. Aguilar-Sierra, G. Flores, S. Salazar, and R. Lozano. Fault estimation for a quad-rotor mav using a polynomial observer. *Journal of Intelligent and Robotic Systems*, 73(1):455–468, 2014.

[4] Abdelouahab Aitouche and Belkacem Ould-Bouamama. Sensor location with respect to fault tolerance properties. *International Journal of Automation and Control*, 4(3):298–316, 2010.

[5] Joelle Al Hage, Maan E El Najjar, and Denis Pomorski. Multi-sensor fusion approach with fault detection and exclusion based on the kullback–leibler divergence: Application on collaborative multi-robot system. *Information Fusion*, 37:61–76, 2017.

[6] Kostas Alexis, George Nikolakopoulos, and Anthony Tzes. Experimental model predictive attitude tracking control of a quadrotor helicopter subject to wind-gusts. *18th Mediterranean Conference on Control & Automation (MED)*, pages 1461–1466, 2010.

[7] Abdel Ilah Nour Alshbatat, Liang Dong, and Peter James Vial. Controlling an unmanned quad-rotor aerial vehicle with model parameter uncertainty and actuator failure. *International Journal of Intelligent Systems Technologies and Applications*, 15(4):295–322, 2016.

[8] H. Alwi and C. Edwards. Fault tolerant control of an octorotor using lpv based sliding mode control allocation. In *IEEE American Control Conference (ACC)*, pages 6505–6510, June 2013.

[9] H. Alwi and C. Edwards. Lpv sliding mode fault tolerant control of an octorotor using fixed control allocation. In *Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 772–777, October 2013.

[10] H. Alwi and C. Edwards. Sliding mode fault-tolerant control of an octorotor using linear parameter varying-based schemes. *IET Control Theory and Applications*, 9(4):618–636, 2015.

[11] Halim Alwi and Christopher Edwards. Sliding mode fault-tolerant control of an octorotor using linear parameter varying-based schemes. *IET Control Theory & Applications*, 9:618–636, 2015.

[12] Halim Alwi, Christopher Edwards, and Chee Pin Tan. *Fault detection and fault-tolerant control using sliding modes*. Springer Science & Business Media, 2011.

[13] Roohul Amin, Li Aijun, and Shahaboddin Shamshirband. A review of quadrotor uav: control methodologies and performance evaluation. *International Journal of Automation and Control*, 10(2):87–103, 2016.

[14] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, 2004.

[15] Remus C Avram, Xiaodong Zhang, and Jonathan Muse. Nonlinear adaptive fault-tolerant quadrotor altitude and attitude tracking with multiple actuator faults. *IEEE Transactions on Control Systems Technology*, 26:701–707, 2018.

[16] Ahmad Taher Azar and Quanmin Zhu. *Advances and applications in sliding mode control systems*. Springer, 2015.

[17] Andrea Bacciotti and Lionel Rosier. *Liapunov functions and stability in control theory*. Springer, Lecture Notes in Control and Information Sciences, 2001.

[18] Kaci Bader. *Tolérance aux fautes pour la perception multi-capteurs : application à la localisation d'un véhicule intelligent*. PhD thesis, University of technology of Compiegne, 2014.

[19] Kaci Bader, Benjamin Lussier, and Walter SchÃűn. A fault tolerant architecture for data fusion: A real application of kalman filters for mobile robot localization. *Robotics and Autonomous Systems*, 88:11 – 23, 2017.

[20] Hamed Badihi, Youmin Zhang, and Henry Hong. Fuzzy gain-scheduled active fault-tolerant control of a wind turbine. *Journal of the Franklin Institute*, 351(7):3677–3706, 2014.

[21] Muhammad Abu Bakr and Sukhan Lee. Distributed multisensor data fusion under unknown correlation and data inconsistency. *Sensors*, 17(11):2472, 2017.

[22] A Baldini, R Felicetti, A Freddi, S Longhi, and A Monteriù. Octarotor fault tolerant control via dynamic surface control. In *IEEE 18th European Control Conference (ECC)*, pages 3892–3897, 2019.

[23] Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.

[24] Denis Berdjag, Jérôme Cieslak, and Ali Zolghadri. Fault detection and isolation of aircraft air data/inertial system. *Progress in Flight Dynamics, Guidance, Navigation, Control, Fault Detection, and Avionics*, 6:317–332, 2013.

[25] Sanjay P Bhat and Dennis S Bernstein. Finite-time stability of continuous autonomous systems. *SIAM Journal on Control and Optimization*, 38(3):751–766, 2000.

[26] Isabelle Bloch, Anthony Hunter, Alain Appriou, André Ayoun, Salem Benferhat, Philippe Besnard, Laurence Cholvy, Roger Cooke, Frédéric Cuppens, and Didier Dubois. Fusion: General concepts and characteristics. *International journal of intelligent systems*, 16(10):1107–1134, 2001.

[27] Jovan Boskovic and Raman Mehra. A multiple model-based reconfigurable flight control system design. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 4, pages 4503–4508, 1998.

[28] Jovan Boskovic, Ssu-Hsin Yu, and Raman Mehra. Stable adaptive fault-tolerant control of overactuated aircraft using multiple models, switching and tuning. In *Guidance, Navigation, and Control Conference and Exhibit*, pages 739–749, 1998.

[29] Henrik Boström, Sten F Andler, Marcus Brohede, Ronnie Johansson, Alexander Karlsson, Joeri Van Laere, Lars Niklasson, Marie Nilsson, Anne Persson, and Tom Ziemke. On the definition of information fusion as a field of research. *Informatics Research Centre, University of Skovde*, 2007.

[30] S. Bouabdallah. *Design and Control of Quadrotors With Application to Autonomous Flying*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 2007.

[31] Tammaso Bresciani. *Modelling, identification and control of a quadrotor helicopter*. PhD thesis, Lund University, 2008.

[32] David Cabecinhas, Rita Cunha, and Carlos Silvestre. A globally stabilizing path following controller for rotorcraft with wind disturbance rejection. *IEEE Transactions on Control Systems Technology*, 23(2):708–714, 2014.

[33] Jossué Carino, Hernan Abaunza, and P Castillo. Quadrotor quaternion control. In *IEEE International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 825–831, 2015.

[34] A. Chamseddine, Y. Zhang, C.-A. Rabbath, J. Apkarian, and C. Fulford. Model reference adaptive fault tolerant control of a quadrotor uav. In *AIAA Infotech@Aerospace*, March 2011.

[35] Abbas Chamseddine, Didier Theilliol, YM Zhang, Cédric Join, and Camille-Alain Rabbath. Active fault-tolerant control system design with trajectory re-planning against actuator faults and saturation: Application to a quadrotor unmanned aerial vehicle. *International Journal of Adaptive Control and Signal Processing*, 29(1):1–23, 2015.

[36] Fuyang Chen, Rongqiang Jiang, Kangkang Zhang, Bin Jiang, and Gang Tao. Robust backstepping sliding-mode control and observer-based fault estimation for a quadrotor uav. *IEEE Transactions on Industrial Electronics*, 63(8):5044–5056, 2016.

[37] Fuyang Chen, Qingbo Wu, Bin Jiang, and Gang Tao. A reconfiguration scheme for quadrotor helicopter via simple adaptive control and quantum logic. *IEEE Transactions on Industrial Electronics*, 62(7):4328–4335, 2015.

[38] M. Chen, Y. Hwang, and M. Tomizuka. A state-dependent boundary layer design for sliding mode control. *IEEE Transactions On Automatic Control*, 47(10):1677–1681, 2002.

[39] Ran Chen, Zhihua Zhang, Ping Zhang, and Michael Mangold. Fault tolerant control for hexacopter with reducing yaw rate. In *IEEE 4th Conference on Control and Fault Tolerant Systems (SysTol)*, pages 171–176, 2019.

[40] Yong Chen and Bin Guo. Sliding mode fault tolerant tracking control for a single-link flexible joint manipulator system. *IEEE Access*, 7:83046–83057, 2019.

[41] Byoung-Suk Choi and Ju-Jang Lee. The position estimation of mobile robot under dynamic environment. In *IECON Annual Conference of the IEEE Industrial Electronics Society*, pages 134–138, 2007.

[42] P Cominos and N Munro. Pid controllers: recent tuning methods and design to specification. *IET Control Theory and Applications*, 149(1):46–53, 2002.

[43] Ren Da and Ching-Fang Lin. A new failure detection approach and its application to gps autonomous integrity monitoring. *IEEE transactions on Aerospace and Electronic Systems*, 31(1):499–506, 1995.

[44] Samar Dajani-Brown, Darren Cofer, Gary Hartmann, and Steve Pratt. Formal modeling and analysis of an avionics triplex sensor voter. In *International SPIN Workshop on Model Checking of Software*, pages 34–48. Springer, 2003.

[45] J. Davila, L. Fridman, and A. Levant. Second-order sliding-mode observer for mechanical systems. *IEEE Transactions On Automatic Control*, 50(11):1785–1789, 2005.

[46] Diego Del Gobbo, Marcello Napolitano, Parviz Famouri, and Mario Innocenti. Experimental application of extended kalman filtering for sensor validation. *IEEE Transactions on control systems technology*, 9(2):376–380, 2001.

[47] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. *ICRA*, 2:1322–1328, 1999.

[48] Arthur P Dempster. Upper and lower probabilities induced by a multivalued mapping. In *Classic Works of the Dempster-Shafer Theory of Belief Functions*, pages 57–72. Springer, 2008.

[49] L. Derafa, A. Benallegue, and L. Fridman. Super twisting control algorithm for the attitude tracking of a four rotors UAV. *Journal of the Franklin Institute*, 349(2):685–699, 2012.

[50] L. Derafa, A. Benallegue, and L. Fridman. Super twisting control algorithm for the attitude tracking of a four rotors uav. *Journal of Franklin Institute*, 349(2):685–699, 2012.

[51] Wei Dong, Guo-Ying Gu, Xiangyang Zhu, and Han Ding. High-performance trajectory tracking control of a quadrotor with disturbance observer. *Sensors and Actuators A: Physical*, 211:67–77, 2014.

[52] Rajamani Doraiswami and Lahouari Cheded. A unified approach to detection and isolation of parametric faults using a kalman filter residual-based approach. *Journal of the Franklin Institute*, 350(5):938–965, 2013.

[53] Guang-Xun Du, Quan Quan, and Kai-Yuan Cai. Controllability analysis and degraded control for a class of hexacopters subject to rotor failures. *Journal of Intelligent and Robotic Systems*, 78(1):143–157, 2015.

[54] Didier Dubois and Henri Prade. Possibility theory and data fusion in poorly informed environments. *Control Engineering Practice*, 2(5):811–823, 1994.

[55] Didier Dubois and Henri Prade. *Possibility theory*. Springer, 2012.

[56] Guy A Dumont and Mihai Huzmezan. Concepts, methods and techniques in adaptive control. In *IEEE American control conference*, volume 2, pages 1137–1150, 2002.

[57] J. Escareño, S. Salazar, H. Romero, and R. Lozano. Trajectory control of a quadrotor subject to 2d wind disturbances. *Journal of Intelligent & Robotic Systems*, 70(1-4):51–63, 2013.

[58] Ramsey Faragher et al. Understanding the basis of the kalman filter via a simple and intuitive derivation. *IEEE Signal processing magazine*, 29(5):128–132, 2012.

[59] Leonie Freeston. Applications of the kalman filter algorithm to robot localisation and world modelling. *Electrical Engineering Final Year Project*, 2002.

[60] Murali Gopinathan, Jovan D Boskovic, Raman K Mehra, and Constantino Rago. A multiple model predictive scheme for fault-tolerant flight control design. In *Proceedings of the 37th IEEE Conference on Decision and Control*, volume 2, pages 1376–1381, 1998.

[61] Chingiz Hajiyev and Halil Ersin Soken. Robust adaptive kalman filter for estimation of uav dynamics in the presence of sensor/actuator faults. *Aerospace Science and Technology*, 28(1):376–383, 2013.

[62] David L Hall and James Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[63] David Lee Hall and Sonya AH McMullen. *Mathematical techniques in multisensor data fusion*. Artech House, 2004.

[64] Hussein Hamadi, Benjamin Lussier, Isabelle Fantoni, Clovis Francis, and Hassan Shraim. Observer-based super twisting controller robust to wind perturbation for multirotor uav. pages 397–405, 2019.

[65] Ola Härkegård. *Backstepping and control allocation with applications to flight control*. PhD thesis, Linköpings university, 2003.

[66] E Pulido Herrera, Hannes Kaufmann, J Secue, Ricardo Quirós, and Germán Fabregat. Improving data fusion in personal positioning systems for outdoor environments. *Information Fusion*, 14(1):45–56, 2013.

[67] Haomiao Huang, Gabriel M Hoffmann, Steven L Waslander, and Claire J Tomlin. Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3277–3282, 2009.

[68] Sunan Huang, Kok Kiong Tan, and Tong Heng Lee. Fault diagnosis and fault-tolerant control in linear drives using the kalman filter. *IEEE Transactions on Industrial Electronics*, 59(11):4285–4292, 2012.

[69] I. Hwang, S. Kim, Y. Kim, and C. E. Seah. A survey of fault detection, isolation, and reconfiguration methods. 18:636–653, 2010.

[70] H. A. Izadi, Y. Zhang, and B. W. Gordon. Fault tolerant model predictive control of quad-rotor helicopters with actuator fault estimation. In *IFAC World Congress*, pages 6343–6348, August 2011.

[71] Hicham Jamouli and Dominique Sauter. A generalized likelihood ratio test for a fault-tolerant control system. In *International Conference on Advances in Computational Tools for Engineering Applications*, pages 474–479. IEEE, 2009.

[72] Li Jiang. *Sensor Fault Detection and Isolation Using System Dynamics Identification Techniques*. PhD thesis, The University of Michigan, 2011.

[73] Tor A Johansen and Thor I Fossen. Control allocationâĂŤa survey. *Automatica*, 49(5):1087–1103, 2013.

[74] Tor A Johansen and Thor I Fossen. Control allocationâĂŤa survey. *Automatica*, 49(5):1087–1103, 2013.

[75] DA Joosten and JM Maciejowski. Mpc design for fault-tolerant flight control purposes based upon an existing output feedback controller. *IFAC Proceedings Volumes*, 42(8):253–258, 2009.

[76] B Jung, Youdan Kim, and C Ha. Fault tolerant flight control system design using a multiple model adaptive controller. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 223(1):39–50, 2009.

[77] Qiu Kai, Yin Hui, Yan Xiao Peng, and Ren Yan. An integrated fault detection scheme for the federated filter. In *Fourth International Conference on Digital Manufacturing & Automation*, pages 161–164, 2013.

[78] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[79] Simon Karpenko, Ivan Konovalenko, Alexander Miller, Boris Miller, and Dmitry Nikolaev. Stochastic control of uav on the basis of robust filtering of 3d natural landmarks observations. *Proceedings of the 39th IITP RAS Inter-disciplinary Conference & School*, pages 7–18, 2015.

[80] J. Kasac, S. Stevanovic, T. Zillic, and J. Stepanic. Robust output tracking control of a quadrotor in the presence of external disturbances. *Transactions of FAMENA*, pages 29–42, 2014.

[81] Thomas Kerr. Decentralized filtering and redundancy management for multisensor navigation. *IEEE Transactions on Aerospace and Electronic Systems*, (1):83–119, 1987.

[82] Son-Goo Kim, John L Crassidis, Yang Cheng, Adam M Fosbury, and John L Junkins. Kalman filtering for relative spacecraft attitude and position estimation. *Journal of Guidance, Control, and Dynamics*, 30(1):133–143, 2007.

[83] Young-Man Kim. Robust data driven h-infinity control for wind turbine. *Journal of the Franklin Institute*, 353(13):3104–3117, 2016.

[84] Manish Kumar, Devendra P Garg, and Randy A Zachery. A method for judicious fusion of inconsistent multiple sensor data. *IEEE Sensors Journal*, 7(5):723–733, 2007.

[85] Alexander Lanzon, Alessandro Freddi, and Sauro Longhi. Flight control of a quadrotor vehicle subsequent to a rotor failure. *Journal of Guidance, Control, and Dynamics*, 37:580–591, 2014.

[86] GR Latif-Shabgahi. A novel algorithm for weighted average voting used in fault tolerant computing systems. *Microprocessors and Microsystems*, 28(7):357–361, 2004.

[87] Jangho Lee, Dongho Shin, Hyeok Ryu, Dasol Lee, and David Hyunchul Shim. Fault tolerant adaptive control using time delay control scheme under motor faults of octocopter. In *IEEE 7th International Conference on Systems and Control (ICSC)*, pages 123–128, 2018.

[88] François Léonard, Adnan Martini, and Gabriel Abba. Robust nonlinear controls of model-scale helicopters under lateral and vertical wind gusts. *IEEE Transactions on Control Systems Technology*, 20(1):154–163, 2012.

[89] Arie Levant. Higher-order sliding modes, differentiation and output-feedback control. *International journal of Control*, 76(9-10):924–941, 2003.

[90] Arie Levant. Homogeneity approach to high-order sliding mode design. *Automatica*, 41(5):823–830, 2005.

[91] Yongming Li and Shaocheng Tong. Command-filtered-based fuzzy adaptive control design for mimo-switched nonstrict-feedback nonlinear systems. *IEEE Transactions on Fuzzy Systems*, 25(3):668–681, 2016.

[92] Yongming Li, Shaocheng Tong, Lu Liu, and Gang Feng. Adaptive output-feedback control design with prescribed performance for switched nonlinear systems. *Automatica*, 80:225–231, 2017.

[93] V. Lippiello, F. Ruggiero, and D. Serra. Emergency landing for a quadrotor in case of a propeller failure: A backstepping approach. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4782–4788, September 2014.

[94] V. Lippiello, F. Ruggiero, and D. Serra. Emergency landing for a quadrotor in case of a propeller failure: A pid based approach. In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–7, October 2014.

[95] Chun Liu, Bin Jiang, and Ke Zhang. Adaptive fault-tolerant h-infinity output feedback control for lead-wing close formation flight. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018.

[96] Lennart Ljung. Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems. *IEEE Transactions on Automatic Control*, 24(1):36–50, 1979.

[97] Binwen Lu, Jianjun Ma, and Zhiqiang Zheng. Control allocation based adaptive dynamic inversion fault tolerant control scheme for an uncertain aircraft. In *IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, pages 907–912, 2016.

[98] Peng Lu and Erik-Jan van Kampen. Active fault-tolerant control for quadrotors subjected to a complete rotor failure. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4698–4703. IEEE, 2015.

[99] Peng Lu, Erik-Jan van Kampen, Cornelis de Visser, and Qiping Chu. Aircraft fault-tolerant trajectory control using incremental nonlinear dynamic inversion. *Control Engineering Practice*, 57:126–141, 2016.

[100] Benjamin Lussier. *Tolérance aux fautes dans les systèmes autonomes*. PhD thesis, Institut National Polytechnique de Toulouse, 2007.

[101] Marcel Luzar and Marcin Witczak. Fault-tolerant control and diagnosis for lpv system with h-infinity virtual sensor. In *3rd Conference on Control and Fault-Tolerant Systems (SysTol)*, pages 825–830. IEEE, 2016.

[102] Zehui Mao, Xing-Gang Yan, Bin Jiang, and Mou Chen. Adaptive fault-tolerant sliding-mode control for high-speed trains with actuator faults and uncertainties. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[103] Andrés Marcos and Gary Balas. Development of linear-parameter-varying models for aircraft. *Journal of Guidance, Control, and Dynamics*, 27(2):218–228, 2004.

[104] Andrés Marcos, Joost Veenman, Carsten Scherer, Gabriele De Zaiacomo, David Mostaza, Murray Kerr, Hakan Köroglu, and Samir Bennani. Application of lpv modeling, design and analysis methods to a re-entry vehicle. In *AIAA guidance, navigation, and control conference*, pages 81–92, 2010.

[105] A. Marks, J. F. Whidborne, and I. Yamamoto. Control allocation for fault tolerant control of a vtol octorotor. In *UKACC International Conference on Control*, pages 357–362, September 2012.

[106] Adrián Martínez-Vásquez, A Rodriguez-Mata, Iván González-Hernández, Sergio Salazar, Alejandro Montiel-Varela, and Rogelio Lozano. Linear observer for estimating wind gust in uav's. In *Electrical Engineering, 12th International Conference on Computing Science and Automatic Control (CCE)*, pages 1–6, 2015.

[107] Stephan Matzka and Richard Altendorfer. A comparison of track-to-track fusion algorithms for automotive sensor fusion. In *Multisensor Fusion and Integration for Intelligent Systems*, pages 69–81. Springer, 2009.

[108] Mohit Mehndiratta and Erdal Kayacan. Reconfigurable fault-tolerant nmpc for y6 coaxial tricopter with complete loss of one rotor. In *IEEE Conference on Control Technology and Applications (CCTA)*, pages 774–780, 2018.

[109] Hemza Mekki, Djamel Boukhetala, and Ahmad Taher Azar. Sliding modes for fault tolerant control. In *Advances and applications in sliding mode control systems*, pages 407–433. Springer, 2015.

[110] A. Merheb, H. Nourra, and F. Batemann. Active fault tolerant control of octorotor uav using dynamic control allocation. In *International Conference on Intelligent Unmanned systems (ICIUS'14)*, September 2014.

[111] Abdel-Razzak Merheb and Hassan Noura. Active fault-tolerant control of quadrotor uavs based on passive controller bank. In *Mechanism, Machine, Robotics and Mechatronics Sciences*, pages 231–241. Springer, 2019.

[112] Abdel-Razzak Merheb, Hassan Noura, and François Bateman. Emergency control of ar drone quadrotor uav suffering a total loss of one rotor. *IEEE/ASME Transactions on Mechatronics*, 22(2):961–971, 2017.

[113] Abdel-Razzak Merheb, Hassan Noura, and FranÃğois Bateman. Passive and active fault tolerant control of octorotor uav using second order sliding mode control. In *IEEE Conference on Control Applications (CCA)*, pages 1907–1912, 2015.

[114] Alaeddin Milhim, Youmin Zhang, and Camille-Alain Rabbath. Gain scheduling based pid controller for fault tolerant control of quad-rotor uav. In *AIAA infotech and aerospace*. 2010.

[115] Mark W Mueller and Raffaello D'Andrea. Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 45–52, 2014.

[116] Ngoc Phi Nguyen, Nguyen Xuan Mung, and Sung Kyung Hong. Actuator fault detection and fault-tolerant control for hexacopter. *Sensors*, 19(21):4721, 2019.

[117] C Nicol, CJB Macnab, and A Ramirez-Serrano. Robust adaptive control of a quadrotor helicopter. *Mechatronics*, 21(6):927–938, 2011.

[118] Hideaki Okazaki, Siyuan Yin, Kaito Isogai, and Hideo Nakano. Motor speed control signals for multirotor flights in the presence of complete propeller motor failures. In *IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 384–387, 2018.

[119] I. J. Olson and E. Atkins. Qualitative failure analysis for a small quadrotor unmanned aircraft system. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, Boston, MA, August 2013.

[120] Ross T Palomaki, Nathan T Rose, Michael van den Bossche, Thomas J Sherman, and Stephan FJ De Wekker. Wind estimation in the lower atmosphere using multirotor aircraft. *Journal of Atmospheric and Oceanic Technology*, 34(5):1183–1191, 2017.

[121] Ron Patton and S Klinkhieo. Lpv fault estimation and ftc of a two-link manipulator. In *Proceedings of the American Control Conference*, pages 4647–4652, 2010.

[122] Sujit Rajappa, Carlo Masone, Heinrich H Bülthoff, and Paolo Stegagno. Adaptive super twisting controller for a quadrotor uav. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2971–2977, 2016.

[123] Mina Ranjbaran and Khashayar Khorasani. Fault recovery of an under-actuated quadrotor aerial vehicle. In *49th IEEE Conference on Decision and Control (CDC)*, pages 4385–4392. IEEE, 2010.

[124] F. Rinaldi. *Automatic control of a multirotor*. PhD thesis, Politecnico di Torino, 2014.

[125] Héctor Ríos, Shyam Kamal, Leonid M Fridman, and Ali Zolghadri. Fault tolerant control allocation via continuous integral sliding-modes: a hosm-observer approach. *Automatica*, 51:318–325, 2015.

[126] Damiano Rotondo. *Advances in gain-scheduling and fault tolerant control techniques*. Springer, 2017.

[127] Iman Sadeghzadeh, Ankit Mehta, and Youmin Zhang. Fault/damage tolerant control of a quadrotor helicopter uav using model reference adaptive control and gain-scheduled pid. In *AIAA Guidance, Navigation, and Control Conference*, pages 1–10, 2011.

[128] Majd Saied. *Fault-tolerant control of an octorotor unmanned aerial vehicle under actuators failures*. PhD thesis, Compiègne, 2016.

[129] Majd Saied, Benjamin Lussier, Isabelle Fantoni, Clovis Francis, and Hassan Shraim. Fault tolerant control for multiple successive failures in an octorotor: Architecture and experiments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 40–45, 2015.

[130] Majd Saied, Hassan Shraim, Clovis Francis, Isabelle Fantoni, and Benjamin Lussier. Controllability analysis and motors failures symmetry in a coaxial octorotor. In *IEEE Third International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, pages 245–250, 2015.

[131] L. Schmidt, T. Andersen, and H. Pedersen. Robust non-chattering observer based sliding control concept for electro-hydraulic drives. In *6th IFAC Symposium on Mechatronic Systems The International Federation of Automatic Contro*, pages 99–108, April 2013.

[132] T. Schneider, G. Ducard, K. Rudin, and P. Strupler. Fault-tolerant control allocation for multirotor helicopters using parametric programming. In *International Micro Air Vehicle Conference and Flight Competition*, July 2012.

[133] Thomas Schneider, Guillaume Ducard, Konrad Rudin, and Pascal Strupler. Fault-tolerant control allocation for multirotor helicopters using parametric programming. *International Micro Air Vehicle Conferance*, 2012.

[134] Mohammad Sepasi and Farrokh Sassani. On-line fault diagnosis of hydraulic systems using unscented kalman filter. *International Journal of Control, Automation and Systems*, 8(1):149–156, 2010.

[135] Qiang Shen, Danwei Wang, Senqiang Zhu, and Eng Kee Poh. Inertia-free fault-tolerant spacecraft attitude tracking using control allocation. *Automatica*, 62:114–121, 2015.

[136] Qiang Shen, Danwei Wang, Senqiang Zhu, and Eng Kee Poh. Robust control allocation for spacecraft attitude tracking under actuator faults. *IEEE Transactions on Control Systems Technology*, 25(3):1068–1075, 2016.

[137] Y. Shtessel, L. Fridman, and A. Zinober. Higher order sliding modes. *International Journal Of Robust and NonLinear Control*, 18:381–384, 2008.

[138] Yuri B. Shtessel, Ilya A. Shkolnikov, and Arie Levant. Smooth second-order sliding modes: Missile guidance application. *Automatica*, 43(8):1470 – 1476, 2007.

[139] Viktor Solovyev, Valery Finaev, Yuri Zargaryan, Igor Shapovalov, and Denis Beloglazov. Simulation of wind effect on a quadrotor flight. 10:1535–1538, 01 2015.

[140] Zhankui Song and Kaibiao Sun. Attitude tracking control of a quad-rotor with partial loss of rotation effectiveness. *Asian Journal of Control*, 19:1812–1821, 2017.

[141] Zhankui Song and Kaibiao Sun. Attitude tracking control of a quad-rotor with partial loss of rotation effectiveness. *Asian Journal of Control*, 19(5):1812–1821, 2017.

[142] Shengqi Sun, Liang Dong, Cuijuan An, and Wenwei Liu. Fault-tolerant control design for linear systems with input constraints and actuator failures. In *Chinese Control and Decision Conference*, pages 5278–5283, 2009.

[143] Nitin Sydney, Brendan Smyth, and Derek A Paley. Dynamic control of autonomous quadrotor flight in an estimated wind field. *IEEE 52nd Annual Conference on Decision and Control (CDC)*, pages 3609–3616, 2013.

[144] D Theilliol, D Sauter, and JC Ponsart. A multiple model based approach for fault tolerant control in non-linear systems. *IFAC Proceedings Volumes*, 36(5):149–154, 2003.

[145] Nguyen Khoi Tran, Eitan Bulka, and Meyer Nahon. Quadrotor control in a wind field. *International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 320–328, 2015.

[146] V. Utkin. *Sliding Modes in Control and Optimization*. Springer, 1992.

[147] Vadim Utkin. Variable structure systems with sliding modes. *IEEE Transactions on Automatic control*, 22(2):212–222, 1977.

[148] Mien Van, Shuzhi Sam Ge, and Hongliang Ren. Robust fault-tolerant control for a class of second-order nonlinear systems using an adaptive third-order sliding mode control. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 47(2):221–228, 2016.

[149] Michel Verhaegen, Stoyan Kanev, Redouane Hallouzi, Colin Jones, Jan Maciejowski, and Hafid Smail. Fault tolerant flight control-a survey. In *Fault tolerant flight control*, pages 47–89. Springer, 2010.

[150] Ronald E Walpole, Raymond H Myers, Sharon L Myers, and Keying Ye. *Probability and statistics for engineers and scientists*, volume 5. Macmillan New York, 1993.

[151] Ban Wang and Youmin Zhang. An adaptive fault-tolerant sliding mode control allocation scheme for multirotor helicopter subject to simultaneous actuator faults. *IEEE Transactions on Industrial Electronics*, 65(5):4227–4236, 2017.

[152] Ban Wang, Youmin Zhang, Jean-Christophe Ponsart, and Didier Theilliol. Fault-tolerant adaptive control allocation for unmanned multirotor helicopter. *Elsevier, IFAC-PapersOnLine*, 50:5269–5274, 2017.

[153] C. Wang, B. Song, P. Huang, and C. Tang. Trajectory tracking control for quadrotor robot subject to payload variation and wind gust disturbance. *Journal of Intelligent & Robotic Systems*, 83(2):315–333, August 2016.

[154] Huanhuan Wang, Youmin Zhang, Yingmin Yi, Jing Xin, and Ding Liu. Nonlinear tracking control methods applied to qball-x4 quadrotor uav against actuator faults. In *IEEE Chinese Control and Decision Conference (CCDC)*, pages 3478–3483, 2016.

[155] Jia-Ying Wang, Bing Luo, Ming Zeng, and Qing-Hao Meng. A wind estimation method with an unmanned rotorcraft for environmental monitoring tasks. *Sensors*, 18(12):4504, 2018.

[156] Rong Wang, Zhi Xiong, Jianye Liu, Jianxin Xu, and Lijuan Shi. Chi-square and sprt combined fault detection for multisensor navigation. *IEEE Transactions on Aerospace and Electronic Systems*, 52(3):1352–1365, 2016.

[157] Greg Welch and Gary Bishop. An introduction to the kalman filter. 1995.

[158] SJ Wellington, JK Atkinson, and RP Sion. Sensor validation and fusion using the nadaraya-watson statistical estimator. In *Proceedings of the Fifth International Conference on Information Fusion.*, volume 1, pages 321–326, 2002.

[159] Yuhu Wu, Kaijian Hu, Xi-Ming Sun, and Yanhua Ma. Nonlinear control of quadrotor for fault tolerance: A total failure of one actuator. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.

[160] Zou Yi, Ho Yeong Khing, Chua Chin Seng, and Zhou Xiao Wei. Multi-ultrasonic sensor fusion for mobile robots. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pages 387–391, 2000.

[161] Yushu Yu and Xilun Ding. A global tracking controller for underactuated aerial vehicles: design, analysis, and experimental tests on quadrotor. *IEEE/ASME Transactions on Mechatronics*, 21(5):2499–2511, 2016.

[162] Yushu Yu and Yiqun Dong. Global fault-tolerant control of underactuated aerial vehicles with redundant actuators. *International Journal of Aerospace Engineering*, 2019, 2019.

[163] Samir Zeghlache, Hemza Mekki, Abderrahmen Bouguerra, and Ali Djerioui. Actuator fault tolerant control using adaptive rbfnn fuzzy sliding mode controller for coaxial octorotor uav. *Elsevier ISA Transactions*, 80:267–278, 2018.

[164] Samir Zeghlache, Djamel Saigaa, and Kamel Kara. Fault tolerant control based on neural network interval type-2 fuzzy sliding mode controller for octorotor uav. *Springer, Frontiers of Computer Science*, 10:657–672, 2016.

[165] Youmin Zhan and Jin Jiang. An interacting multiple-model based fault detection, diagnosis and fault-tolerant control approach. In *Proceedings of the 38th IEEE Conference on Decision and Control*, volume 4, pages 3593–3598, 1999.

[166] Youmin Zhang and Jin Jiang. Integrated active fault-tolerant control using imm approach. *IEEE Transactions on Aerospace and Electronic systems*, 37(4):1221–1235, 2001.

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

[168] Youmin Zhang, V Sivasubramaniam Suresh, Bin Jiang, and Didier Theilliol. Reconfigurable control allocation against aircraft control effector failures. In *IEEE International Conference on Control Applications (CCA)*, pages 1197–1202, 2007.