# Redundant autopilot system based on COTs open source solution

Pedro Nuno Ferreira Afonso
pedro.nuno.afonso@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

October 2021

**Abstract**

Recently the unmanned air vehicle (UAV) development has been increasing due to the large number of applications these already play at the society. Naturally, the autopilots software and hardware caught up the UAV development. The mitigation or elimination of these system failures, which can imply unexpected behaviours with losses to the users and to others, has become a concern. Developing a fault detection and tolerant system to the autopilot is the main objective of this thesis. From the reliability and redundancy concepts study, it was concluded that three autopilot units are enough to integrate the system. After analysing some commercial-of-the-shelf open source autopilots, the PX4 firmware was chosen. The estimation control library is the most complex module from the firmware. By default it uses seven independent extended Kalman filter instances at the same time, offering redundancy to the estimator level and capability for soft and hard fault detection. Each estimator health is evaluated through its innovations and innovations variance. The decision algorithm was computed on the external ring based on the final results from all these redundant mechanisms existing at PX4 firmware. The selection is computed in such a decentralized way, inside each autopilot permanently on communication with the other two. The triple redundancy system was validated running the software-in-the-loop with the gazebo simulator.

**Keywords:**Unmanned air vehicle, PX4 firmware, Extended Kalman filter, Fault detection, Fault tolerant system

## 1. Introduction

Automation in aviation, as in other domains, has increased demands on the pilot to monitor systems for possible failures. As research on vigilance has shown, this is a role for which humans are poorly suited." [1]. To support this transcription, two examples from accidents related with automation overreliance by humans are given below. The crew from China Airways Flight 006 preoccupied with an engine problem, did not notice the autopilot gradually loosing control of the plane and it plummeted 31 000 feet in 1985.

NASA Aviation Safety Reporting System (ASRS) database was examined by Mosier et al. (1994). The conclusions are: 77% of the incidents in which overreliance on automation was suspected, involved a probable vigilance failure as well as the vast majority of them occurred during cruise, when the pilot primary role was to monitor and supervise the automation. Under certain conditions, pilot overreliance on automation can make detecting failures problematic since pilots may ignore other sources of information.

Unmanned aerial systems (UAS), include an Unmanned Aerial Vehicle (UAV), a ground-based controller and a system of communications between the two. UAV came from the airplanes following an historical perspective and so they belong to the aeronautics field as well. So the framework about automation and fault reporting systems used on airplanes plays an important role because the knowledge was adapted to develop the most recent fault detection systems used by UAV. Some examples from UAV applications already used: fire detection [2], rescue operations [3], farming [4], transportation [5], law enforcement [6].

Instead of doing a new controller without guaranties it would be better than many others already on the market, the following objectives were proposed for this project:

1. Decrease the autopilot failure probability or increase its reliability;
2. Design a fault tolerant system with redundant autopilots integration;
3. Use commercial-off-the-shelf(COTS) open source autopilots;
4. Demonstration of proof of concept of the proposed solution. Validate the system running

on the software-in-the-loop with an UAV quadcopter model.

This project was made in collaboration with Ceiia, Centre of Engineering and Product Development. Ceiia has been working in partnership with the Portuguese Air Force doing various missions with the UAS-30, figure 1.3, using different payloads. It is intended to apply the designed system to the UAS-30 in the future.

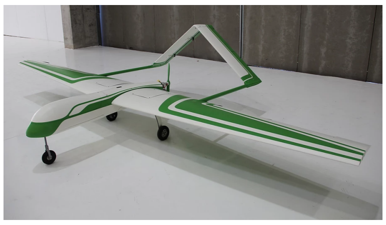

Figure 1: UAS30 from Ceiia [7]

## 2. Background

Unmanned aerial systems are divided in two subsystems, the Ground Station and the Airborne, figure 2. The communications are present in the two subsystem.

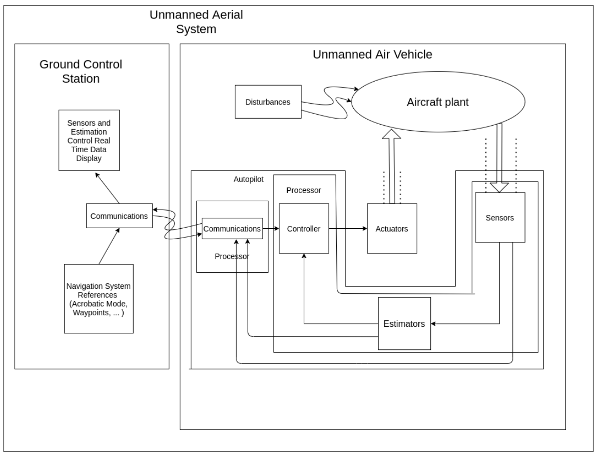

Figure 2: UAS integration

Ground control station (GCS) is where the mission is planned and the operation control center. It is the machine's interface with the human regardless the flight mode operation with more or less autonomy.

The airborne refers to the flying equipment: wings, motors, flight controllers,... The avionics components will be focused here, mainly the flight controller and sensors. The flight state estimator plays a big role on the autopilot since the states are directly used by the control unit to calculate the outputs that will be sent to the actuators. The PX4 autopilot firmware, by default, uses the Extended Kalman filter (EKF) for states estimation.

### 2.1. Fault Detection

"Fault is a deviation (of a feature) from the acceptable, standard operational condition. Error represents an incorrect status resulting from a fault, information inaccuracy. A fault tolerant system is capable to perform its function properly in the presence of one or several faults. Failure is a permanent interruption of a system ability to perform a required function." [8]

### 2.2. Reliability

Defines the system capability to keep running properly while being affected by faults. It is the fault tolerance measure from a system. The Reliability, $R(t)$, equation states:

$$R(t) = e^{-\int_0^t \lambda(t)dt} \tag{1}$$

where $\lambda$ means the probability of failure which can be considered constant to simplify ($\lambda(t) = \lambda$).

Depending on the system configuration, the System Reliability is calculated from different ways through its components Reliability. System Reliability in a serial connection (figure 3):

$$R(t) = \prod_{i=1}^{n} R_i(t) = \prod_{i=1}^{n} e^{-\int_0^t \lambda_i(t)dt} \tag{2}$$

where $R(t)$ is the system reliability, $R_i(t)$ is the reliability from each unit, $\lambda_i(t)$ is the failure probability from each unit (which can be considered constant $\lambda_i(t) = \lambda_i$) and $n$ is the total number of units.

System Reliability on a parallel connection (figure 4):

$$R(t) = 1 - \prod_{i=1}^{n}(1 - R_i(t)) \tag{3}$$

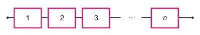

Figure 3: Failure in serial connection[8]

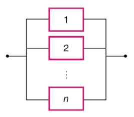

Figure 4: Failure in parallel connection [8]

Therefore to increase the system reliability, identical components should be added in parallel whenever possible creating redundancy.

### 2.3. Redundancy

"It is the process of adding identical critical components to increase the system reliability."[8]

A cross communication data link will be used for data exchange between the flight controllers. For the actuators selection there must be a voting system. The system design should have a distributed architecture ideally to achieve a greater reliability - each subsystem do the calculations and communicates with others, so the dependence on a single hardware is reduced because there is not a centralized control unit. It is demonstrated in figure 5, there is no dependency from on flight control system. To be a distributed architecture the software will be decentralized too.

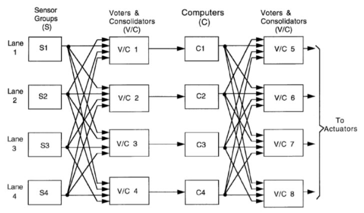

Figure 5: Quadruplex redundant flight control system. A possible distributed architecture with the voting mechanism system[8]

Adding monitoring points increases the fault detection capability and speed. Therefore one monitoring point must be at the sensors level due to their multiplicity and major failure probability. There should be redundancy at the sensors. The figure 6 shows an hard sensor failure. It could be detected and isolated using the middle value read from the three outputs when compared with a threshold value imposed by the system designer. From this point, the advantages of using four units instead of three are not enough to cover the increasing weight, size and cost, since the failure probability from the two autopilots at the same time is very low.

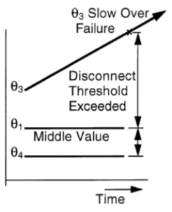

Figure 6: Triplex Slow-over failure[8]

## 3. Autopilots Overview And Autopilots Selection

The Analytical Hierarchical Process [8], AHP, was applied to choose the most suitable open source autopilot available on the market. The same method was used again to find the hardware which fits better the firmware as well as the communication protocols.

### 3.1. Firmware

Good application programming interface (API), boards compatibility to run the Firmware and the community/support were the criteria to take the PX4 as the best option, between the Paparazzi [9], LibrePilot[10] and Ardupilot[11].

### 3.2. Hardware

For the hardware identification, it was taken in consideration the affordability, the processor clock speed, RAM memory capacity, the available sensors and Support. The result from the AHP method fell on the Pixawk Cube [12]. The other considered options were the Holybro Pixhawk 4 4[13], the BeagleBone Blue [14] and the Qualcomm Flight Pro[15].

### 3.3. Communication Protocol

The decision about the communication protocol was based on the data rate, possible simultaneous transmission directions (duplex, half duplex), data protection (error checking) and complexity (number of wires and ports). Universal asynchronous reception and transmission (UART) was chosen while the inter-integrated-circuit(I2C), serial peripheral interface (SPI) were passed over.

### 3.4. Message Code Protocol

The data is codified for security and to be understood by external systems. The PX4 uses two different message protocols for external communication: Micro Air Vehicle Message Marshalling Library (MAVLink) [16]) and data distribution service real time publish subscribe (*DDS RTPS* [17]).

MAVLink is still the most used protocol between UAS by far. So there is a greater support and community which brings more safety to the developer overtakes possible problems. MAVLink is the codifying protocol on the data shared between the PX4 and the external systems.

## 4. Redundant System Design

The PX4 autopilot was divided in two subsystems: sensors module and flight control module. They will be added to the new subsystem responsible for fault tolerance: fault detection/autopilot selection module.

The point is to raise the fault detection/autopilot selection module reliability, $R_i(t)$, adding a monitoring point after each sensor used by the PX4 and to compare the values with the same sensor type from the other two autopilots. So the fault detection capability and speed increase and so the fault detection module reliability increases too. The system total reliability, see definition 2.2, states the entire system $R(t)$ increases if its belonging subsystems $R_i(t)$ increases.

### 4.1. Sensors Module

Although PX4 has already fault tolerant mechanisms to isolate and mitigate sensor failures. After the sensors data being treated, by low pass filters to eliminate the noise or other normal peaks, the process uses already redundancy at the sensors level. Throughout data comparison between sensors from the same typology, it classifies each sensor priority and selects the best source to use as observation. Meanwhile the data published from the selected sensor (the best ranked sensor from its typology) follows to the Extended Kalman filter for state estimation usually as observation, exception for the inertial measurement unit (IMU).

### 4.2. Extended Kalman Filter

The EKF is implemented throughout the introduction of the system dynamics and the observations linearized model relative to the current estimated state, refining the estimation with the sensor measurements. Those states are used by the control unit [18]. The EKF computes two steps for each iteration $k$, since it works in discrete time:

- **Prediction -** the predicted state estimate $\hat{\mathbf{x}}_{k+1}^{-}$ is computed using the state estimate $\hat{\mathbf{x}}_k^{+}$ from the previous iteration and the input $u(k)$ (the input is missing in this flowchart 7),
- **Filtering -** the $\hat{\mathbf{x}}_k^{+}$ is updated using the predicted state estimate and the current observations.

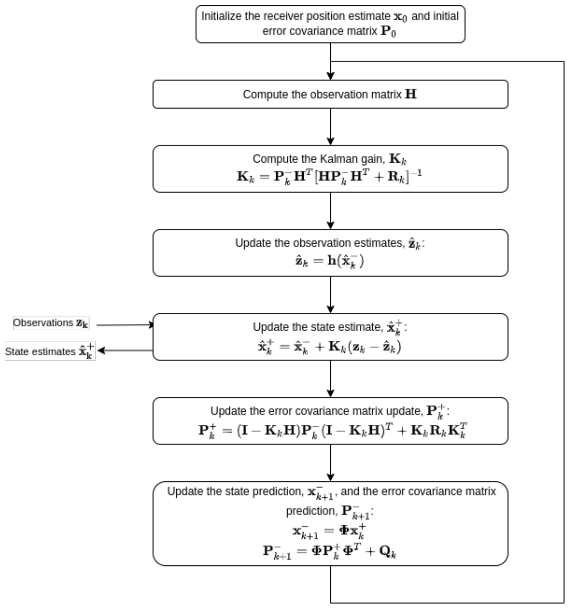
Initialize the receiver position estimate $\mathbf{x}_0$ and initial error covariance matrix $\mathbf{P}_0$
Compute the observation matrix $\mathbf{H}$
Compute the Kalman gain, $\mathbf{K}_k$
$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T [\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}_k]^{-1}$
Update the observation estimates, $\hat{\mathbf{z}}_k$:
$\hat{\mathbf{z}}_k = \mathbf{h}(\hat{\mathbf{x}}_k^-)$
Observations $\mathbf{z}_\mathbf{k}$
State estimates $\hat{\mathbf{x}}_\mathbf{k}^+$
Update the state estimate, $\hat{\mathbf{x}}_k^+$:
$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \hat{\mathbf{z}}_k)$
Update the error covariance matrix update, $\mathbf{P}_k^+$:
$\mathbf{P}_k^+ = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^-(\mathbf{I} - \mathbf{K}_k\mathbf{H})^T + \mathbf{K}_k\mathbf{R}_k\mathbf{K}_k^T$
Update the state prediction, $\mathbf{x}_{k+1}^-$, and the error covariance matrix prediction, $\mathbf{P}_{k+1}^-$:
$\mathbf{x}_{k+1}^- = \mathbf{\Phi}\mathbf{x}_k^+$
$\mathbf{P}_{k+1}^- = \mathbf{\Phi}\mathbf{P}_k^+\mathbf{\Phi}^T + \mathbf{Q}_k$

Figure 7: Flowchart of the EKF algorithm [18]

#### 4.2.1. Dynamics Model

[19]

The EKF has 24 states, equation (4), where the first states $(q_0, q_1, q_2, q_3)$ correspond to the quaternions that define the angular position (rotation) from the $XYZ$ body frame relative to the (North, East, Down) navigation inertial frame of reference and the 6 next states $V_N, V_E, V_D, P_N, P_E, P_D$ refers to the velocity and position in the navigation inertial frame of reference. The first 10 states capture the position information through a dynamic process model. The states $\Delta_{ang\_bias\_x}, \Delta_{ang\_bias\_y}, \Delta_{ang\_bias\_z}$ and $\Delta_{vel\_bias\_x}, \Delta_{vel\_bias\_y}, \Delta_{vel\_bias\_z}$ refer to the gyro delta angle bias and accelerometer delta velocity bias respectively (both gyro and accelerometer belong to the IMU. Following the matrix columns order, 4, $M_N, M_E, M_D$ and $M_X, M_Y, M_Z$ represent the magnetic field on the navigation inertial frame and on the body frame respectively. Finally the states $V_{wind_N}, V_{wind_E}$ refers to the wind velocity on the navigation inertial frame of reference.

$$\mathbf{X} = \begin{bmatrix} q_n(4) \\ V_{(NED)} \\ P_{(NED)} \\ \Delta_{ang\_bias\_(xyz)} \\ \Delta_{vel\_bias\_(xyz)} \\ M_{(NED)} \\ M_{(XYZ)} \\ V_{wind_N} \\ V_{wind_E} \end{bmatrix} \tag{4}$$

From the IMU (gyro + accelerometer), $\Delta_{ang\_meas}$ and $\Delta_{vel\_meas}$ are defined by:

$$\Delta_{ang\_meas} = \begin{bmatrix} \Delta_{ang\_meas\_x} \\ \Delta_{ang\_meas\_y} \\ \Delta_{ang\_meas\_z} \end{bmatrix} = \int_{t_k}^{t_{k+1}} \vec{w} dt \tag{5}$$

$$\Delta_{vel\_meas} = \begin{bmatrix} \Delta_{vel\_meas\_x} \\ \Delta_{vel\_meas\_y} \\ \Delta_{vel\_meas\_z} \end{bmatrix} = \int_{t_k}^{t_{k+1}} \vec{a} dt \tag{6}$$

The truth delta angles $\Delta_{ang\_truth}$ are calculated from the IMU and delta angle bias states $\Delta_{ang\_bias}$:

$$\Delta_{ang\_bias\_(x,y,z)} = \begin{bmatrix} \Delta_{ang\_bias\_x} \\ \Delta_{ang\_bias\_y} \\ \Delta_{ang\_bias\_z} \end{bmatrix} \tag{7}$$

$$\Delta_{vel\_bias\_(x,y,z))} = \begin{bmatrix} \Delta_{vel\_bias\_x} \\ \Delta_{vel\_bias\_y} \\ \Delta_{vel\_bias\_z} \end{bmatrix} \tag{8}$$

$$\Delta_{ang\_truth} = \Delta_{ang\_meas} - \Delta_{ang\_bias\_(x,y,z)} \tag{9}$$

$$\Delta_{vel\_truth} = \Delta_{vel\_meas} - \Delta_{vel\_bias\_(x,y,z)} \tag{10}$$

The quaternion $\Delta_{quat}$ defines the rotation from frame $k$ to $k+1$. The truth delta angle, $\Delta_{ang\_truth}$, is used to calculate $\Delta_{quat}$ using a small angle approximation:

$$\Delta_{quat} = \begin{bmatrix} \Delta q_0 \\ \Delta q_1 \\ \Delta q_2 \\ \Delta q_3 \end{bmatrix} = \begin{bmatrix} 1 \\ \frac{\Delta_{ang\_truth\_x}}{2} \\ \frac{\Delta_{ang\_truth\_y}}{2} \\ \frac{\Delta_{ang\_truth\_z}}{2} \end{bmatrix} \tag{11}$$

To rotate the quaternion state forward from frame $k$ to $k+1$ the $\Delta_{quat}$ is used in the quaternion product rule:

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}_{k+1} = \begin{bmatrix} q_0\Delta q_0 - q_1\Delta q_1 - q_2\Delta q_2 - q_3\Delta q_3 \\ q_0\Delta q_1 q_0 + q_1\Delta q_0 + q_2\Delta q_3 - q_3\Delta q_2 \\ q_0\Delta q_2 + q_2\Delta q_0 - q_1\Delta q_3 - q_3\Delta q_1 \\ q_0\Delta q_3 + q_3\Delta q_0 + q_1\Delta q_2 - q_2\Delta q_1 \end{bmatrix} \tag{12}$$

The velocity states from frame $k$ to $k+1$ are calculated by the truth delta velocity vector, $\Delta_{vel\_truth}$, rotated from the body frame to the Inertial Navigation frame, $[T]_B^N$, and subtracting gravity:

$$\begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix}_{k+1} = \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix}_k + [T]_B^N . \Delta_{vel\_truth} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} . \Delta t \tag{13}$$

The position estimates are updated:

$$\begin{bmatrix} P_N \\ P_E \\ P_D \end{bmatrix}_{k+1} = \begin{bmatrix} P_N \\ P_E \\ P_D \end{bmatrix}_k + \begin{bmatrix} V_N \\ V_E \\ V_D \end{bmatrix}_k . \Delta t \tag{14}$$

The remaining states (IMU sensor bias, magnetic field and wind) use a static process model. They do not change from $k$ to $k+1$ frame. The accelerometer and gyroscope raw data are used as input ($u(k)$) to the EKF and not as observation like the remaining sensors.

#### 4.2.2. Observations Model

[19]

The EKF solves this problem by linearizing the observations model. Starting with the observations equation [18]

$$\mathbf{z_k} = \mathbf{h}\left[\mathbf{x}(t_k)\right] + \mathbf{v}_k \tag{15}$$

where $\mathbf{z}_k$ is the sensor measurement vector, $\mathbf{v}_k$ the observations noise and $\mathbf{h}\left[\mathbf{x}(t_k)\right]$ is the relation between states and the observations. Some examples are given below.

The GPS position, barometer height and GPS velocity are direct observations from states, so the observation mode is trivial.

It is assumed the magnetometer to be aligned with the body frame and experiences a magnetic field vector which is the sum from the navigation inertial frame rotted into body frame ($[T]_B^N$) and a body frame bias:

$$\begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix}_{meas} = [T]_B^N . \begin{bmatrix} M_N \\ M_E \\ M_D \end{bmatrix} + \begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix}_{bias} \tag{16}$$

In order to linearize the observations equations, the Jacobian matrix of $h(x)$ is obtained. This matrix is called the observation matrix, $\mathbf{H}$, and is given by

$$\mathbf{H}_k = \left[\frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}\right]_{\mathbf{x}=\mathbf{x}_{\mathbf{k-1}}^{+}}, \tag{17}$$

The last observation model parameter is the observation noise covariance matrix, $\mathbf{R}_k$, which is a $n \times n$ diagonal matrix given by

$$\mathbf{R}_k = \begin{bmatrix} \sigma_1^2 & & \mathbf{0} \\ & \sigma_2^2 & & \\ & & \ddots & \\ \mathbf{0} & & & \sigma_n^2 \end{bmatrix} \tag{18}$$

Having obtained all the parameters necessary to the computation of the EKF, the 24 states can be estimated.

### 4.3. Sensor Fusion

PX4 can change the source of observations while running the EKF. For a better understanding, a flowchart was designed (see figure 8) relative to the height source. The barometer is the primary source for height determination but the GPS can be used as height observation instead.

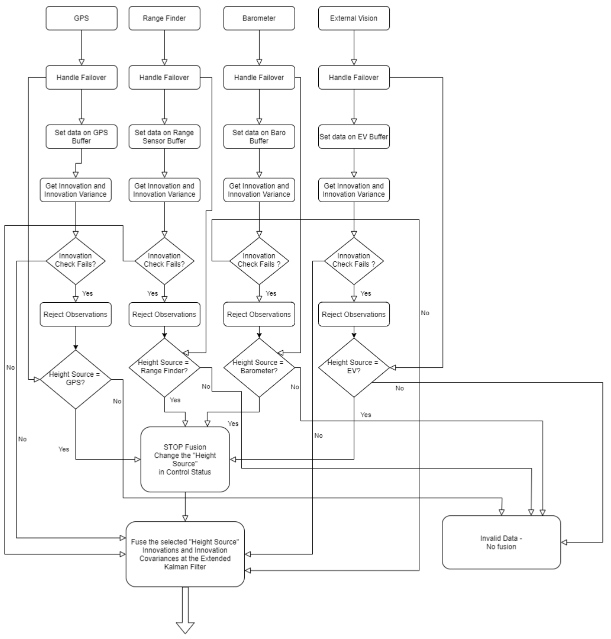
GPS
Range Finder
Barometer
External Vision
Handle Failover
Set data on GPS Buffer
Set data on Range Sensor Buffer
Set data on Baro Buffer
Set data on EV Buffer
Get Innovation and Innovation Variance
Innovation Check Fails?
Yes
No
Reject Observations
Height Source = GPS?
Height Source = Range Finder?
Height Source = Barometer?
Height Source = EV?
STOP Fusion Change the "Height Source" in Control Status
Fuse the selected "Height Source" Innovations and Innovation Covariances at the Extended Kalman Filter
Invalid Data - No fusion

Figure 8: Sensor fusion example with the height source.

In case of a sensor failure, it is replaced or corrected by another sensor source. This process is called sensor fusion and it is not related with the sensors modules which replaces sensor values with other from the same typology. Also the fault detection criteria is different from the sensors module. The observation measurements are evaluated by their innovations or residuals, the difference between the sensor measurement (observation) and the predicted sensor value computed through the predicted states, $z(k) - \hat{z_k} = z(k) - h(\hat{x}^{-}(k))$.

### 4.4. Multi-EKF

The most recent fault detection method from PX4 uses 7 (by default) EKF running on parallel. Each EKF is attached to one accelerometer, one gyroscope and one magnetometer and it cannot change any of these sensors but the same sensor can be attached to more than one EKF, figure 9.

The multi-EKF allows soft failures detection beyond the hard faults which were detected until now by the sensors modules. The EKF selector module judges the EKF health through the innovations evaluation and the Bitmask flags correspondent to the filter internal faults (numerical errors).

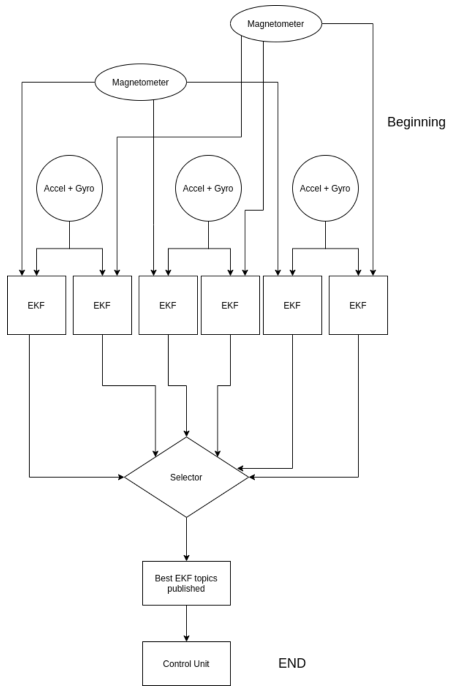

Figure 9: PX4 Multi-EKF mode flowchart.

The commander module from PX4 relates the chosen/best EKF published *estimator status* topic data with time periods to conclude if the autopilot has detected an hard failure and if it has to trigger the failsafe mode. For the case of the *condition_global_position_valid*, *condition_local_position_valid* or *condition_local_velocity_valid* the interested data from the topics refers to the innovations exceeded limits, standard deviation errors from horizontal (*eph*) / vertical (*epv*) positions and velocities (*epv*) exceeded limits, dead reckoning time exceeded or publishing timeouts. If the navigation keeps failing for more than 2 seconds, the commander module changes the state to "navigation failure". That decision will change three flags in the "vehicle_status" topic published by this module: *condition_local_position_valid*, *condition_local_velocity_valid* and *condition_global_position_valid*. A fault tree using the example from the global and local position flag is represented in the figure 10.

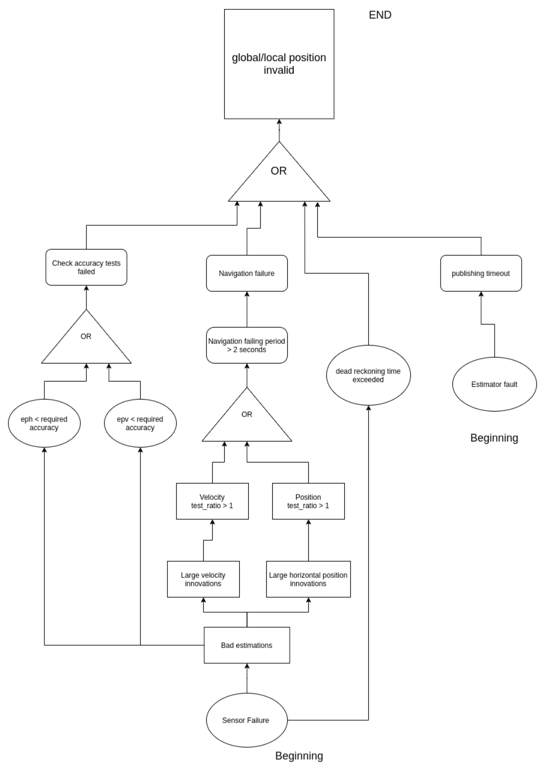

Figure 10: Global or local position validation fault tree used by the commander module.

More three boolean flags are determined by the commander module: *condition_angular_velocity_valid*, *condition_attitude_valid* and *condition_local_altitude_valid*. The angular velocity, the attitude and the local altitude are verified due to their importance to the navigation. The commander module checks if those variables are updated (less than 1 second from the last publish) as well as if their values are finite. So the 6 error flags were considered relevant for fault detection about one PX4 unit: *condition_local_position_valid*, *condition_local_velocity_valid*, *condition_global_position_valid*, *condition_angular_velocity_valid*, *condition_attitude_valid* and *condition_local_altitude_valid*. The Commander module publishes other flags suitable to evaluate crucial subsystems for the autopilot operations: battery_healthy, *condition_system_sensors_initialized*, *failsafe* and *data_link_lost*.

The triple redundant system provides the possibility to detect and mitigate faults by comparing the system or subsystem states. When two of the three systems agree about its state, the third one which disagrees is probably failing. Because the chances of two systems taking the wrong decision at the same time are much lower. This logic will guide the system fault detection. Therefore the mentioned six flags relative to the estimator/sensor health plus the other 4 flags, relative to other autopilot subsystems, were chosen to be monitored and compared with the respective flags from the others redundant autopilots.

## 4.5. Algorithm Implementation

To exemplify the fault detection method implemented with one flag, the figure 11 is shown. It refers to the failsafe variable comparison between autopilots. At this time a 'FALSE' Boolean flag state means there is no error. The failsafe is triggered when an hard failure is detected by the PX4 itself. Like any error flags, when the fault is detected, the respective error counter from the autopilot is increased.

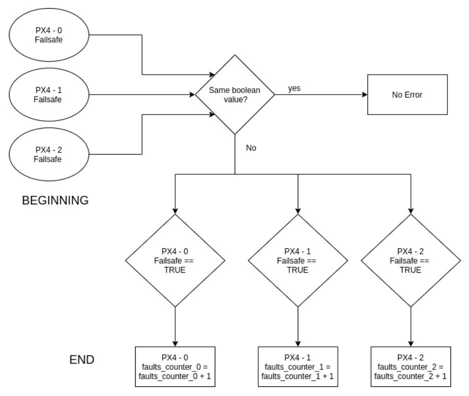

Figure 11: Comparison diagram of the redundant Boolean failsafe flag and failure counter for each PX4 instance.

The comparisons are done inside each PX4, since it is a decentralized and distributed system. An extra module was added to the firmware code, the *redundancy_manager.cpp*, written in C++ programming language. Also the voting process is decentralized, each PX4 has autonomy and computes its own vote on the best autopilot based on the data shared between them. All the autopilots vote on the respective autopilot with the lowest error/faults counter computed by themselves based on the information they received about the 10 error flags states from the other 2 PX4. The error counters values are reflected on the autopilot priority (another created variable) unless the communications between the PX4 stand as the error origin. The autopilot which does not communicate for more than 3 seconds, has its priority immediately changed and becomes the last choice regardless the other two PX4 error counters values.

The autopilot priorities values are reflected on the voted autopilot by each instance. When the autopilots computed priorities are the same (equal error counters values), each autopilot votes on the autopilot numbered with the smallest value.

The first autopilot (PX4 instance 0) computation algorithm and the external selection using the other two autopilots decisions can be visualized on diagram 12.

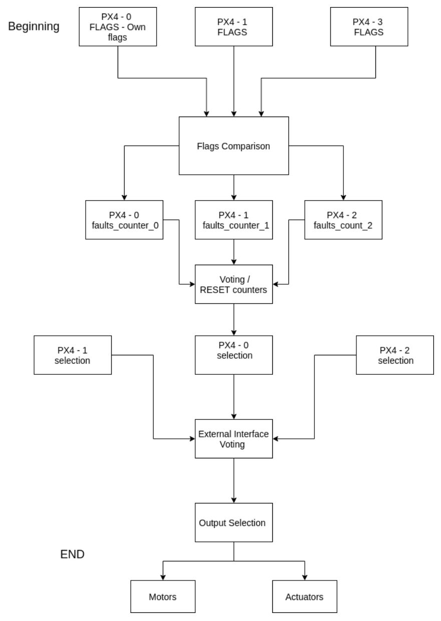

Figure 12: Decentralized voting algorithm for the first PX4 instance.

## 5. Experimental Results

In this chapter, the experimental methods and results are summarily described.

## 5.1. Experimental Setup

Sensor faults were artificially injected in the autopilot for time periods through the system failure injection module, on the PX4 console. Those faults consequences are observed through logged internal topics as well as the system reaction and fault mitigation in order to validate the system.

The implemented algorithm was run on Software-in-the-loop, SITL. A quadcopter model, "iris" was tested. Three different programs were implemented to allow the autopilot selection. Beyond the fault detection module added to the PX4 firmware, two interface software were designed to manage the communications between the autopilot, simulator (Gazebo) and ground control station (QGroundControl). Both interface software forward data just from the selected PX4 instance, accordingly to the voting system, to prevent communications conflicts on the simulator and GCS ports. So the redundancy manager programs were designed to be the the only one communication contact point from both GCS software and simulator and to make the three PX4 instances to be invisible to them both. Also it is desired the user experience to be transparent (one vehicle - one autopilot) although the developer knows that three autopilot instances are running.

The communications were done polling Transmission control protocol (TCP) sockets in multiple threads written in C programming language. The simulator redundancy manager architecture is shown in figure 13. A similar redundancy manager was implemented for the QGC.

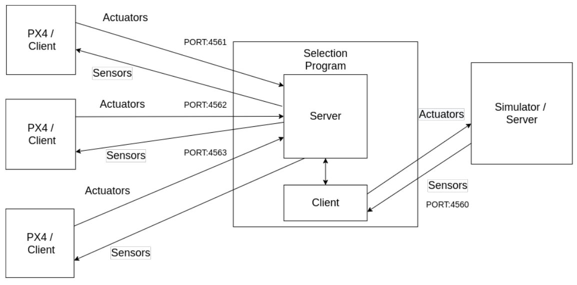

Figure 13: Communication protocol between PX4, Simulator Redundancy Manager and Simulator.

The mission takes place in the PX4 default location (Zurich, Switzerland). It was chosen a typical survey mission where the UAV, after the takeoff, follows the Waypoints number order to up and down throughout vertical lines on the map, see figure 14.

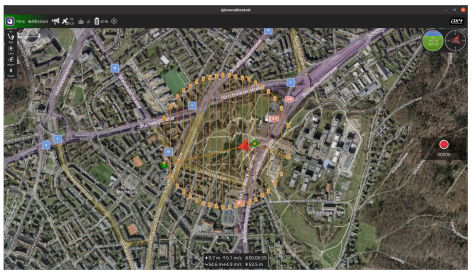

Figure 14: Mission plan view from QgroundControl.

For the flight log analysis were chosen the "Flight Review" [20] software for the general topics and its graphs visualization and the "Plotjuggler" [21] software for specific topics inspection like the added messages from the redundancy algorithm or other messages containing flags considered important for this project.

## 5.2. GPS Failure

A GPS failure was injected on the first autopilot 37 seconds after the simulation start. The dead reckoning was triggered around the 38 seconds while the standard deviation errors had increased in the PX4 instance 0, see figure 15.

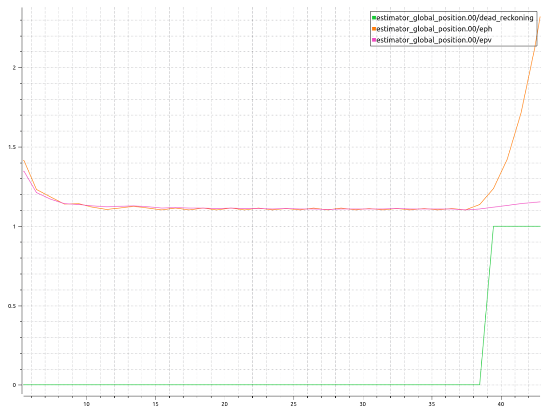

Figure 15: PX4 instance 0 - Dead reckoning, horizontal and vertical standard deviation errors.

The instance 1 actuator controls, figure 5.2 begin with instability while the instance 0, figure 17, is the only autopilot controlling the UAV, so the other instances are receiving the feedback from the other autopilot control. When the instance 0 initiates the landing phase, the other two instances show opposite values to the actuators since they are not aware of the triggered failsafe and they will do an effort to keep the UAV on the previous route to reach the next waypoint. Fifty seconds after the simulation start, the actuator controls from the instance 1 stabilize since it has taken over the UAV and it receives now the correct feed backs from the vehicle answers as well as it turns back to the right position to accomplish the mission.

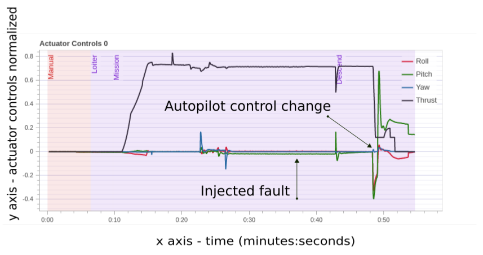

Figure 16: PX4 instance 0 actuator controls (roll, pitch, yaw and thrust).

## 5.3. Magnetometer Failure

Two stuck magnetometers connected to the first autopilot were simulated, two minutes and 27 seconds since the simulation start, throughout the failure injection command leaving the PX4 instance 0 without any working magnetometer available, see figure

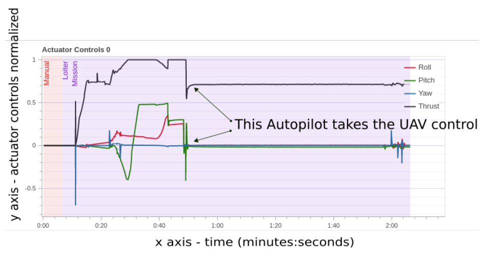

Figure 17: PX4 instance 1 actuator controls (roll, pitch, yaw and thrust).

18. At 2 minutes and 27 seconds after the simulation start, raw magnetic field strength values had become stuck.

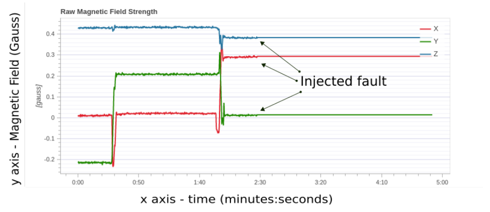

Figure 18: Magnetic field strength measured by the PX4 instance 0 magnetometers.

The GPS and Magnetometer innovations test ratios from instance 0, figure 19, are greater than 1 for a time period of 20 seconds approximately and 30 seconds after the magnetometer fault injection. The GPS (not just the magnetometer) bad estimations, reflected on its respective innovations, were provoked by the magnetometer failure too. So the fault detection is triggered by the *global_position_valid* and *local_position_valid* flag like it was demonstrated in the fault tree, figure 10. While the PX4 instance 1 keeps good estimations with peaks around 0.005, figure 20.

The difference between the pitch, yaw and thrust actuator control values from instance 0 (figure 21) to the instance 1 (figure 22) until the third minute are explained by a different first waypoint on the uploaded mission by the QGC, nothing related with faults.

The landing maneuver caused by the failsafe mechanism is visible on the first autopilot, figure 21, at 3 minutes and 10 seconds decreasing the thrust. The landing phase also confuses the second autopilot, figure 22, which tries to contradict the maneuver not commanded by itself. After taking over the vehicle control, the thrust returns to same initial value from the instance 0 as well as the roll, pitch and yaw which turns 0. Now it is the instance 0 turn to try contradicting the maneuver commanded from the instance 1.

## 6. Conclusions

The main objective for this project was to design a redundant autopilot system to increase its reliability when compared with a single autopilot. The reliability and redundancy concepts were applied to choose the system architecture and the number of units in order to get the best results considering the trade-off with complexity and weight. In the end, a triple redundancy system with distributed architecture was chosen as the best approach.

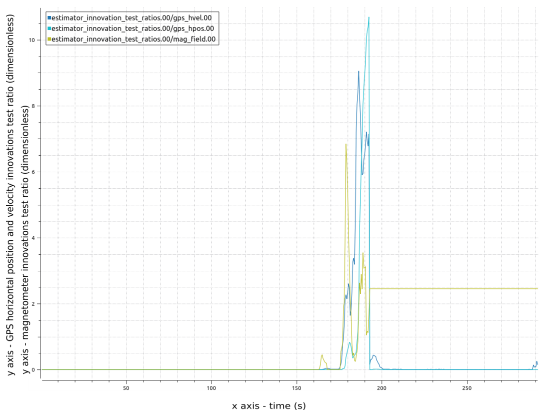

Figure 19: PX4 instance 0 innovations test ratios.

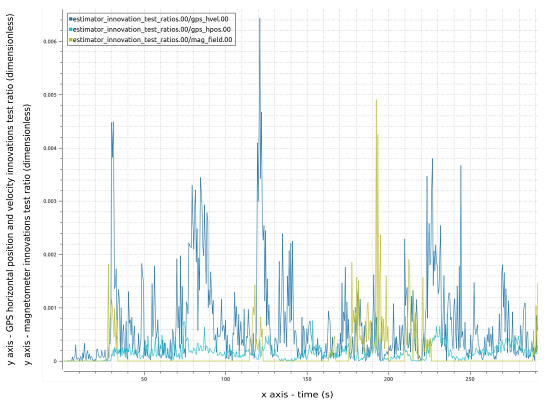

Figure 20: PX4 instance 1 innovations test ratios.

The PX4 was considered the best autopilot to fit on this triple redundancy system. MAVLink protocol was chosen for the autopilots external communications. The system design was implemented reaching a solution which benefits from the fault detection systems existing at PX4 such as the sensor fusion and the multi-EKF.

The simulation results obtained from the designed system of this project when GPS, magnetometer and PX4 communication failures occurred are summarized on the table 1. It was filled with the time periods for fault detection followed by the autopilot substitution. The magnetometer failure took more 37 seconds to be detected than the GPS failure meaning the PX4 relies more on GPS than on Magnetometer. The PX4 was able to control the vehicle properly for a longer period without magnetometers. The PX4 communication failure is the fastest to be detected. However 7 seconds is still a lot of time in flight for the actuators to be locked

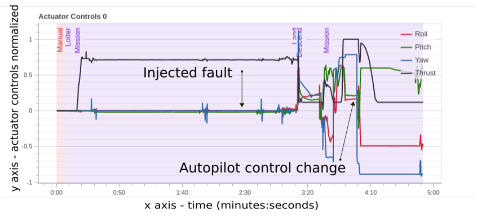

Figure 21: PX4 instance 0 actuator controls (roll, pitch, yaw and thrust).

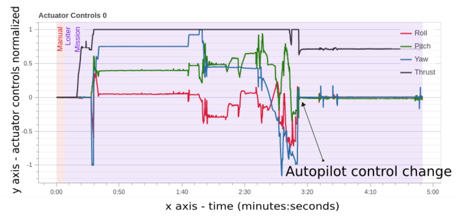

Figure 22: PX4 instance 1 actuator controls (roll, pitch, yaw and thrust).

by an autopilot without heartbeat. In the case of GPS and magnetometer failures, the PX4 failsafe mechanisms were triggered before the autopilot actuators control changed. Because two seconds are needed, at least, to update the error counters and the correspondent vote. To avoid the trajectory deviation, the fault detection speed of the system has to increase and so the time between the iterations (to check the error flags and to communicate with other autopilots) is decreased. Although the heavy computational communications from the interface programs limits the available computer resources. Just running the software on a powerful processor allows to reduce the time between iterations in real time without loosing program threads.

This project, while achieving the objectives set, has a lot of potential for further advancements. System validation for accelerometer and gyroscope faults or for the QGC communication fault with the correspondent error flag *data_link_lost* already implemented on the algorithm. Hardware-in-the-loop validation and in the UAS-30 real flight are the last steps.

Table 1: Fault tolerance time performance of the system.

| Failure Typ. | GPS | Mag. | Comm. |
|---|---|---|---|
| FD Time Period (s) | 6 | 43 | 4 |
| PX4 Substitution (s) | 10 | 46 | 7 |

## Acknowledgements

The author would like to thank his supervisors, Prof. Alexandra Moutinho and Eng. Renato Machado , for all their support during the project; Centre of Engineering and Product Development (Ceiia) for providing all the available resources to design and test the software.

## References

[1] Robert Molloy and Raja Parasuraman. Monitoring an automated system for a single failure: Vigilance and task complexity effects. *Human Factors*, June 1996.

[2] C. Sathiya Kumar S. Sudhakar, V. Vijayakumar. Unmanned aerial vehicle (uav) based forest fire detection and monitoring for reducing false alarms in forest-fires. *Pre-proof*, 2019.

[3] Lukasz Kuziora Marzena Pólkaa, Szymon Ptaka. The use of uav's for search and rescue operations. In *TRANSCOM 2017: International scientific conference on sustainable, modern and safe transport*. Elsevier Ltd, 2017.

[4] Panagiotis Sarigiannidis Panagiotis Radoglou-Grammatikis. A compilation of uav applications for precision agriculture. *Pre-proof*, 2020.

[5] John C. Golias Emmanouil N. Barmpounakis, Eleni I. Vlahogianni. Unmanned aerial systems for transportation engineering: 4 current practice and future challenges. *International Journal of Transportation Science and Technology*, 2017.

[6] DRONEFlY. Police drone infographic. `https://www.dronefly.com/police-drone-infographic`. Accessed on: 28-11-2019.

[7] Ceiia. Uas-30. `https://www.ceiia.com/single-post/2017/09/15/SEGUNDA-GERA%5C%C3%5C%87%5C%C3%5C%830-DO-UAS30-DO-CEIIA-J%5C%C3%5C%81-TEM-ASAS-PARA-VOAR`. Accessed on: 28-11-2019.

[8] Bertinho D'Andrade da Costa and Agostinho da Fonseca. *Support text slides from Integrated Avionic Systems course*. Instituto Superior Técnico, Universidade de Lisboa, 2017/2018.

[9] *Paparazzi Autopilot Developer Guide*. `https://wiki.paparazziuav.org/wiki/Developer_Guide`.

[10] *LibrePilot Developer Guide*. `https://librepilot.atlassian.net/wiki/spaces/LPDOC/pages/2818105/Welcome`.

[11] *ArduPilot Developer Guide*. `https://ardupilot.org/dev/index.html`.

[12] Cube autopilot technical report. Technical report, Pixhawk, `https://ardupilot.org/copter/docs/common-thecube-overview.html`.

[13] Holybro pixhawk autopilot technical report. Technical report, Pixhawk, `https://github.com/ArduPilot/ardupilot/blob/master/libraries/AP_HAL_ChibiOS/hwdef/Pixhawk4/README.md`.

[14] Beagle bone blue technical report. Technical report, Beagle Boards, `https://beagleboard.org/blue`.

[15] Qualcom technical report. Technical report, Lantronix, `https://www.intrinsyc.com/qualcomm-flight-pro-development-kit/`.

[16] *Micro Air Vehicle Message Marshalling Library Aviation, Volume I - Radio Navigation Aids*. `https://mavlink.io/en/guide/define_xml_element.html`.

[17] *Real Time Publishers and Subscribers - ROS2 Interface*. `http://dev.px4.io/v1.9.0/en/middleware/micrortps.html`.

[18] Fernando Duarte Nunes. *Sistemas de Controlo de Tráfego - Apontamentos das Aulas*. Instituto Superior Técnico, Universidade de Lisboa, 2017.

[19] Estimation Control Library Dynamics Model Github page. `https://github.com/PX4/PX4-ECL/blob/master/EKF/documentation/Process%20and%20Observation%20Models.pdf`.

[20] PX4. Flight review. `https://review.px4.io/`.

[21] Marxlp. Px4 flight log visual analysis tool. `https://plotjuggler.io/#one`.