POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering

UAVs system for autonomous indoor flight and remote controlled

Supervisors

Prof. Alessandro RIZZO

Dr. Stefano PRIMATESTA

Dr. Orlando TOVAR ORDOÑEZ

Candidate

Christian BONOTTO

December 2022

Abstract

Nowadays, the industrial environment has strong dependencies on robotic solutions. In fact, these represent a good investment for both major companies and small starts ups. The implementation of robotic and unmanned systems can improve both the efficiency and security of the production. Moreover, in critical and threatening conditions, autonomous vehicles are necessary to limit risks for human operators. Among UVS (Unmanned Vehicle System), drones occupy an important and fascinating spot. The greatest feature of this class of vehicle is that, ideally, they can be used in many different situations. In general, drones can provide autonomous flight, both indoor and outdoor, and a great agility also in the tightest places. In this master thesis the autonomous behaviour of the FIXIT's drone will be considered. The FIXIT one, is a project carried on by CIM4.0, the Competence Centre for Industry 4.0 of Torino; such drone consists of an UAV (Unmanned Aerial Vehicle) that can fly in an industrial environment and eventually dock on a moving rover. The result has the aim to assist human operators in specific missions, collecting and elaborating data. The drone should be able to provide stable flight in GNSS (Global Navigation Satellite System) denied environment exploiting the UWB (Ultra Wide Band) wireless technology and specific sensors. Moreover, an obstacle avoidance algorithm is implemented. The flight controller's program is based on Ardupilot and run on the PixHawk board 2.4.8. In this study, the focus is on the sensors needed in order to obtain stability for different conditions and on the development of the flight controller code to provide personalized actions.

Table of Contents

List of Figures V

Acronyms VIII

1 Introduction and state of the art 1
- 1.1 Localization problem generalities 1
- 1.2 Filtering techniques 4
  - 1.2.1 Kalman Filters 4
  - 1.2.2 Particle Filter 7
  - 1.2.3 Complementary Filters 7
  - 1.2.4 Simultaneous Localization and Mapping (SLAM) 8
- 1.3 Pre-processing operations 9
  - 1.3.1 Sensor calibration 9
- 1.4 Flight configurations 10
  - 1.4.1 Outdoor flight 10
  - 1.4.2 Indoor flight 11
- 1.5 Autopilot 12

2 Hardware description 15
- 2.1 Sensor list 15
  - 2.1.1 Intel RealSense d435i 15
  - 2.1.2 Inertial Measurement Unit 16
  - 2.1.3 Magnetometer 16
  - 2.1.4 Global Positioning System 17
  - 2.1.5 Rangefinders 18
  - 2.1.6 Ultrasonic Sensor 19
  - 2.1.7 Optical Flow sensor 20
  - 2.1.8 Additional Hardware 21
- 2.2 Drone configuration 21
  - 2.2.1 Autopilot boards 23

3 Learning the code languages 25
3.1 ArduPilot . . . 25
3.2 Structure of the code . . . 25
3.2.1 High level . . . 25
3.2.2 Low level . . . 26
3.3 Arduino . . . 28
3.3.1 Hardware selection . . . 28
3.3.2 Arduino IDE . . . 29
3.3.3 Processing . . . 30
3.4 Matlab . . . 30

4 Sensors data implementation 31
4.1 Raw Data . . . 31
4.2 Pozyx UWB . . . 32
4.2.1 UWB implementation on the UAV . . . 33
4.3 Pozyx magnetometer . . . 34
4.4 Lidar vs Barometer . . . 37
4.5 RM3100 vs HMC5883L . . . 39

5 First set-up and performance analysis 40
5.1 Arduino setup . . . 40
5.2 Tests and problems analysis . . . 41
5.2.1 Positioning . . . 41
5.2.2 Heading . . . 45

6 Methodology and implementation 46
6.1 Origin drift problem . . . 46
6.2 Heading problem . . . 48
6.2.1 Magnetic Calibration . . . 49
6.2.2 Double Tag Heading . . . 49
6.2.3 Pozyx AHRS . . . 50
6.2.4 Lane Switching . . . 53
6.2.5 Filter Customization . . . 54
6.3 Final set up and experimental tests . . . 56
6.3.1 Additive mission - Quadcopter . . . 57
6.3.2 Digital mission - Quadcopter . . . 63
6.3.3 Additive mission - Octocopter . . . 67

7 Conclusions 72

Bibliography 74

List of Figures

2.1 Intel RealSense camera 15
2.2 Magnetometer RM3100 16
2.3 Taoglass ZED-F9P 17
2.4 TFmini Lidar 18
2.5 HC SR-04 sensor 19
2.6 PX4 Optical Flow sensor 20
2.7 Buzzer and safety switch 21
2.8 Pixhawk 2.4.8 23
2.9 Cube Orange 24

3.1 High level architecture of the code 26
3.2 The Arduino IDE environment 29

4.1 Pozyx's anchors displacement 32
4.2 Autopilot connection to the UNO board 33
4.3 The paper box used for calibration 34
4.4 Not-calibrated magnetometer measurements 35
4.5 Calibration Process 35
4.6 Calibrated magnetometer measurements 36
4.7 Altitude sensors comparison 37
4.8 Altitude evaluation 38
4.9 Magnetometers comparison 39

5.1 System configuration: each yellow pin represents the relative anchor position on the map 41
5.2 Localization tests 42
5.3 Shift errors 43
5.4 Motors magnetic disturbance 44

6.1 Shift errors solved 47
6.2 Lane switching between two autopilot cores 54
6.3 The two final UAVs 56

6.4 Path desired (red) and achieved (blue) 57
6.5 North-axis positioning difference 58
6.6 East-axis positioning difference 58
6.7 North-axis positioning error 59
6.8 East-axis positioning error 59
6.9 North-axis difference between Pozyx's estimate and EKF's one . . . 60
6.10 East-axis difference between Pozyx's estimate and EKF's one 60
6.11 Yaw innovations 62
6.12 Yaw EKF's test-ratio 62
6.13 Path desired (red) and achieved (blue) 63
6.14 North-axis positioning difference 64
6.15 East-axis positioning difference 64
6.16 North-axis positioning error 65
6.17 East-axis positioning error 65
6.18 Yaw innovations 66
6.19 Yaw EKF's test-ratio 66
6.20 Path desired (black) and achieved (red) 67
6.21 Yaw innovations 69
6.22 Yaw EKF's test-ratio 69
6.23 North-axis positioning difference 70
6.24 East-axis positioning difference 70
6.25 North-axis positioning error 71
6.26 East-axis positioning error 71

Acronyms

SITL
Software In The Loop

UAV
Unmanned Aerial Vehicle

GCS
Ground Control Station

CC
Companion Computer

MP
Mission Planner

QGC
QGround Control

KF
Kalman Filter

EKF
Extended Kalman Filter

UWB
Ultra Wide Band

AHRS
Attitude and Heading Reference System

Chapter 1

Introduction and state of the art

Autonomous movement in the robotic field is an important achievement. A robot, which is as much as possible independent during its mission, has many advantages. Multiple companies are moving through a gradual automatization of working places, and unmanned robots are a fundamental piece in this sense. However, for what drones are concerned, they are complex to manage and control because of their aerial nature. In last years different strategies have been tested, bringing good results. In general, making a vehicle able to move with autonomy means providing a localization system and implementing sensors to achieve obstacle avoidance behaviour in different conditions. Therefore, there is not the perfect configuration, but the more appropriate for the case of interest. In fact, in this project two different situations are considered: the indoor movement and the outdoor one. In each environment a specific configuration is implemented.

1.1 Localization problem generalities

The first problem is the localization one, which means estimate attitude and heading of the vehicle in the space while it is moving. This problem can be solved using sensors and localization algorithms. The system that includes different sensors for the evaluation of the orientation of a vehicle is referred to as Attitude Heading Reference System (AHRS). Many autonomous vehicles, such as submarines or UAVs use this technology. Localization is a term that must be considered in relation to the environment we are working in. The attitude and heading of a vehicle are computed with respect to a reference. Thus, two coordinates' systems, at least, are described: the first one is the world reference frame, the second one is the vehicle frame (or body frame). Normally, any transformation between frames can be described as a sequence of rotations and translations in the space. For what the orientation is concerned, it is possible to consider multiple rotation around the axes of the frame. The more general transformation is the composition of three consequent movements; this sequence is described as the rotation of an angle $\psi \in (-180{,}180]$ around the z axis followed by the rotation of an angle $\theta \in [-90{,}90]$ around the y axis and the consequent rotation around the x axis of an angle $\phi \in [-180{,}180)$. These can be described using the specific matrices, as follows.

$$M_z = \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{1.1}$$

$$M_y = \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \tag{1.2}$$

$$M_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix} \tag{1.3}$$

The total transformation matrix, also called Direction Cosine Matrix (DCM), can be obtained by multiplying in the specified order the previous matrices. Such matrix defines the different orientation between the two frames. This orientation can be explained also using Euler Angles. However, Euler angles are not always the best choice to compute orientation with sensors. For example, consider the gyroscope. This sensor works returning the angular velocities ω_i of an object around the frame axes, and it is used to understand the orientation of the body. The relation between the derivatives of the Euler angles and the angular velocities is the next one:

$$\begin{pmatrix} \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \\ 0 & \cos(\phi) & -\sin(\phi) \\ 1 & \tan(\theta)\sin(\phi) & \tan(\theta)\cos(\phi) \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{1.4}$$

At the denominator of some matrix elements there is the $\cos\theta$; if such angle is equal to ± 90, the results is a division by zero. This situation is known as gimbal lock. In this situation, two axes of the considered object are in parallel, “locking" the system into rotation in a degenerate two-dimensional space, and the body loses one degree of freedom. Thus, another system for attitude representation will be used. Such solution consists in quaternions. They are an equivalent way to represents rotations but avoiding singularities. The generic quaternion ca be written as $\boldsymbol{q}$ and defined as the combination of real numbers (q_0, q_1, q_2, q_3) and hypercomplex numbers (i, j, k). In particular, given $\mathbf{q} = q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$,

$q_r = q_0$ is the real part, while $\mathbf{q}_v = q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k}$ is the vectorial part. The general transformation matrix ca be defined with quaternions as in equation 1.5.

$$M = \begin{pmatrix} {q_0}^2 + {q_1}^2 - {q_2}^2 + {q_3}^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & {q_0}^2 - {q_1}^2 + {q_2}^2 - {q_3}^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_0q_2 - q_1q_3) & 2(q_2q_3 - q_0q_1) & {q_0}^2 - {q_1}^2 - {q_2}^2 + {q_3}^2 \end{pmatrix} \tag{1.5}$$

Thus, quaternions can be used to define the position and attitude of the body. It will be considered later as state vector using filters.

Positioning information are provided by sensors. Some of them will be described more in detail later, during the explanation of the hardware configuration for tests. In general, most common type of sensors used for AHRS are inertial sensors like gyroscopes, accelerometers, and magnetometers. However, the presence of disturbances can bring errors to the measurements. Accelerometers sense the gravitational field; thus, vibrations will affect the calculations. At the same time, magnetometers are affected by ferromagnetic elements that can interfere with the magnetic field measurements. For what gyroscopes are concerned, it is possible to compute the orientation of the vehicle using a discrete-time integrator applied to the measured angular velocities. But the zero-bias integration, in presence of uncertainties, will lead to an accumulating error.

These facts show how it is not possible to obtain correct and reliable attitude and heading information just from one type of sensor alone. The sensors fusion will consider the best measurement from all the available ones at that instant, providing better estimations. There are several methods widely documented in literature. For a multi-sensor combination on a rigid platform, in general, its fusion can be given by 1.6.

$$\begin{cases} \boldsymbol{D}_1^b = \boldsymbol{C}\boldsymbol{D}_1^r \\ \quad\dots \\ \boldsymbol{D}_n^b = \boldsymbol{C}\boldsymbol{D}_n^r \end{cases} \tag{1.6}$$

Where $\boldsymbol{D}_i^b = (D_{x,i}^b, D_{y,i}^b, D_{z,i}^b)_T$ denotes the vector observation of sensor i in the body frame and $\boldsymbol{D}_i^r = (D_{x,i}^r, D_{y,i}^r, D_{z,i}^r)_T$ represents the vector observation of sensor i in the world reference frame. $\boldsymbol{C}$ is the DCM. The previous equation can be converted to a least square loss function, which has to be minimized.

$$J(\boldsymbol{C}) = \sum_{i=1}^{n} \left\| \boldsymbol{C}\boldsymbol{D}_i^r - \mathbf{D}_i^b \right\|^2 \tag{1.7}$$

When the DCM is represented by $\mathbf{q} = (q_0, q_1, q_2, q_3)^T$, the single sensor fusion error is given by

$$\mathbf{f}(\mathbf{q}, i) = \left[\mathbf{D}_{x,i}^r\mathbf{P}_1(\mathbf{q}) + \mathbf{D}_{y,i}^r\mathbf{P}_2(\mathbf{q}) + \mathbf{D}_{z,i}^r\mathbf{P}_3(\mathbf{q})\right]\mathbf{q} - \mathbf{D}_i^b \tag{1.8}$$

In 1.8, $\mathbf{P}_i(\mathbf{q})\mathbf{q}$ is an alternative way to represent the i-th column of $\boldsymbol{C}$.

The combination of all sensors can be described by an augmented form such in 1.9, where a_i are the related weights. Weights are useful in specific conditions to give relevance to a sensor with respect to the others. The final combined error's function is:

$$\boldsymbol{F}(\boldsymbol{q}) = \mathbf{f}(\mathbf{q}, \{w, v, \dots, n\}, \{a_w, a_v, \dots, a_n\}) = \begin{pmatrix} \sqrt{a_w}\mathbf{f}(\mathbf{q}, w) \\ \sqrt{a_v}\mathbf{f}(\mathbf{q}, v) \\ \dots \\ \sqrt{a_n}\mathbf{f}(\mathbf{q}, n) \end{pmatrix} \tag{1.9}$$

The starting problem can be rewritten as

$$\arg \min_{\left\|\boldsymbol{q}\right\|=1} \left\|\mathbf{F}(\boldsymbol{q})\right\|^2 \tag{1.10}$$

There are several ways to solve such optimization problem. The next paragraphs outline some of the most common methods.

1.2 Filtering techniques

The system's state analysis is a crucial problem in modern automation and robotics. Some methods able to provide positioning of the robot are based on specific probabilistic algorithms. These algorithms correct the information coming from the odometry with the data obtained by exteroceptive sensors. In these cases, localization becomes a problem of state estimation, where the state is a random variable. Thus, the evolution of the robot position in time can be described as a dynamical system, which equations are describing the process and the measurement models.

The most common algorithms used in these cases are Kalman Filter (KF), Particle Filter (PF) and Complementary Filter (CF).

1.2.1 Kalman Filters

The Kalman filter, also known as linear quadratic estimation (LQE), is a problem where the estimation of the unknown variable tends to become more accurate increasing the number of measurements, computing a joint probability distribution over the variables. The filter is very powerful since it supports estimations of past, present and future states, and it can do so even when the precise nature of the model is unknown. The filter has two main phases. The first, is the prediction of the current state exploiting a mathematical model of the system. Then, such estimation, with the uncertainties, is updated with a weighted average of the sensors measurements. The assigned weight is related to the precision of the estimation. Such filter works well when noises have Gaussian distribution.

However, there are several variants to this linear estimator; this is quite important since most of the measurements from the real world comes with nonlinear uncertainties. Such variants are the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF).

Extended Kalman Filter

The EKF has differentiable functions as transition and observation models. It can linearize the current mean and the covariance. Let assume that the system has a state vector $x \epsilon \mathbb{R}^n$, and that the process is lead by the nonlinear stochastic differential equation:

$$x = f(x_{k-1}, \upsilon_{k-1}, w_{k-1}) \tag{1.11}$$

With measurement $z \epsilon \mathbb{R}^m$ described as:

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

In the previous equations w_k and ν_k represent respectively the process and measurement noises, and they are assumed to be zero mean multivariate Gaussian noises with covariances Q_k and R_k. This can be written as follow:

$$w(t) \simeq N(0, Q(t)) \tag{1.13}$$

$$\nu(t) \simeq N(0, R(t)) \tag{1.14}$$

Furthermore, the function 1.11 relates the state at instant k with the state at instant $k-1$, plus it has as parameters the driving function u and the zero-mean process noise w. Also, h in formula 1.12 is a nonlinear differential equation, and it relates the state x with the measurement z.

However, in practice, the actual values of the noises w and v are not known at each time instant. Therefore, the following approximations are considered:

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0) \tag{1.15}$$

$$\tilde{z}_k = h(\hat{x}_k, 0) \tag{1.16}$$

Where $\tilde{x}_k$ is a posteriori estimate of the state, and $\tilde{z}_k$ is its relative measurement. In this case, both functions can not be applied to the covariance directly, instead a matrix of partial derivatives (the Jacobian) is computed. First, for each time step, is computed the Jacobian A_k of partial derivatives of f with respect to x, and the Jacobian W_k of partial derivatives of f with respect to w. The partial derivatives of h with respect to x and v lead to the respective Jacobian H_k and V_k.

At this point let's define a new notation for the prediction error and the measurement residual:

$$\tilde{e}_{x_k} = x_k - \tilde{x}_k \tag{1.17}$$

$$\tilde{e}_{z_k} = z_k - \tilde{z}_k \tag{1.18}$$

However, the actual state is not known, but it is possible to define an error process like:

$$\tilde{e}_{x_k} = A(x_{k-1} - \hat{x}_{k-1} + \epsilon_k \tag{1.19}$$

$$\tilde{e}_{z_k} = H(\tilde{e_{x_k}}) + \eta_k \tag{1.20}$$

Where ϵ_k and η_k are independent random variables with zero mean and covariances WKW_T and VRV_T. From here it is possible to obtain a-posteriori state estimate:

$$\hat{e}_k = K_k \tilde{e}_{z_k} \tag{1.21}$$

Where K is the Kalman gain that must be computed during the measurement update phase. The final formula to be used in this update phase will be:

$$\hat{x}_k = \tilde{x}_k + K_k \tilde{e}_{z_k} = \tilde{x}_k + K_k(z_k - \tilde{z}_k) \tag{1.22}$$

The complete set of EKF equations are shown in the following groups of equations. The EKF time update step, described by equations 1.23, projects the state and covariance estimates from the previous time step to the actual one. While, in the EKF measurement update phase, equation 1.24, there is the correction of the estimations with the measurement z_k.

$$\begin{cases} \hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}, 0) \\ P_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \end{cases} \tag{1.23}$$

$$\begin{cases} K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \\ \hat{x}_k = \tilde{x}_k + K_k(z_k - h(\hat{x}_k^-, 0)) \\ P_k = (I - K_k H_k) P_k^- \end{cases} \tag{1.24}$$

Unscented Kalman Filter

Whenever the transition and update equations are highly nonlinear, the EKF doesn't work well, the UKF instead, uses a deterministic sampling technique to pick a minimal number of points among the mean and then propagate such points via the nonlinear functions, from which a new mean and covariance estimate are then computed. A quaternion based unscented Kalman filter is used in [1], to provide good heading estimation. The proposed multi sensor implementation includes an accelerometer, a gyroscope, and a magnetometer. The first measurement is provided by the magnetometer and accelerometer, but the signal is cleared using the gyroscope data. The result shows accuracy in heading estimation with errors less than 5° and 10°. However, the goodness of the result depends also on the fact that the drone was moving at constant velocity; more critical conditions could have changed the final measurements. For what Kalman filters are concerned, additional research is explained by [2]. In this paper an algorithm based on the filter is tested and evaluated. Such algorithm exploits the Wireless Sensor Network implemented for tracking a mobile target moving at low dynamics. The results show better accuracy of the Kalman filter compared to other Least Square estimators. The good behaviour of the filter was due also to the choice of Received Signal Strength (RSS) measurements. Furthermore, the paper reveals that with high nonlinearity, the EKF doesn't always guarantee convergence of the solution; in such cases the UKF was used, since it does not need to calculate the Jacobian.

1.2.2 Particle Filter

The particle filter can be used for non-linear systems, it is considered a better option than the Kalman filter with non-Gaussian distributions, but it requires higher computational effort. Probability densities are approximated by a set of weighted particles; higher weights indicate a higher probability that the particle is representative of the real pose of the robot. It has some similarities with UKF because both transform a set of point through known nonlinear equations and combine the result to estimate the mean and covariance of the state. However, in Particle filter the points are chosen randomly. Particle filter localization is also known as Monte Carlo Localization. The method consists of three consecutive steps:

1. prediction: where a new set of particles is determined from the previous one.
2. update: where each particle is assigned a weight.
3. resampling: when a new set of particles is determined to approximate the robot pose.

1.2.3 Complementary Filters

Kalman and Particle filter, thus, are a good and wide used option when comes to clean disturbances from signals, however, they are both quite complex from a computational point of view. On the other hand, another common solution is the Complementary Filter, which is a simple algorithm that uses the frequency domain methods to filter out the signal and obtains the estimation without any stochastic assumptions. There are many ways to realize a complementary filter, depending also on the strategy used to solve the optimization problem of equation 1.10.

In [3] the Madgiwick method described in [4] is revisited. The original algorithm is based on a fixed gain filter to estimate attitude expressed in quaternions using AHRS observations. Using such filter as base core, the quaternion estimation of the attitude was computed using an optimized Levenberg-Marquardt (LM) algorithm.

In [5] the complementary filter is realized using strap-down vector observations via gradient descent algorithm (GDA) instead of LM. Moreover, the two method, LM and GDA are compared. It is noticed that the final results are similar, however, the LM technique provides a smoother response. In fact, the Levenberg-Marquardt is cable of adapting the convergence rate of the iterations.

At the end, the flight controller chosen has an EKF algorithm implemented, which is able to estimate vehicle position, velocity and angular orientation based on the sensor's measurements. Its parameters can be tuned depending on the preferences of the pilot or on the requirements of the mission. Its code will be explained in the proper section later.

Independently on the case, any Extended Kalman Filter must be set in such a way to provide good stability. To do this, the combination of sensors and in which phase (prediction or update) they are used become an important choice. In [6] cameras and inertial sensors (accelerometers and gyroscopes) are fused together with an EKF for 3D tracking. Independently on the combination, what is known is that the camera data are improved with fused accelerometer data for the position estimation, while the gyroscopes help more the attitude estimation. Although the two types of measurements can be merged in either the first stage of filtering or in the second one, there are some differences between the eight possible combinations. One option is to use inertial information at the prediction stage, the second option is to use them in the correction phase. The considered paper analyses with both real data and simulated ones the eight combinations for sensor fusing, and at the end it concludes that the best results are obtained using both inertial sensors during the second step of the filtering process. Moreover, the paper demonstrate that, the accelerometer is more useful for 3D positioning, while the gyroscope is better for 3D orientation estimate.

1.2.4 Simultaneous Localization and Mapping (SLAM)

Indoor navigation presents multiple difficulties, such as the heading problem and the limitation in coverage of wireless localization. Some possible solutions, implementing sensors fusion, have been defined previously. However, there is a different way to solve these problems; such solution is Simultaneous Localization and Mapping (SLAM). SLAM is a computational problem of constructing the map of the unknown environment surrounding the robot. Given a series of control and sensor observation over a certain time, the purpose is to compute the estimation of the state variable and of the environment. The problem can be solved in different ways, like set-membership identification methods, expectation-minimization algorithms or with the already presented probabilistic filters. An example is presented in [7] where a SLAM augmented UWB localization is used to provide autonomous flight in an indoor environment; the quadcopter of the paper was able to map the room and recognize heading itself.

1.3 Pre-processing operations

Kalman filter, Complementary filter, personalized filter exploiting the methods and algorithms explained in the previous paragraphs are possible and valuable approaches for data fusion and outliers' elimination in attitude estimation. However, the accuracy of the estimation is reduced if the measurement and data are considered as raw as they come from sensors. To solve this problem a filtering action is needed. In the already cited paper [3], the magnetic distortion was treated applying a low pass filter for the magnetometer raw data. This filter reduces the effect of magnetic disturbances on yaw evaluation. Another possible solution requires the implementation of a pass band filter. Nevertheless, this filter must have a narrow band, and this is achievable just with a high order filter, increasing the complexity of the system and, in navigation cases, longer delays on the actions.
An example is presented in [8], where the raw data coming from inertial sensors are pre-processed using an adaptive variable bandwidth filtering via sinusoidal data estimation. Their solution can be presented in three consecutive steps:

1. 1: raw data from sensors are approximated as sinusoids at pre-set frequencies.
2. 2: the error between the original raw data and their approximation is computed by SSE (sum of squares error).
3. 3: the final filtering action is realized based on the operating frequency.

The sinusoidal technique is applied to guarantee less computational demand and making the filter suitable for real time applications. With this implementation, low frequency vibration influences are eliminated, making the signal smoother.

1.3.1 Sensor calibration

Another important operation, to improve the quality of the sensor measurement, is the calibration of raw data. This operation is critical both for magnetometer and accelerometer. There are several techniques available. One possibility is to use after-production calibrated sensors, but this solution requires the use of expensive equipment like Gauss magnetic chambers or Maxwell coils. A more interesting approach consists of specific algorithms to solve the calibration problem. Especially in the case of magnetometer, the noises can be distinguished in two categories. The first one is the soft-iron distortion, which belongs to the sensor reference frame and directly affects the measurement of the magnetic field. The second one is referred to as hard-iron distortion and is due to the presence near the sensor of magnetic material, which causes a permanent bias to the measured output. For 3-axis accelerometer calibration, the constant magnitude of gravity is mostly used as a reference. For 3-axis magnetometer calibration, the constant magnitude of the Earth's magnetic field is exploited. Nonetheless, in applications like autonomous navigation, or sensor fusion, the data coming from accelerometer and magnetometer are merged. Thus, the sensitivity axes of the sensors must be aligned together. This operation can be done using specific algorithms; in [9], the author exploits a gyroscope to compute the misalignment between sensitivity axes, in [10] a more personal algorithm is exploited. Alternatively, to decrease the complexity of the operation, just one sensor at time instant can be treated. In [3], the accelerometer is just calibrated using the sensitivity gains provided by the manufacturing sheet, without any deeper algorithm. However, the magnetometer is calibrated using the ellipsoid fitting method. This method is explained in [11], but in few words, without interferences, raw magnetometer data should form a perfect sphere, with diameter equal to intensity of the magnetic field. The presence of distortions affects the magnetic field shape, making it more like an ellipsoid. The interference can be collected during tests measurement in matrix form as corrective parameters to be used later during the effective measurements.

1.4 Flight configurations

1.4.1 Outdoor flight

For what outdoor flight is concerned, 2D localization is achieved thanks to the GNSS system. This returns the spatial coordinates of the vehicle, enabling localization and the possibility to program autonomous flight. Heading information in this case are generally provided by magnetometers, in this way it is possible to understand the orientation of the drone while moving, which is an important feature when comes to make the vehicle to follow a specific waypoints path. Heading is very important also for the obstacle avoidance system, since in our configuration, the more precise camera (able to identify obstacles) is placed on the front of the drone. In addition to GNSS and magnetometer, an IMU is used, to collect data regarding the acceleration around the pitch and roll axes (yaw is mainly given by the compass).

1.4.2 Indoor flight

Indoor, things are more complex. In fact, in this situation the environment is limited in space, and the risk of causing damages to things or people increases. Moreover, the presence of ferromagnetic disturbances inside the room makes the magnetometer based heading system unreliable. Finally, indoor, the GNSS system doesn't work. There are several possible implementations to achieve autonomous flight and correct positioning for the drone in such environments. A first approach is the realization of a room with OptiTrack cameras, these can provide 6DoF tracking with very high precision and low latency. However, this solution is quite expensive and limited by the fact that the required setup must be fixed, and thus cannot be implemented easily in multiple environments. Lately, in the market is spreading new low-cost wireless systems, one of these is the UWB. Ultra Wide Band is a radio technology for transmitting communications across a wide bandwidth enabling modulation of the signal. It can be used to determine the time of flight of the transmission, and, thanks to the low power characteristic, it is used for localization in different environments. An example of UWB application is the use of drones for inspecting close environments like oil tanks and vessels. This is explained in [12], where an ad-hoc algorithm, the Vision Aided Self-Localization Two-way Time of Flight (VASTWTOF) was implemented for the achievement of UWB anchors self-localization in GPS-denied situations. Ultra Wide Band represents a good solution, but it does not always return perfect results. In order to improve the measurements of the position, a sensors fusion is needed. Furthermore, localization is not just based on positioning, but also on heading (orientation). This problem is handled with the magnetometer in outdoor situations; but like GNSS, magnetometers suffer of indoor ferromagnetic problems. The reliability of this instrument must be tested merging the data with the ones of the IMU. Alternatively, to compute the orientation, two UWB tags can be placed, one on the front and one on the rear of the drone, but in this way the space needed on the quadcopter would increase. Another solution consists of exploiting UWB and LiDAR technologies together with map information of the environment [13]; in this case UWB is used to estimate the robot location and then the LiDAR to detect the direction of the robot along with the map data, the entire algorithm follows a least squared approach. In [14], an alternative to the LiDAR and RangeFinders, which are generally ToF (Time of Flight) sensors, is considered. The paper verifies three different indoor heading methods, where are used two cameras (one pointing forward, the second downward) and a magnetometer. The tests show how the downward camera highly depends on the surface characteristic, in fact, if the floor is rather monochromatic or reflective or moving, the measurements lose reliability. On the other hand, forward camera shows good results, exploiting a visual-based navigation technique. Furthermore, fusing the camera and the magnetometer together, will increase the accuracy; but it is important to remember that this fusion is not trivial, because it requires to place many sensors on the drone, and this is not always possible. Finally, autonomous flight is challenging matter also for the obstacle avoidance behaviour, to do this the drone must be able to sense what it has around in the space. The ability for an UAV to detect near obstacles is based on the exploitation of specific sensors. In this case, there are different solutions and configurations that can return good results. Generally, for this purpose, rangefinders, ultrasonic sensors and camera are implemented. To avoid overload of information the multithread behaviour of the flight controller is exploited: it is possible to use each time instant the most reliable sensor among the one available, obtaining the best possible performances.

1.5 Autopilot

The autopilot running on the copter is chosen between PX4 and ArduPilot. They are very similar, however the main two differences are that ArduPilot is older and, as a consequence, with a slightly more progressed and stable code. The second big difference is the license. ArduPilot has a GPL one, while PX4 has the BSD one. Thus, every change made to the ArduPilot code must be push on Github and made public, while the code cloned by PX4 can be maintained private. The latter is also one of the reasons for which the ArduPilot code is the more updated between the two. ArduPilot is a trusted, versatile, and open-source autopilot system supporting many vehicle types. The source code is developed by many professional figures and experts. It is constantly evolving based on rapid feedback from a large community of users. ArduPilot does not provide a specific hardware, however, the code works in many different boards available in the market. In this project case is used the PixHawk 2.4.8, which can receive information from the sensors and send commands to the copter devices like ESC, motors, camera etc. The features of the firmware depend on the type of vehicle that is considered. Anyway, some libraries are common to different vehicles. The pilot can interface with the UAV with a software called Ground Control Station (GCS); it can be downloaded in PC or mobile devices. For this project the GCS chosen are Mission Planner with Windows and QGroundControl with Linux. In addition, especially when simulation time comes, MAVProxy is preferred to the previous two because of its minimalist design and its characteristic to be used as a command-line based GCS; these facts make it also a good developer instrument. It is used with SITL for testing new builds. ArduPilot uses the MAVLink serial protocol for communication between the Ground Station and the UAV (either the main autopilot board or the Companion Computer) . MAVLink supports a wide range of messages, these can be found in common.xml and ardupilot.xml. Data can be sent over almost any serial connection not depending on the technology. Although MAVLink is reliable, the operator must always check the status of the vehicle and if the required command has been executed. This protocol will be useful especially when some calculations will be done by the CC and at the end comes to send the resulting instructions to the autopilot. The Ground Control Station for windows users is Mission Planner, it can be used with Google Maps to create a Point and Click waypoint/fence for the drone. As the name suggests it is used for mission planning, but it can additionally download mission log files and analyse them (in order to realize in-flight and post-flight monitoring), configure autopilot settings and interface with a PC flight simulator to create a SITL simulation of many frame types for all the ArduPilot vehicles. Furthermore, Mission Planner source code can be downloaded and compiled, since the code is written in C. ArduPilot developer guide presents some instructions on how to modify the Mission Planner platform; this can become useful to build a custom ground station. Previously, it has been noticed how the Pixhawk board is not enough powerful to compute all the data elaboration. The companion computer is an additional unit to the autopilot used to obtain more power in terms of processing data and instructions. This is fundamental during flight, especially when autonomous behaviour is needed. The used CC is the Nvidia Jetson Nano, it communicates with the PixHawk autopilot via MAVLink protocol over a serial connection. They should be alimented separately. The CC software refers to the programs and tools that run on the companion computer, they can read and run the telemetry data. Among different softwares, the Robotic Operating System (ROS) is very common and flexible. ROS is an open-source, meta operating system for the robot, it provides libraries, tools, hardware abstraction, low-level device control, message-passing between processes, visualizers and packages management to help software developers in creating robot applications. MAVROS is a ROS node that can convert between ROS topics and MAVLink messages allowing Ardupilot vehicles to communicate with ROS. The operating system has some specific terms which are very important to understand the different operations. A ROS node is an application, and it belongs to a package. The most important node in ROS is the Master node, it initializes the system and enables communications. Nodes communicate with each other via topics, which can be understood as a channel for exchanging async information. Such information is contained in messages, a message belongs to a package. In node communication, there is one node who send the message via topic, such node is called publisher. All nodes interested in that published data have to subscribe to that topic, these nodes are called subscribers. Moreover, synchronous communication between nodes is possible thanks to services. The node referred to as service server advertise the service, the one accessing the service is called service client. Testing new features can be dangerous, especially for the integrity of the copter. Therefore, new changes on the code should first be tested by simulations. The SITL (software in the loop) simulator allows to run the desired vehicle without any hardware. It is built from the ArduPilot code using a

C++ compiler. This is possible thanks to the portable nature of ArduPilot. When running the simulation, the sensors (like antennas, Lidars and optical flow sensors) data come from a flight dynamic model. The simulation allows the developer to exploit directly specific tools available such as interactive debuggers, static and dynamic analysers; in this way testing the functionality of the code becomes faster and easier. SITL can run in parallel with Gazebo and the simulation can be controlled by MAVProxy or directly with the classical GCS of the PC. Gazebo is another important virtual environment. It is a well-known and respected robotic simulator. At the time of the thesis, there is not an official built-in-support for Ardupilot, but on GitHub is available a custom version of ArduPilot with plugin for Gazebo. Such plugin can be used with or without ROS.

Chapter 2

Hardware description

2.1 Sensor list

In this work, both indoor and outdoor autonomous flights are analysed. Stability of the UAV is determined, as it has been written in previous chapters, by the correct combinations of different sensors. Some of them, like accelerometer, magnetometer, gyroscope, are already described, even if from a more theoretical and marginal perspective. In order to better understand the final implementation, a list of the main available sensors in the drone market is provided.

2.1.1 Intel RealSense d435i

The Intel RealSense d435i is a RGB depth camera that is implemented beacuse it is able to measure distances and thus it is considered as a reliable component for obstacle avoidance. The camera has its own microprocessor and IMU, in this way it can understand its position. It could be used also for object recognition. In addition, it has a shutter sensor able to guarantee a low-light sensitivity, which is a very important feature for a drone that is expected to move indoor.

Figure 2.1: Intel RealSense camera

2.1.2 Inertial Measurement Unit

The Inertial Measurement Unit (IMU) is the basic sensor of the vehicle. The project's IMU is based on the accelerometers and gyroscopes of the Pixhawk board. The flight controller hardware has also a barometer. The IMU is used to detect rotation and movements around the three main axes, in particular, it is possible to define the angles for yaw, roll and pitch. Eventually, such angles can be converted into quaternions depending on the final algorithm implementation.

2.1.3 Magnetometer

During the sensors fusion analysis, Micro Electro-Mechanical Systems (MEMS) IMU presents, further than accelerometer and gyroscope, magnetometers. In fact, the Pixhawk board has an internal compass, however, its precision decreases considerably in indoor applications. Therefore, the heading of the drone is guaranteed by the RM3100 magnetometer. It provides no drift, low noise, high sensitivity, no hysteresis, and it can be integrated with either I2C or SPI interfaces. Although it represents a good element for outdoor missions, and even if it shows better accuracy than the internal board magnetometer, in closed environments it can have some problems due to ferromagnetic material. Additionally, in [12], is shown the dependencies of height on the indoor positioning precision. Such uncertainties must be taken into account during the analysis and validation of the sensor accuracy.

Figure 2.2: Magnetometer RM3100

2.1.4 Global Positioning System

The GPS (Global Positioning System) is the reference sensor for positioning in outdoor environment. GPS is one of the GNSS systems that provide geolocation and time information to a GPS receiver. GPS is owned by the USA government and exploits the satellites in the space. It does not require the user to transmit any data, and it operates independently of any telephonic or internet reception. Nonetheless, the signal is relatively weak, and it can be disturbed by obstacles like mountains or buildings. The Taoglass ZED-F9P positioning module is the one used for the final implementation, it features the new u-blox F9 receiver platform and it can reach accuracy of centimetres. This precision is due to the implementation in the sensor of the Real Time Kinematic positioning (RTK), which is the application of surveying for common errors using measurement of the phase of the signal carrier wave, ignoring the information carried within.

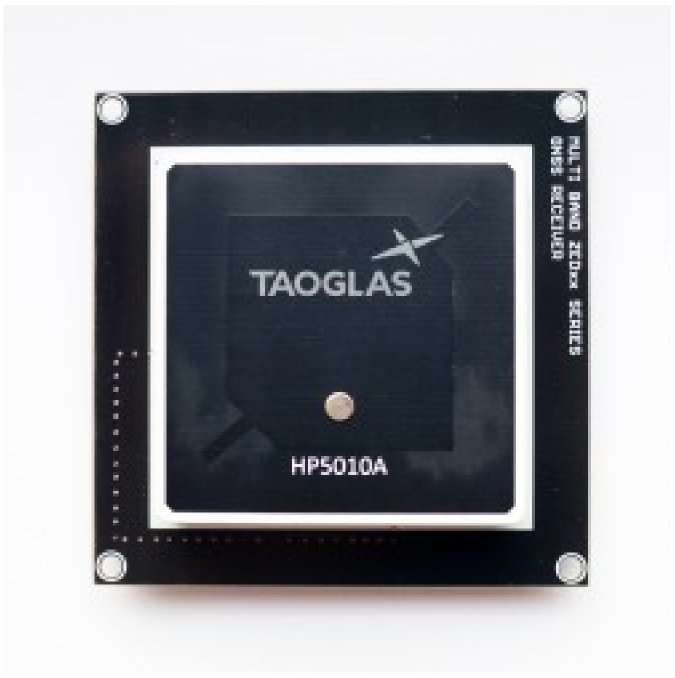


Figure 2.3: Taoglass ZED-F9P

2.1.5 Rangefinders

Rangefinders are laser sensor to measure distances between objects. The most common principle that is exploited by rangefinders is time of flight (ToF), which consists of sending a laser in a narrow beam and measuring the time taken by the pulse to be reflected by surrounding objects. Some different sensors are tested for height measurements. The first was the VL53L1X sensor. The original implementation wanted two of these sensors on the drone, one on the top of it, and one on the bottom. The data coming from these two rangefinders were merged thanks to the Kalman Filter. This solution is good in indoor environment since it is possible to monitor both the distance to the floor and the one to the ceiling. Moreover, the VL53L1X has a ready interface with ArduPilot and PixHawk board. Another similar sensor, with good integration with ArduPilot is the Benewake TFmini lidar. This should have a better accuracy of the VL53L1X both indoor and outdoor, in fact, it can measure up to 12 meters in closed environments, and 7 meters when outside. Furthermore, both are very convenient in terms of weight.

Figure 2.4: TFmini Lidar

2.1.6 Ultrasonic Sensor

Ultrasonic sensors are cheap alternatives to Lidars to achieve the capability for the drone to be aware of what are the surroundings, but their range of function is very limited. In this case the HC SR-04 chip emits high frequency sound wave (40 kHz) via one of its piezoelectric transducers and detects the returning pulse (echo) and converts it to a proportional voltage variation. The affinity depends on the light conditions, a parameter that can become critical both indoor and outdoor, furthermore, it is affected by the absorptivity of the reflecting material. Other parameters that can affect the measurements are noise, temperature and humidity. Because of these drawbacks, the sensor is neither use as main device for obstacle avoidance nor for height measure, but it should be eventually implemented to avoid lateral collision with unseen elements.

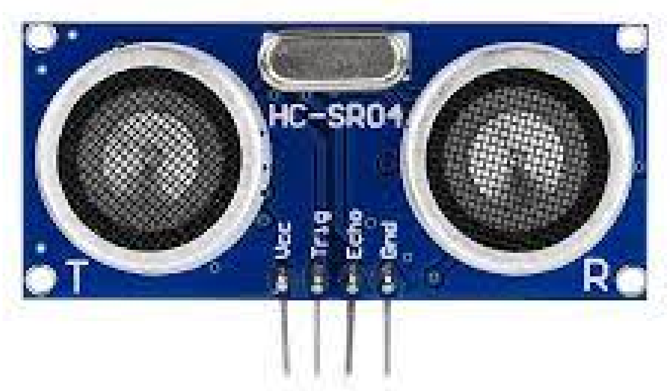


Figure 2.5: HC SR-04 sensor

2.1.7 Optical Flow sensor

For what indoor flight is concerned, one of the most used sensors in terms of accuracy and cost is the optical flow one. This consists of a camera module that uses ground texture and visible features to determine aircraft ground velocity. More than that, it is often used for non-GPS navigation. The one considered in this thesis is the PX4FLOW optical flow sensor, since it is well documented in the Ardupilot webpage. Such sensor is a specialized high resolution downward pointing camera module and a 3-axis gyro. Although this sensor is supplied by an internal sonar to measure height, this has not been reliable over multiple surfaces during testing. Thus, an external rangefinder like the one discussed above is suggested.

Figure 2.6: PX4 Optical Flow sensor

2.1.8 Additional Hardware

In addition to the previous sensors, a buzzer and a safety switch are used. These are not directly needed for the drone and its autonomous behaviour but help the operator to better interface with the vehicle and understand the different flight modes.

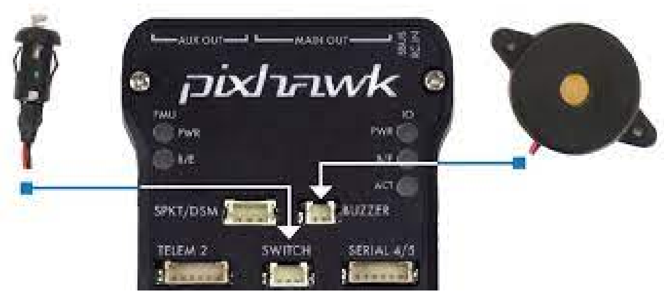


Figure 2.7: Buzzer and safety switch

2.2 Drone configuration

Until now, nothing has been said about the basic components of the drone. Essentially, any existing drone has the following elements: batteries, Electronics Speed Controllers (ESCs), electric motors and propellers. Moreover, there are many possible configurations for a copter, which can have from three to (normally) eight arms, the motors can be in a planar displacement or vertically aligned in pairs. The final drone for the FIXIT project is an octocopter (with eight motors), two motors per each arm, displaced in a counter rotation way. This configuration is chosen to maintain the spatial size of a quadcopter but with double thrust. Nowadays, the drone motors are only brushless. These are synchronous DC motors using direct current electric power supply. This uses an electronic controller to switch DC currents to the motor windings producing rotating magnetic fields which the permanent magnet rotor follows. Such controller is the ESC, and it must be chosen depending on the amount of maximal current that the motor requires. This quantity can be approximately estimated considering the size of the motor, its KV (speed constant, $[rpm/V]$) and the size of the propeller. The KV is one key parameters on the choiche of the motor, and it indicates the speed per volts of the rotor without any attached propeller. In general, grater KV tends to move faster the rotor, thus are used with small propellers; using bigger propellers will require higher currents. The important fact is that the current flowing is not too high, in order to avoid overheating. However, once the motor is chosen, the productor suggests specific ESCs and propellers. The motor is selected depending on the guaranteed thrust, which should be enough to contrast the total weight of the drone and to allow certain agility. The total weight is computed considering the copter itself plus the payload, which is everything on the drone that is not directly correlated to the propulsion system. The LiPo battery has to be able to provide the correct amount of current, moreover it is characterized by two parameters, the discharge rating C and the capacity in mAh:the first one is the speed at which it can be charged completely, the second is the time of flight for a drone mission. The max current draw can be computed as follow:

$$MaxCurrentDraw\ [A] = Capacity\ [mA] \times C_{rating}\ [C] \tag{2.1}$$

Throughout the tests, the small spaces available for indoor flight causes some difficulties with the octocopter model, in fact, it has a 450mm diagonal and weights almost 3kg. Thus, in order to limit the damages to the UAV and to the surrounding devices in the room, a smaller copter is built. This latter, is similar to race quadcopters, it has a 250mm diagonal and weights more or less 700g. Both drones will be tested in different conditions. Between these two copter models, apart from the size and the type of the propulsion elements, the biggest difference is the on-board autopilot. The smallest mounts the Pixhawk 2.4.8 while the other has the Cube Orange. However, both boards can have external companion computers (CC). The two drones have an Arduino each for UWB implementation, the octocopter has also the Jetson Nano, which can be used to handle the Intel RealSense depth camera.

2.2.1 Autopilot boards

Pixhawk 2.4.8

The 2.4.8 board is not the newest, however, it is relative cheap, and it has everything is needed. Moreover, it is highly documented, which makes it the perfect one for starting a new project. Its main characteristics are:

1. ARM Cortex M4;
2. 168 Mhz/256 KB RAM;
3. MPU6000 as main IMU;
4. ST Micro gyroscope;
5. ST Micro accelerometer/compass;
6. MEAS barometer

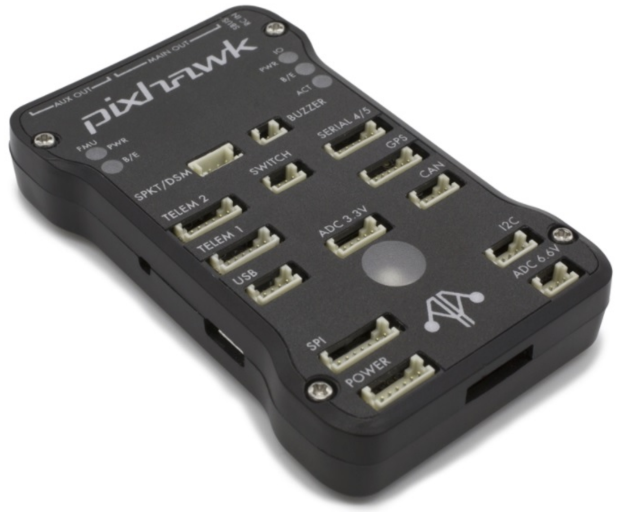


Figure 2.8: Pixhawk 2.4.8

Cube Orange

At a certain point, the Pixhawk board has been replaced by the Cube Orange flight controller. The two devices differ, rather than in dimension, by the number of interfaces, the quality of the internal sensors and the processor features. The Cube has some additional I/O ports, which allow to add external accessories to the board, increasing the UAV potentials. In addition, the Cube has three redundant IMUs compared to the single one of the Pixhawk. This will be commented in later paragraphs (6.2.4). The onboard available sensors are:

1. ICM 20649 (IMU) and MS5611 barometer;
2. ICM 20602 (IMU) and MS5611 (temperature controlled barometer);
3. ICM 20948 (IMU) and MS5611 (temperature controlled barometer);

In addition, the board has a bigger RAM (1MB versus 256KB of the Pixhawk) and a higher rate (400Mhz versus 168Mhz), these features are fundamental for some advanced implementations. Finally, the Cube has the Cortex M7 processor. Compared to the Cortex M4 available in the Pixhawk board, the newest is more expensive, but executes branches faster and read memory twice as fast.

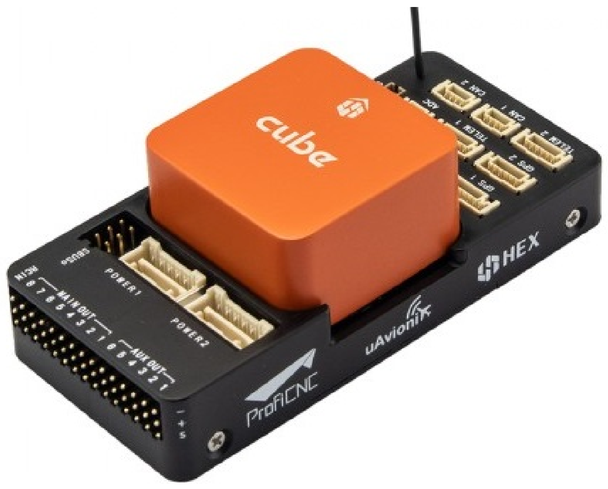


Figure 2.9: Cube Orange

Chapter 3

Learning the code languages

3.1 ArduPilot

As already explained, the ArduPilot firmware was the starting point for this thesis project. However, the code may have some errors which must be corrected to improve the performance of the UAV. In order to debug the code, is important to be able to understand it deeply. In this chapter a brief description of the code is granted.

3.2 Structure of the code

3.2.1 High level

The entire code is organized into five main parts, which are:

1. the vehicle directories: these contains the code related to one specific vehicle type; there are .cpp files and the list of library dependencies;

2. shared libraries: which are the parts that contain the code shared among all vehicle types;

3. the Hardware Abstraction Layer (AP_HAL): is used to interface the code with many different boards;

4. tools directories: are miscellaneous support directories. For example, the Pozyx code running inside the arduino board is copied in one of these directories;

5. external support code: used generally to provide board support.

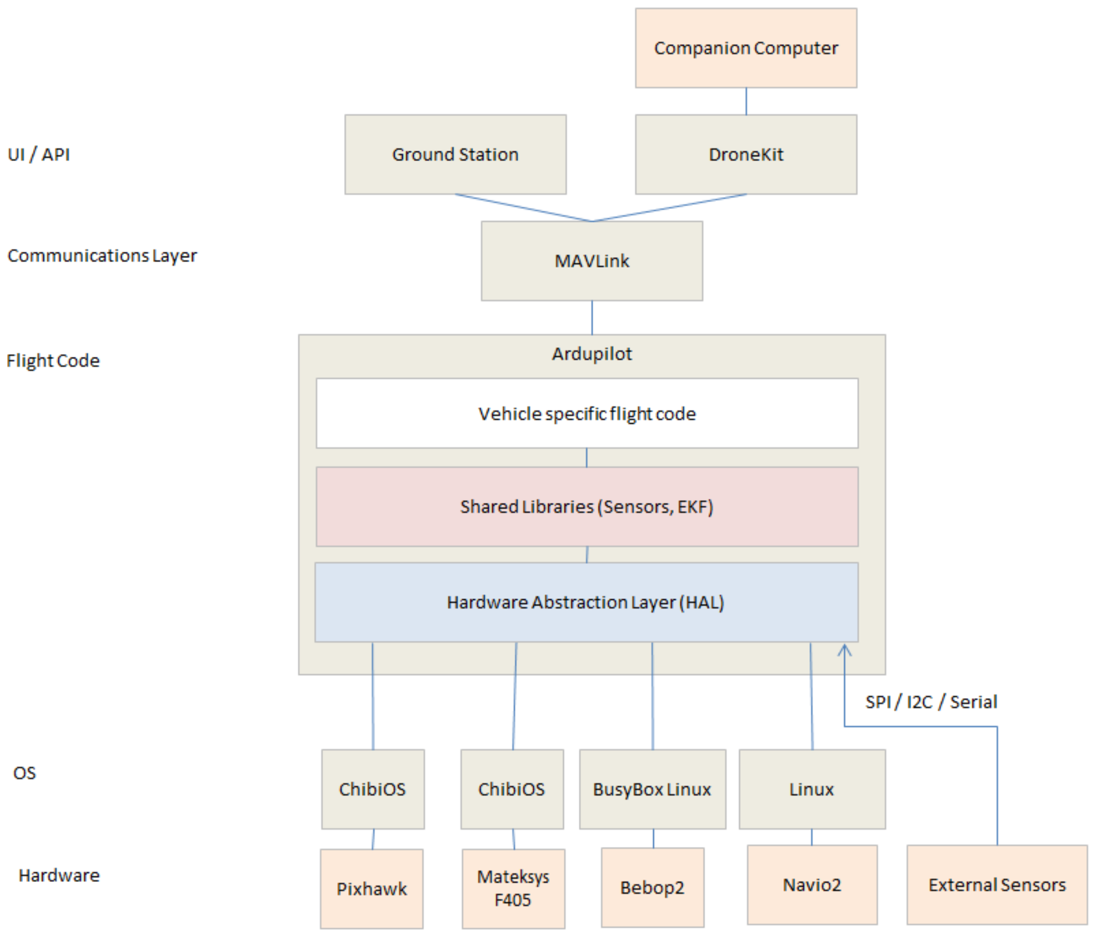


Figure 3.1: High level architecture of the code

3.2.2 Low level

More in detail, each one of the part listed before, can be analyzed deeply. Throughout this work, the main firmware required some changes; this means to handle the code at a lower level. Among all the libraries and folders of the ArduPilot project, the most important are the ones concerning the beacon system implementation and the Extend Kalman Filter tuning. However, except for solving some bugs and adding few personalized parameters for monitoring, the ArduPilot code remained almost the same as the one available on Git.

Although the new additional functions are implemented via Arduino, to better understand what's coming next, it is fundamental to describe some libraries characteristics of the code. Thus, the AP_NavEKF3_RngBcnFusion function, responsible for the UWB data fusion in the Extended Kalman Filter, is described.

This method works in a very peculiar way; every k instant it receives as inputs the North, East, Down positions of one anchor and the relative measured range. As a consequence, the filter doesn't work computing the position error but the range error. In this way, the estimated position is done just using one anchor each time, not the entire set. The filter variables are listed.

- bcn_{rng}: measured range between the tag and the beacon
- p_n: North position of the UAV
- p_e: East position of the UAV
- p_d: Down position of the UAV
- $\boldsymbol{pos}$: is a vector which components are (p_n, p_e, p_d)
- bcn_{pn}: North position of the beacon
- bcn_{pe}: East position of the beacon
- bcn_{pd}: Down position of the beacon
- $\boldsymbol{bcn_{pos}}$: is a vector which components are $(bcn_{pn}, bcn_{pe}, bcn_{pd})$

Then, recalling 1.2.1, the EKF is realized first by defining the range prediction (3.1) and innovation (3.2).

$$rngPred = |\boldsymbol{pos} - \boldsymbol{bcn_{pos}}| \tag{3.1}$$

$$innovRngBcn = rngPred - bcn_{rng} \tag{3.2}$$

Since the innovations (ranges) are not the desired system states (NED position), the transformation matrix H must be computed. This is done for each one of the North, East, Down components.

$$temp = \{\sqrt{[(bcn_{pn} - p_n)^2 + (bcn_{pe} - p_e)^2 + (bcn_{pd} - p_d)^2]}\}^{-1} \tag{3.3}$$

$$H_n = -temp(bcn_{pn} - p_n)^2 \tag{3.4}$$

$$H_e = -temp(bcn_{pe} - p_e)^2 \tag{3.5}$$

$$H_d = -temp(bcn_{pd} - p_d)^2 \tag{3.6}$$

Once this is done, the variance of the range innovation is computed, and from here, the Kalman gain is defined as in 1.24. Actually, during the state update, the Down component is not changed if the primary source chosen is different from the beacon system. As already discussed, in this project the height of the UAV is given either by the barometer or by the Lidar, thus the third component of the position can be neglected.

3.3 Arduino

Arduino IDE will be used a lot in the following operations. Thus, is important to evince its main characteristics. Basically, Arduino is an open source electronics platform based on easy-to-use hardware and software. In fact, it is intend for a large diversity of clients so that everyone can be able to realize small projects. However, expert users have the possibility of doing their own code and/or improve already exiting libraries on the basis of their willingness.

3.3.1 Hardware selection

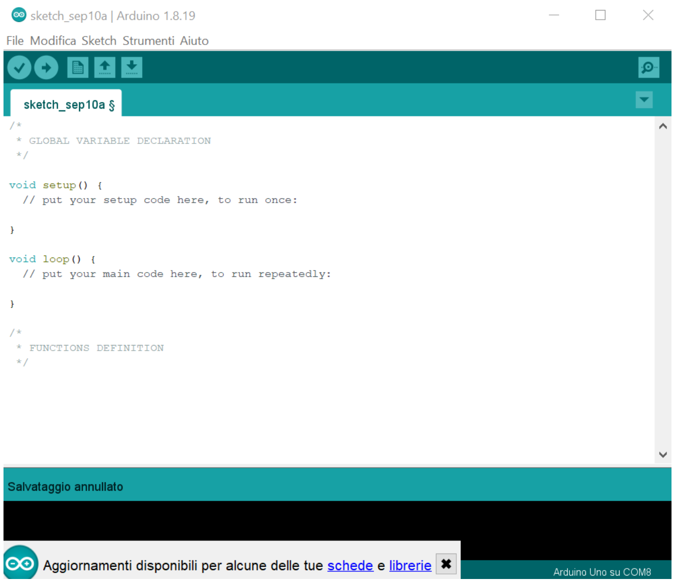
In the previous chapters were presented both the Arduino and the Jetson Nano boards. However, up to know, the FIXIT UWB system is implemented with the Arduino UNO board. As a consequence, it is important to highlight the differences between the two solutions. Instead of the Arduino, a more general USB could be used, this leads to:

1. Advantages:
   (a) Very cross platform serial protocol;
   (b) Exploit Python for flexible functionality and extendability;
   (c) Computers (Jetson or Raspberry) can be as powerful as desired;
   (d) Serial communication can be implemented in any programming lenguage.
2. Drowbacks:
   (a) Arduino is cheap and small, compared to other hardwares.

In addition, it can be said that if the user is more comfortable using C++ as programming language, it can be better to use Arduino rather than Jetson or similar, since the latter runs with Python. The choice should be done also considering the final mission, in fact, the computers listed above have a RAM of at least 2GB, while Arduino Uno has 2kB. Moreover, Arduino stores just 8-bit instructions in stead of the 32-bit available with computers.
Nonetheless, since the Arduino board would be directly connected to the Pozyx module for controlling its features and send data to the flight controller, it is preferred with respect to the Jetson computer.

3.3.2 Arduino IDE

The Arduino programming environment can be divided in two parts, which are called "Setup" and "Loop". As their names may suggest, the first is run just one time when the code is launched into the hardware, while the second one has the main instructions which are execute repeatedly. In addition, generally there is an upper area where global variables are declared, while at the end of the page the functions are collected. The described way, displayed in figure 3.2, is how have been written the codes for this projects.


sketch_sep10a | Arduino 1.8.19
File Modifica Sketch Strumenti Aiuto
sketch_sep10a §

```
/*
 * GLOBAL VARIABLE DECLARATION
 */

void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}

/*
 * FUNCTIONS DEFINITION
 */
```

Salvataggio annullato

Aggiornamenti disponibili per alcune delle tue schede e librerie

Arduino Uno su COM8

Figure 3.2: The Arduino IDE environment

Arduino has several features that must be listed in order to fully understand the procedures in the following chapters. First of all, Arduino has a libraries cloud, with ready-to-use functions. These can be downloaded and included in the code. Besides, some interesting code is available on platforms such as GitHub; from here is often possible to obtain zipped libraries for more advanced code. Regarding this fact, the Pozyx group shares codes and libraries with their clients in order to provide a simple software implementation and a ready-to-use product. These examples are listed in the Arduino IDE menu, in section "Examples". These referenced operation can be called as follows "Pozyx.name_of_the_function".
Another useful Arudino's tool are the Serial monitor and the Serial plotter. Both can be used for debugging and displaying results. However, the monitor simply shows the values of the variables of interest, while the plotter just provides the time evolution through a visual representation.

3.3.3 Processing

Often, understanding data scrolling on the serial monitor of the Arduino IDE could be very difficult because of the fast rate. Unfortunately, the situation does not get any better even with the serial plotter. In this cases, Processing comes in hand. Processing is not subjected to Arduino, it is a flexible sketchbook provided by Processing Foundation, but it can be used to read serial data elaborated in the Arduino COM port and plot the results. It was used especially for testing and visualize UWB performances: the master tag was connected to the Arduino and then to the ground station (which had processing installed), while the slave tag was the one used for collecting data. The variables were passed to the master tag and then elaborated. Eventually, the results were available in the serial port, and use by Processing to realize better graphs.

3.4 Matlab

Even if it isn't listed in previous chapters, Matlab is a very powerful tool for analysis of data. It will be used for plotting and data manipulation. More in details, the logs collected during flight are downloaded and converted into .mat files using Mission Planner. Whats is returned is read by Matlab as a struct element. Such objects can be analyzed and compared with simple Matlab script. Eventually, the log variables list will be modified with some custom adds.

Chapter 4

Sensors data implementation

Sensors have been already listed in chapter 2, however, some of them need to be further discussed. In fact, sensors need to be calibrated, in a similar way as explained in paragraph 1.3. Moreover, before their implementation on the drone is important to test their performances.

4.1 Raw Data

Raw data, sometime also called source data, are non-processed information, directly evaluated by the sensors. These quantities are difficult to understand, and it is quite complicated to use them directly for processes development. In fact, raw data need to be converted into physical reasonable quantities. However, this conversion is not as trivial as it could seems. In order to obtain useful values is important to understand the contest the raw data are collected from, as a consequence, the physical data will be extrapolated differently from one sensor and another. Luckily, in both Ardupilot and Pozyx libraries there are specific sections where conversion units for each used sensor are listed as global variables. In this way, such data are accessible from every part of the remaining code, and can be used to obtain useful data.

4.2 Pozyx UWB

The Pozyx tag is mainly used for indoor localization, however, it has some additional features such as an internal IMU and a magnetometer. Its characteristics are evaluated in this chapter. The system consists of four anchors, which are displaced in the corner of a square. The height of the devices is not critical for the copter positioning. In fact, the Pozyx provider suggests to place the anchor not at the same level from the ground whenever the user needs a greater accuracy for altitude estimation. Nonetheless, in this project, altitude is measured with the Lidar and barometer. However, the measure of the height of the anchors should be as precise as possible to increase the Pozyx algorithm accuracy. For all the tests in this project, the anchors are displaced with their ID increasing through an N-path, such as in figure 4.1.

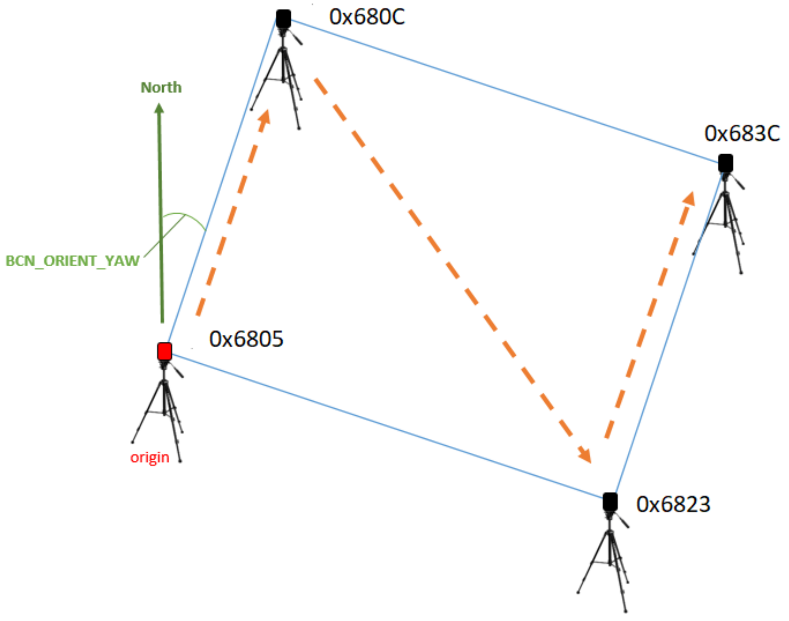
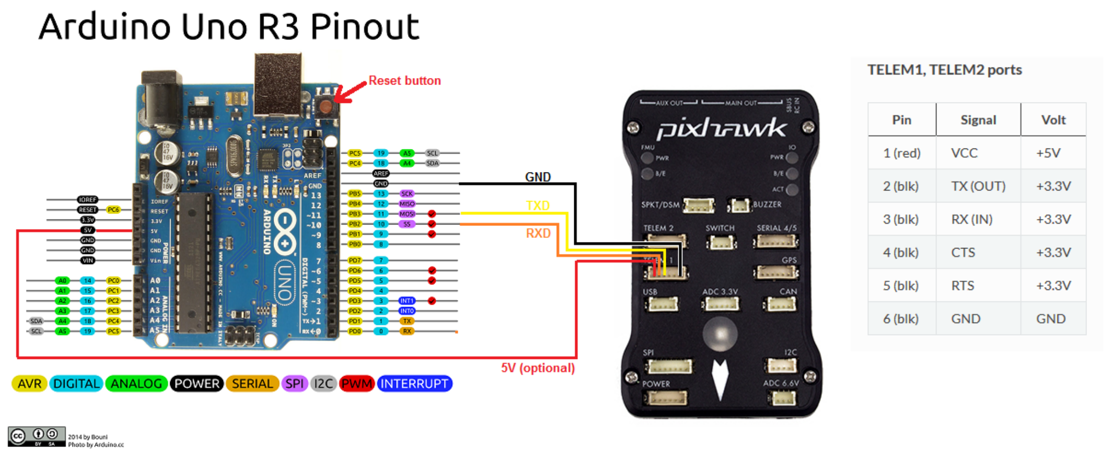


Figure 4.1: Pozyx's anchors displacement

In this work, the anchors ID are 0x6805, 0x680C, 0x6823 and 0x683C. This set-up determines a 2D local reference system which origin is the first anchor (0x6805). The X and Y axis are respectively the one between anchors 0x6805-0x680C and 0x6805-0x6823. The distance between the anchors is changed in order to understand the capabilities of the system. The results of the test are discussed later but, all changes require just little adaptations in terms of code. This fact makes the Pozyx system adaptable to different working conditions.

4.2.1 UWB implementation on the UAV

It is recommended to mount the tag vertically, however, for convenience, the tag is connected to the Arduino board, where the positioning algorithm is running, and both devices are placed at the top of the UAV. The connection between the Arduino and the autopilot is done via the telemetry port, however, the modality changes accordingly to the selected board. In the Pixhawk 2.4.8 the plugging is displayed in figure 4.2. Actually, the wiring is not established between the tag and the Telem1 port as shown in the figure, but with the Telem2 one. Regarding the Cube Orange, the Telem1 port is exploited.



TELEM1, TELEM2 ports

Pin	Signal	Volt
1 (red)	VCC	+5V
2 (blk)	TX (OUT)	+3.3V
3 (blk)	RX (IN)	+3.3V
4 (blk)	CTS	+3.3V
5 (blk)	RTS	+3.3V
6 (blk)	GND	GND

Figure 4.2: Autopilot connection to the UNO board

For each sensor the code requires some adaptations. Such changes are done working on the parameters, which can be modified through Mission Planner. To exploit the Pozyx tag for indoor localization, this is the set-up:

1. set BCN_TYPE to 1: means use Pozyx as beacon system.
2. set BCN_LATITUDE, BCN_LONGITUDE and BCN_ALT to match the location of the flight tests (this is done exploiting Google Tools).
3. set BCN_ORIENT_YAW:angle between the true North and the local x-axis.
4. set GPS_TYPE to 0: this disables the GPS for positioning.
5. set ARMING_CHECK to disable GPS pre-arm checks.
6. set SERIAL2_BAUD to 115.
7. set SERIAL2_PROTOCOL to 13.
8. set BRD_SER2_RTSCTS to 0.

4.3 Pozyx magnetometer

The Pozyx device, in addition to localization, can compute its own orientation in the space. In fact, the tag has an on-board 3-axes magnetometer, a 3-axes accelerometer and a 3-axes gyroscope. These sensors will have an important role in further solutions, so it is important to treat them accordingly. Among the three listed sensors, the one which requires the higher level of attention is the magnetometer. In fact, it must be calibrated. In paragraph 1.3.1 were enunciated several possible methods to achieve this task; in this case a specific method from [15] is used. To complete this, the programs MagMaster and MagViewer are needed, plus some Arduino code and a paper box, as the one in figure 4.3.

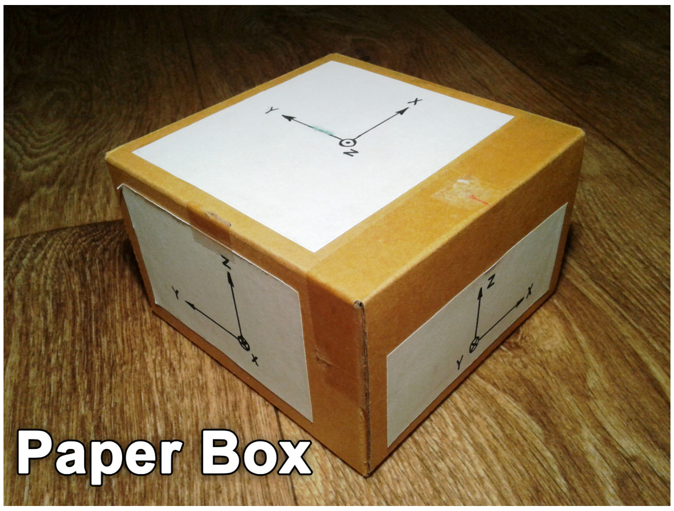
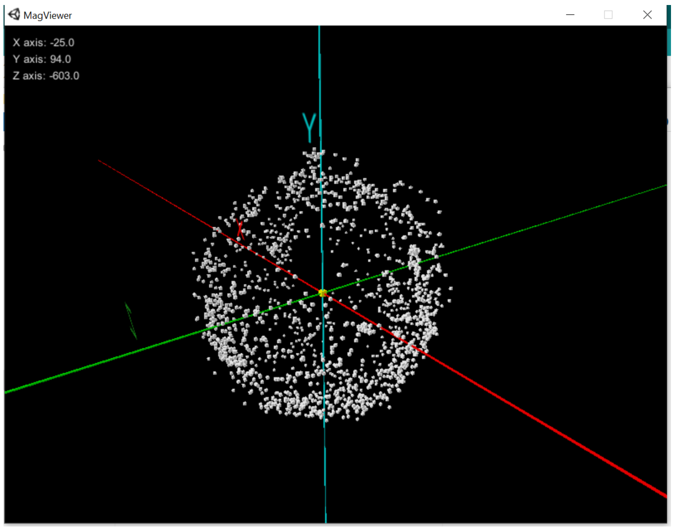


Figure 4.3: The paper box used for calibration

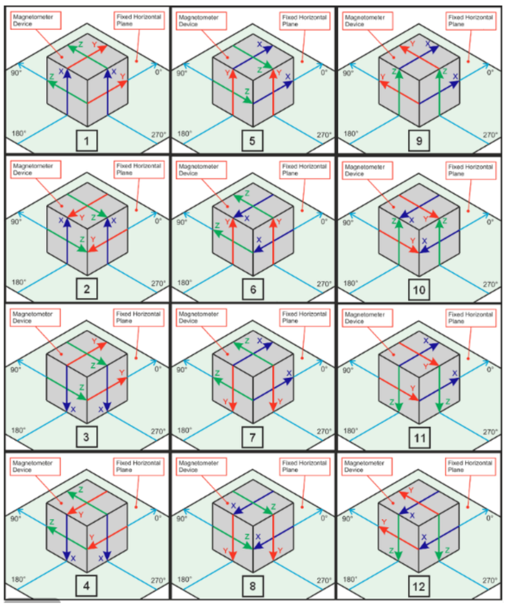
The sensor must be placed on top of the paper box. Its surfaces have the reference system x, y, x according to the orientation of the sensor. This will increase the accuracy during the following steps.
Before starting, the Arduino code must be written, it would be very easy. The main point is to call the *Pozyx.getRawData* to obtain the raw measures from all the tag's sensors. From these, only the *raw_sensor.magnetic* array is printed by the Serial monitor. At this point, the MagViewer tool is used to observe the behave of the magnetic fields strength on the 3-axes; this is shown in figure 4.4.



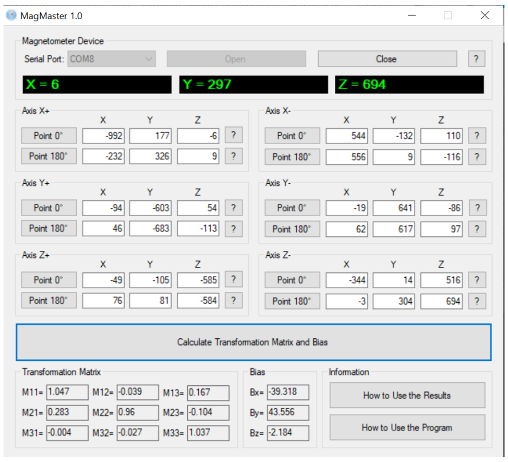
```
X axis: -25.0
Y axis: 94.0
Z axis: -603.0
```

Figure 4.4: Not-calibrated magnetometer measurements

Then, MagMaster is opened, the Arduino Serial port is selected, and the x, y, x magnetic measurements are displayed on the top of the program page (figure 4.5b).The paper box with the magnetometer attached to it, should be positioned as indicated in figure 4.5a, and after each step, the measurements must be inserted in the proper space in MagMaster.



(a) Calibration steps



Magnetometer Device

Serial Port: COM8

X = 6 | Y = 297 | Z = 694

Axis X+	X	Y	Z
Point 0°	-992	177	-6
Point 180°	-232	326	9

Axis X-	X	Y	Z
Point 0°	544	-132	110
Point 180°	556	9	-116

Axis Y+	X	Y	Z
Point 0°	-94	-603	54
Point 180°	46	-683	-113

Axis Y-	X	Y	Z
Point 0°	-19	641	-86
Point 180°	62	617	97

Axis Z+	X	Y	Z
Point 0°	-49	-105	-585
Point 180°	76	81	-584

Axis Z-	X	Y	Z
Point 0°	-344	14	516
Point 180°	-3	304	694

Transformation Matrix

M11= 1.047	M12= -0.039	M13= 0.167
M21= 0.283	M22= 0.96	M23= -0.104
M31= -0.004	M32= -0.027	M33= 1.037

Bias

Bx= -39.318
By= 43.556
Bz= -2.184

Information

(b) MagMaster view

Figure 4.5: Calibration Process

When all the above procedure is complete, the transformation matrix and the bias are computed by MagMaster. The results are displayed at the bottom of figure 4.5b.

Finally, the Arduino code must be upgraded with two more functions:

1. $transformation(raw\_magnetic\_data)$;
2. $vector\_length\_stabilization()$.

The first one is used for the correction of the measures, starting from the not calibrated raw data while the second normalizes the radius of the magnetic sphere. To be more precise, the first one computes the following operation:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = \begin{pmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{pmatrix} \left(\begin{bmatrix} X_{nc} \\ Y_{nc} \\ Z_{nc} \end{bmatrix} - \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \right) \tag{4.1}$$

In equation 4.1 $(X_c, Y_c, Z_c)^T$ and $(X_{nc}, Y_{nc}, Z_{nc})^T$ are respectively the calibrated and not calibrated vectors of magnetic measurements, while the $M_{i,j}$ and $(B_x, B_y, B_z)^T$ are the transformation matrix and bias vector from MagMaster.
Eventually, the calibrated results is shown in figure 4.6.

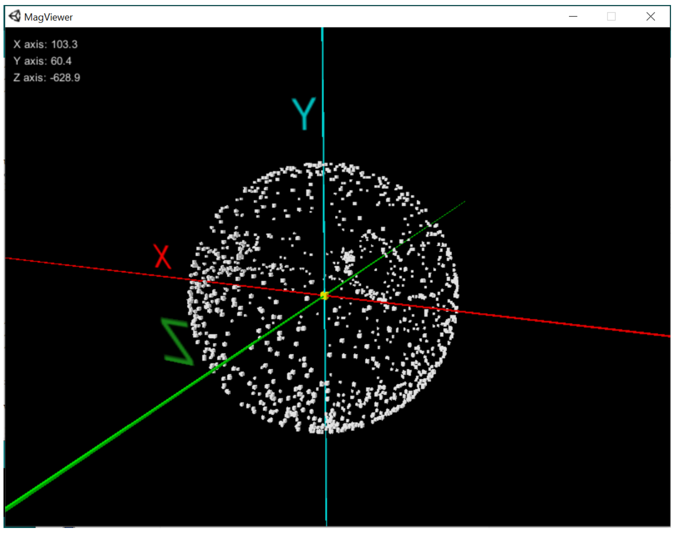


Figure 4.6: Calibrated magnetometer measurements

4.4 Lidar vs Barometer

The altitude in the copter can be evaluated in different ways. The main source generally is the barometer. However, especially indoor, this sensor is not always reliable. Moreover, it is very noisy, which is not a good feature. As a consequence, the Lidar is added, this was described in 2.1.5. Especially indoor, this sensor can work in a range from 1 cm up to 12 meters. However, since the drones of this project are tested both outdoor and indoor, the filter is fed with the two sensors together. The final altitude estimation takes into account also the IMU data. The performances are compared in the images below.

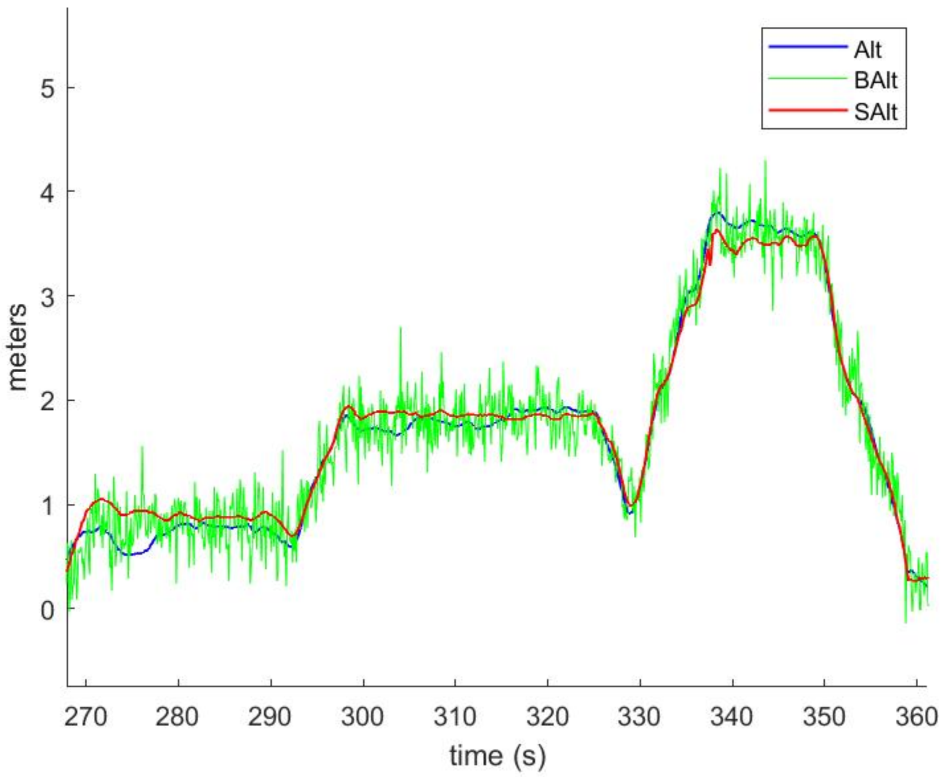


Figure 4.7: Altitude sensors comparison

Figure 4.7 shows in green the barometer measure, the red line is the Lidar measurement while the blu one is the estimated altitude computed by the EKF. The main source is the Lidar, in fact, the blue line has a behaviour similar to the red one, however, its value is corrected each time with the barometer measurement.

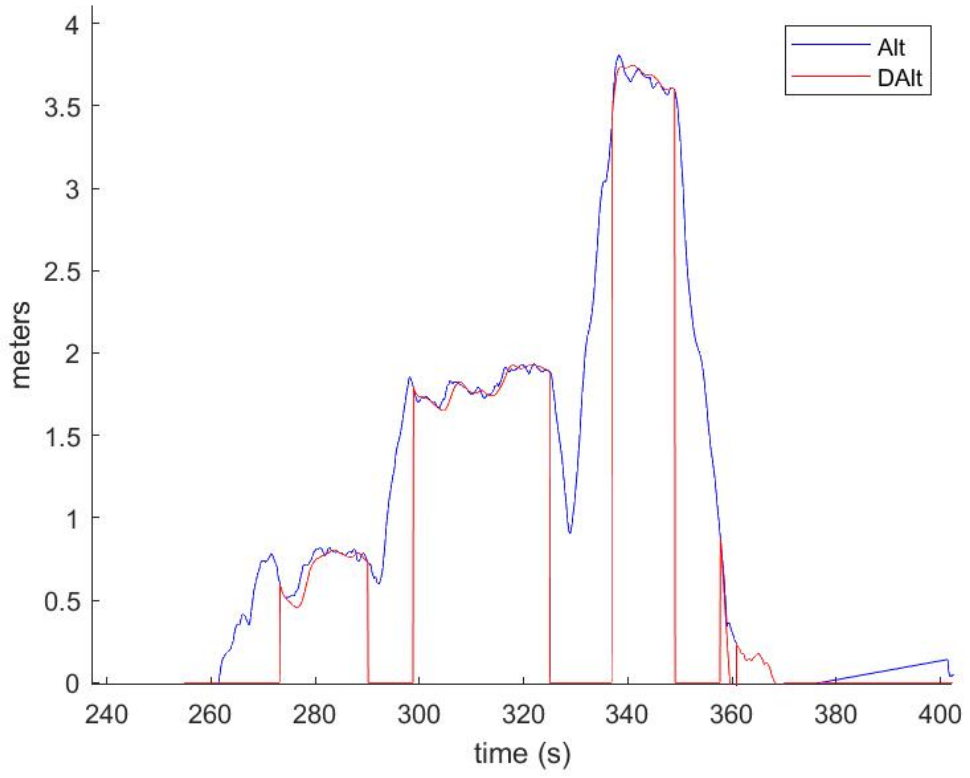


Figure 4.8: Altitude evaluation

Instead, figure 4.8 represents the desired altitude (red line), which can be assigned either by an AUTO mission command or by the throttle level stick of the pilot, compared to the actual altitude estimated by the filter (blue line). In this case, the desired altitude is established in-flight by the pilot, the UAV behaves quite well, following the trace.

4.5 RM3100 vs HMC5883L

Both Pixhawk 2.4.8 and Cube Orange boards come with internal compasses. However, these magnetometers cannot provide reliable heading information alone. As already anticipated, external compasses are added to the UAV, these are the RM3100 and the HMC5883L. The following pictures (4.9) show more in details the differences between the available magnetometers. It is very clear how the internal one is the worst. Moreover, the RM3100 shows a more stable behaviour and lower amplitude oscillations than the HMC5883L (which is cheaper). These facts determine why the RM3100 is the chosen one, although, the MC5883L will be used too in further applications.

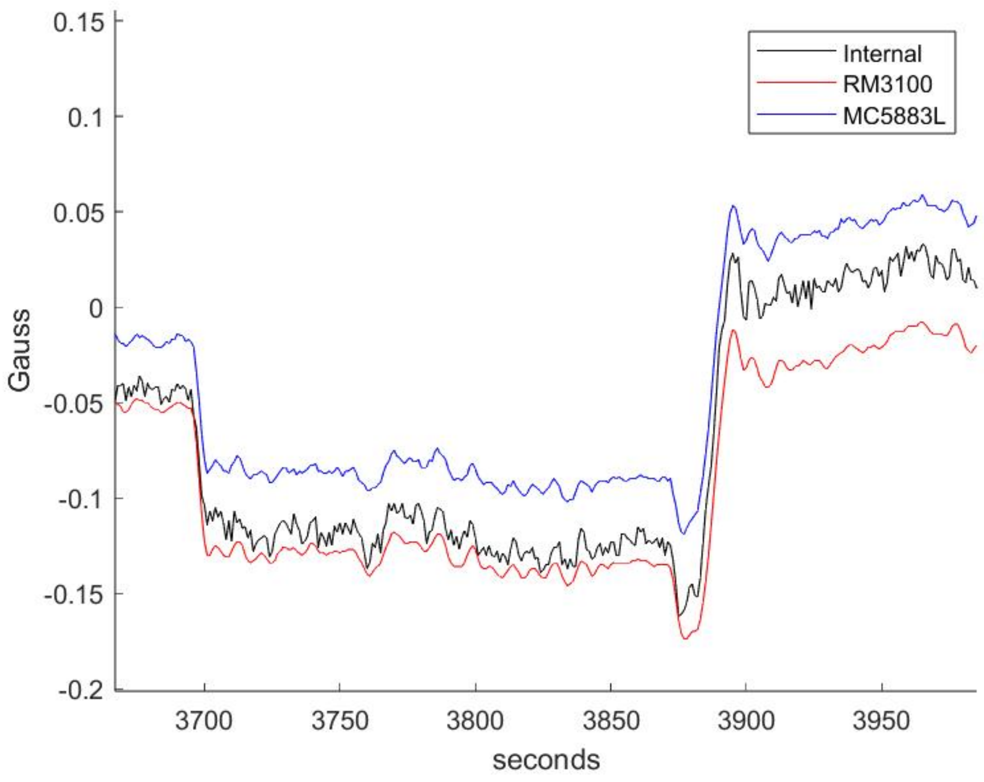


Figure 4.9: Magnetometers comparison

Chapter 5

First set-up and performance analysis

5.1 Arduino setup

As explained before, Pozyx team provides an online source of code in order to start testing their devices. Initially, the system performances were evaluated using the *ReadytoRange* function, which computes the distance of the tag to each single anchor. Then, once the hardware setup was completed, the *ReadytoLocalize* function was tested with good results in positioning. However, at the end, the code running on the Arduino was something provided by ArduPilot developers. Such algorithm, exploits the same function of *ReadytoLocalize*. For this study, some modifications were made. First of all the first lines of the code must match the actual anchor ID in the correct order. Then, in order to obtain a more reliable result, the automatic function for the computation of the distances among anchors was disabled. Instead, the manual configuration was activated, measuring by hand the perimeter of the local system identified by the anchors. The available code works sending position information every two seconds. This is done since the autopilot exploits the distances from the anchors and not the computed position. Still, in order to improve post flight analysis, the position messages rate between Arduino and the board is increased up to the maximum available, which is around 90 milliseconds.
The Arduino is connected to the ArduPilot board as explained in paragraph 4.2.1. In this first part, the Arduino is loaded with the custom code, while the autopilot is the one latest version of ArduCopter available online. First, the octocopter is used, but, for the reasons explained before in section 2.2, it is sided by the smaller quadcopter.

5.2 Tests and problems analysis

5.2.1 Positioning

The anchor configuration respects the guidelines in paragraph 4.2; the following is a reconstruction of the reference system using Google Earth. Anchor 1 is 0X6805, anchor 2 is 0X680C, anchor 3 is 0X6823, anchor 4 is 0X683C.

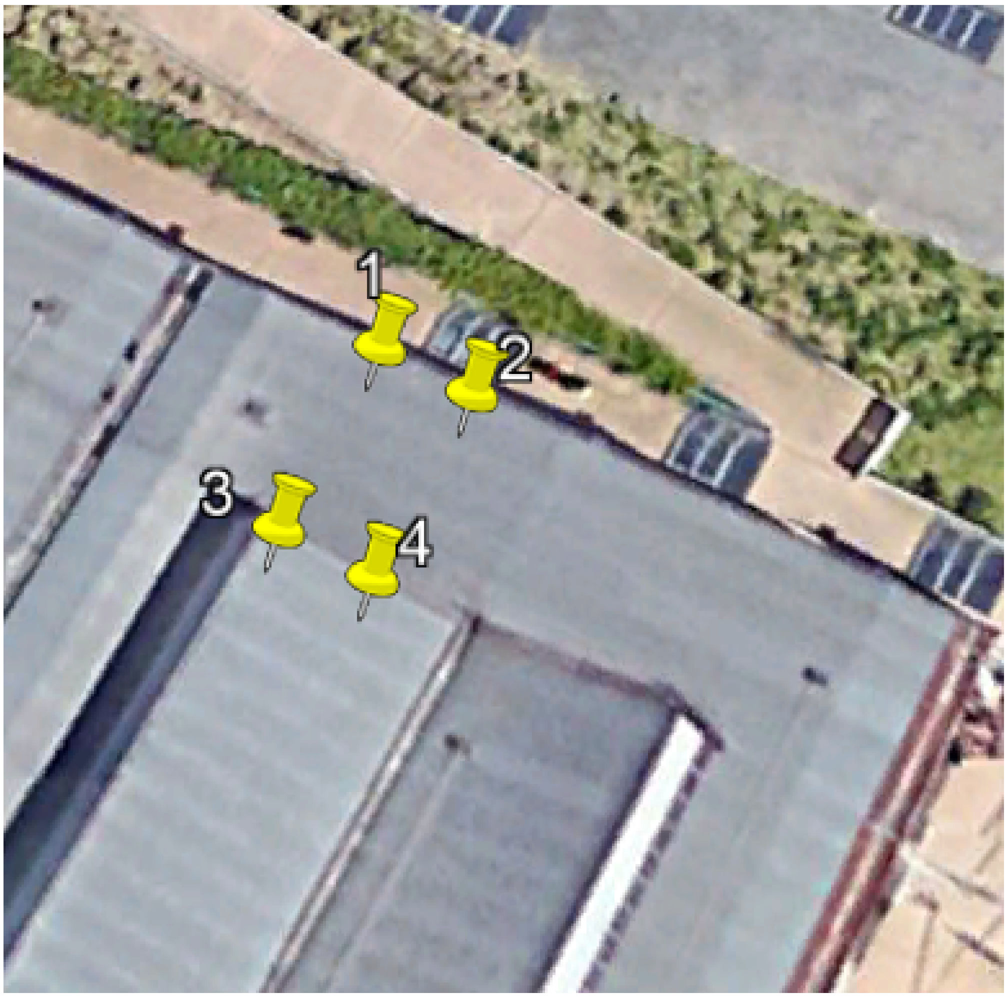
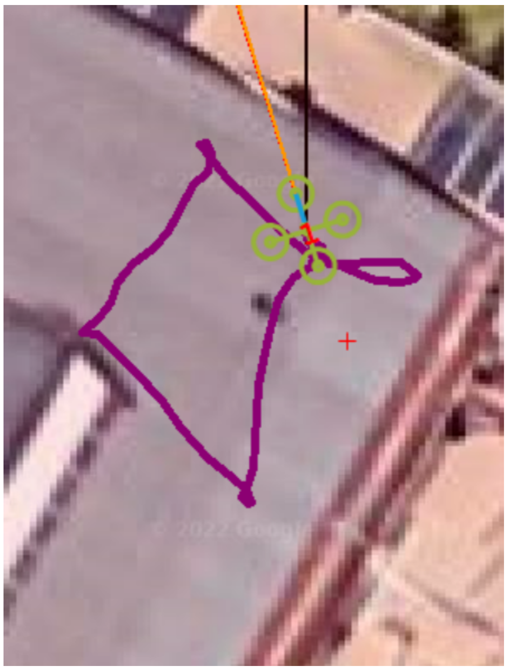


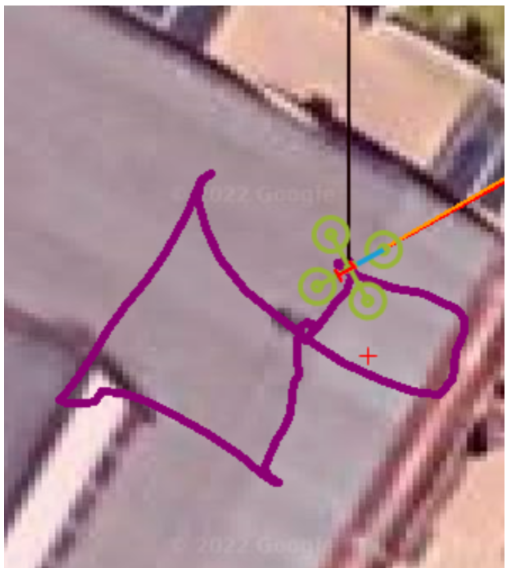
Figure 5.1: System configuration: each yellow pin represents the relative anchor position on the map

To start, the drone is displayed on the map at its initial position, then it is moved by hand by an operator around the local reference system. In Mission Planner the drone path is visualized and can be compared with the actual one. In figure 5.2, the UAV follows the perimeter of the anchors with two different paths.

In figure 5.2a the copter is moved along the perimeter, it can be noticed that in the top right, there are some disturbances that affect the localization system. The path is way more clear in figure 5.2b, where the critical area is avoided, passing by the center of the anchors system.



(a) Path 1



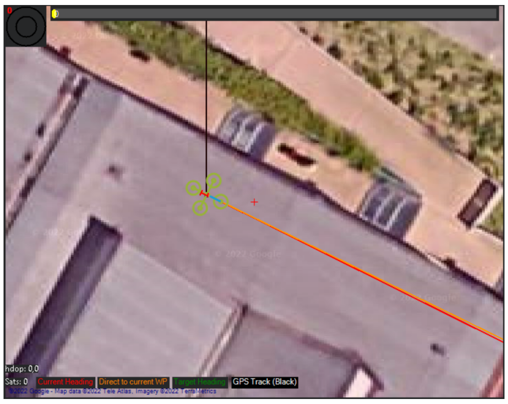
(b) Path 2

Figure 5.2: Localization tests

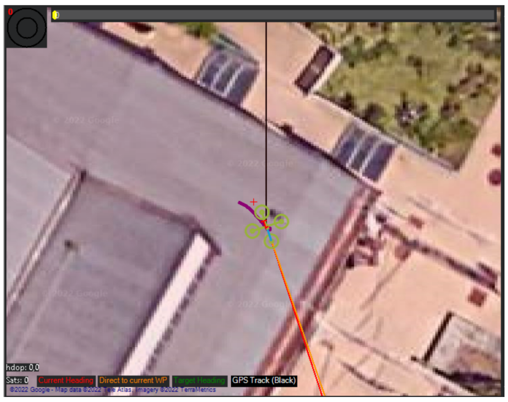
This previous attempt returns a first problem. It can be noticed that even if the copter in the reality is placed at the center of the squared anchor system shown in figure 5.1, its position on the ground station map is shifted to the right. Additional trials are done, initializing the drone from different points inside the perimeter. In this way, it is understandable that this shift is related to the position of the UAV with respect to the first anchor. Image 5.3 shows this dependency.

In figure 5.3a the drone is started under anchor 1, and there are no problems on the origin location setting. In the case represented by figure 5.3b, there is a certain shift with respect to the actual position of the drone, that should be near the waypoint number two.

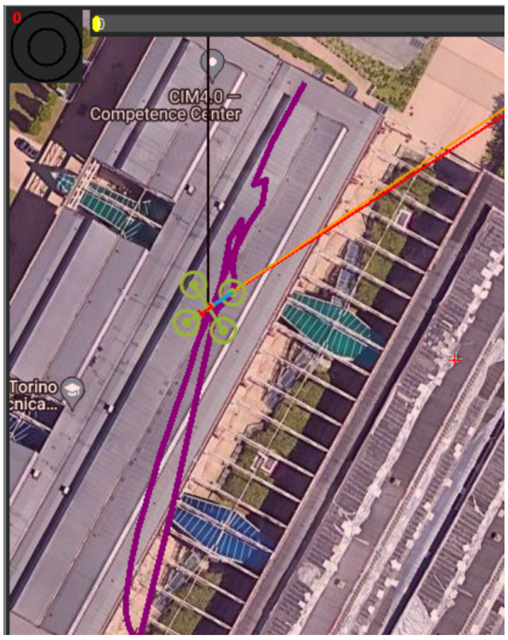
Just like in image 5.3b, figures 5.3c and 5.3d represent two critical cases where the UAV is represented on the map far away from its real position. Moreover, in all the cases the shift error is proportional with respect to the distance from the origin, and the direction of the virtual movement is the same of the one the drone would have if it was moving from anchor 1.



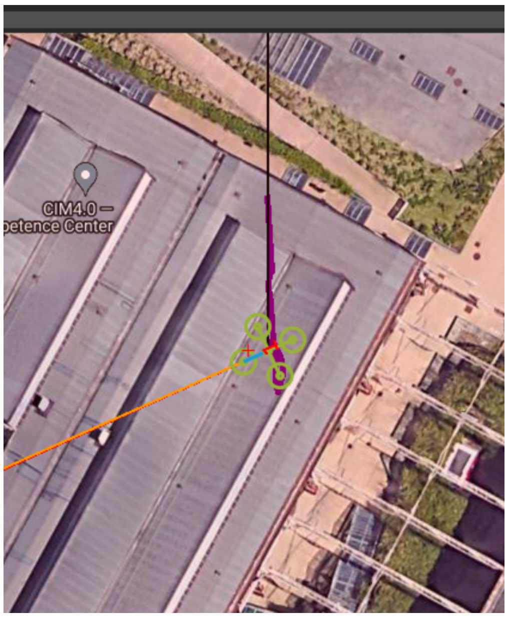
(a) Shift error when near anchor 6805



(b) Shift error when near anchor 680C



(c) Shift error when near anchor 6823



(d) Shift error when near anchor 683C

Figure 5.3: Shift errors

Despite this origin shift, the movements of the copter after the initialization are correct. The previous problem is not considered in a first moment, since is something that affects the user interface and the return to launch automatic action rather than the in-flight performances. As a consequence a couple of flights are tried just to understand the behavior of the copter. The mission difficulty will increase step by step. Initially, the UAV has to take off, reach two meters altitude and suddenly land. This, is more a way to check that the motors work and the communication between autopilot and ground station is good.

Then, the drone has to take off, reach two meters of altitude, maintain the position for a few seconds, and then land. This last mission is repeated several times. Unfortunately, despite the magnetometer calibration, the indoor magnetic disturbances affect the autopilot and thus, the stability of the flight. This noises are due to the presence of electromagnetic devices, but their intensity is not uniform around the room. Thus, in some points of the space, the UAV has no problem, but in others, the compass variance fed to the EKF is too high, causing the copter to eventually start a toilet bowling behavior. This error is limited in position holding mode (such as Loiter and PosHold), but is critical during waypoints missions, as will be explained later. This is due to the high dynamic requested to the filter. In addition, it can be noticed in figure 5.4 that during take off, the magnetometer has a lot of disturbances with respect to its land period. This indicates a certain dependence on the motors working condition.

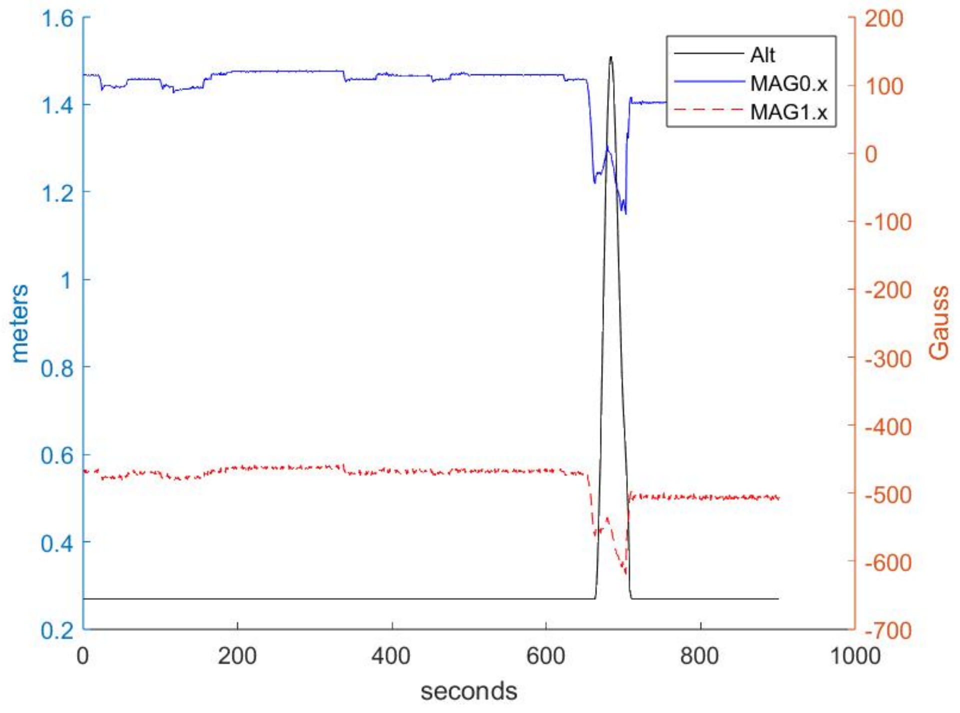


Figure 5.4: Motors magnetic disturbance

5.2.2 Heading

The heading problem of the UAV is further considered. Thus, in this case multiple tests were done. The overall behaviour can be resumed in three main characteristics. As expected, outside, no strong magnetic disturbances are present, thus no problems were registered with the orientation. The observed behavior is that the magnetic field in three different axes oscillates with a maximum variations of more or less 0.05 Gauss between consecutive peaks if the copter maintains the heading, while during yaw rotation, a variation of intensity depending on the angle performed is noticed.
The second trial of tests are organized in a small room, with a cage of 2 meters x 5 meters; in this condition, the magnetometer takes approximately one minute to stabilize, but with even small movements, the stability of the yaw measurement is corrupted by the magnetic disturbances. This leads to a vibration behaviour and to the unstable movement of the drone even in loiter mode. In this room, the magnetic field oscillates more than the 0.05 Gauss needed for a stable yaw alignment. The flight gained benefit in the third case scenario, an indoor environment but bigger with respect to the first one (10 meters squared area) and with less magnetic disturbances; despite the performance improves rather than the previous case, the heading continues suffering from motors action.

Actually, the Pozyx provider suggests a minimum distance between the anchors, also to improve the positioning measurements, thus prefer a bigger operating space should be the optimal solution. Eventually, different areas will be exploited for the final flights and tests. The solutions implemented to improve the performance of the UAV are described in next chapter.

Chapter 6

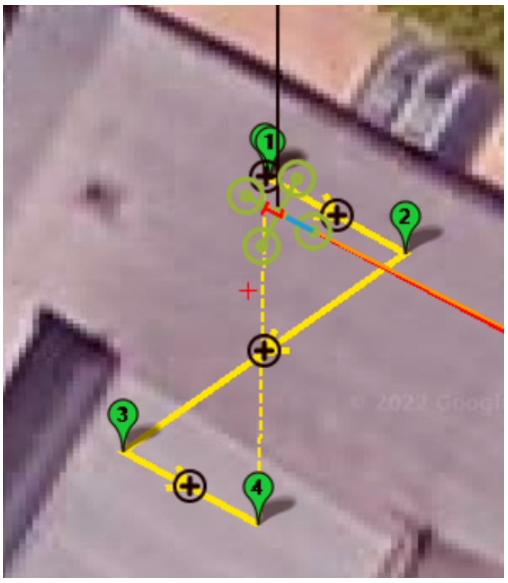
Methodology and implementation

In paragraph 5.2 the first tests highlighted some problems affecting the UAV performances. In this chapter are developed the possible solutions to improve the drone behavior.

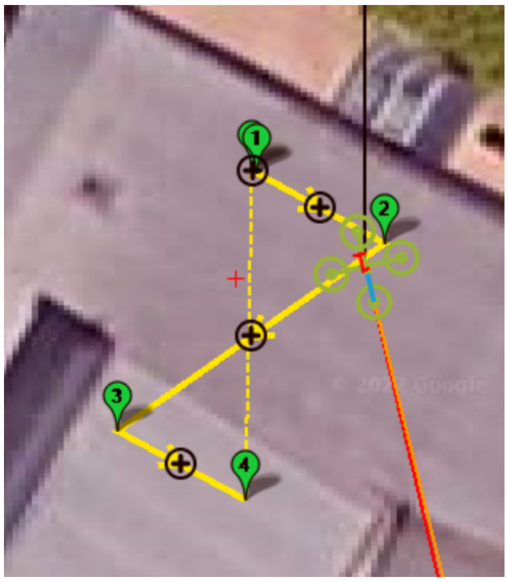
6.1 Origin drift problem

One of the difficulties that the tests presented is the origin shift of the drone during set-up. This condition is shown in 5.3. More in details, the UWB tag needs more or less 30-40 seconds to setting up and start collecting measurements, nonetheless, during this amount of time the drone filter is working and tries to localize itself in the space. Since neither GPS nor UWB is available, the copter gives priority to a dead reckoning system for positioning, causing the initial origin drift. To solve this error, it is necessary to analyse the ArduPilot code. The reference libraries are the ones concerning the beacons implementation and the fusion of UWB measurements into the EKF. These were described in paragrph 3.2.2. Some modifications are tried, first acting on the AP_Beacon library, then on the AP_NavEKF3_RngBcnFusion.cpp method. Unfortunately, nothing of these solved the problem. Thus, several other attempts are made, like changing the origin offset manually, or imposing the global coordinates computed exploiting Google Earth programs. Still, none of these worked. Another attempt was made changing the reference for the AHRS of ArduPilot. This orientation system is based on the Extended Kalman Filter. However, there are two versions of this filter. The newest, is the so called EKF3, which is the stable one for developers. Although this filter is the most updated, it has the mentioned problem for what UWB positioning is concerned. Thus, the previous version, EKF2, is exploited.

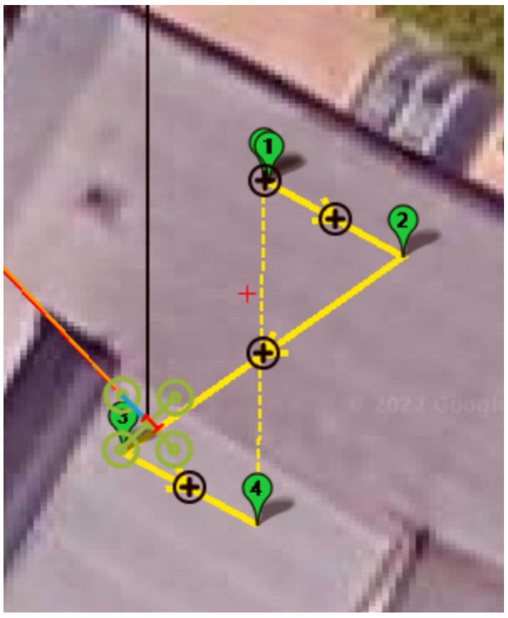
The results are actually quite good.



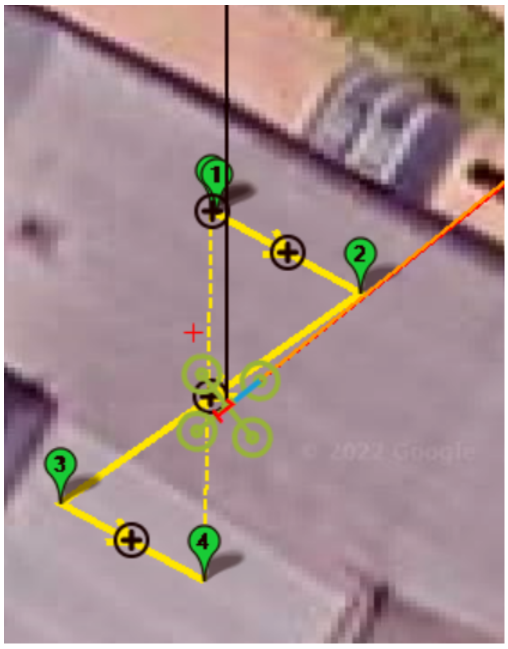
(a) Situation near anchor 6805



(b) Situation near anchor 680C



(c) Situation near anchor 6823



(d) Situation near the center

Figure 6.1: Shift errors solved

Compared to images 5.3, the new conditions in figures 6.1 look way better. This should help the drone performance during the mission, plus it is a critical aspect for monitoring condition; indeed the human operator now has a clear idea of the actual position of the UAV on the map.
Nonetheless, downgrading from the EKF3 to the EKF2 is not a light operation. The first one has some improvements in multiple aspects. Moreover, it can work with many additional sensors and functionalities that EKF2 cannot support. This aspects are not relevant for this project, however, it could be interesting to solve the drift directly on the latest filter version, looking at the difference with the EKF2.

Eventually, the EKF3 turns out to have an additional feature compared to EKF2. This is an automatic and repeated origin computation done by the filter. Actually, there is a function in the AP_NavEKF3_core.cpp file called $moveEKFOrigin$, which is used to move the EKF origin to the current position at 1Hz, while the public origin doesn't move. This function is called as soon as the GPS measurements are available, and it is needed to keep distortions due to spherical shape of the Earth to a minimum. Anyway, in this project GPS is not used, so the EKF origin should be defined manually via GCS. Since the origin marker is not easy to establish manually, the $moveEKFOrigin$ function is deleted. In this way, the drift problem is solved, and the EKF3 is maintained rather than EKF2, with all the benefits of the new version.

6.2 Heading problem

The second documented difficulty during the first tests is the one related to the magnetometer. The presence in indoor environments of devices such as monitors, PCs, other robots and special machines, in addition to the walls, is very critical for the compass behavior. Moreover, the UAV is known to be highly affected by the on-board electronics; in fact, the magnetometer should be placed somewhere far from the motors, ESCs and battery cables. Mission Planner presents a technique for the computation of magnetic interference due to the current flowing in the propulsion devices. This interference depends, as already said, by the proximity of the magnetometer to electromagnetic sources and the type of this elements. In fact, the motors of the smaller quadcopter has an higher KV than the ones of the octocopter; this means that the amount of current flowing during throttle is higher in the first one, causing a bit more interference for singular motor. In order to try to improve the heading computation, different solutions are exploited.

6.2.1 Magnetic Calibration

The first and more obvious thing to do is to calibrate the magnetometer each time the flying space is changed. This is achieved using the MP calibration method. The algorithm fixes autonomously the magnetic offsets, however, the order of relaxation of these elements can vary, making the calibration more or less strict. Furthermore, MP has an advanced set-up function called "CompassMot" which determines the amount of magnetic interference affecting the UAV referring to the current drawn by the system (linear dependence). At the end of the calibration, new offset are applied to the filter, in order to account as much as possible for the interference.

6.2.2 Double Tag Heading

As noticed during the tests, even if a magnetometer is calibrated it is not sufficient for providing reliable heading in indoor environments. Thus, the idea is to exploit the UWB to achieve orientation. More in detail, the reference is the virtual axis starting from the anchors origin and pointing to anchor 0x6823. The second axis, is the rotating one, and it is identified as the virtual line connecting the master tag 0x6833 with the slave tag 0x6815. These tags are placed on the drone, with a fixed distance of $16cm$. The heading is the angle between the reference axis and the one related to the tags. Obviously, with this method the orientation is not subjected anymore to the magnetic disturbances characterizing the indoor environment. On the other hand it is affected by the precision of the UWB positioning and ranging. As noticed, the original precision in positioning is about $10cm$, which is already a critical value for the final results. This because the Double Tag Heading (DTH) algorithm is based on the distances between the two tags. As a consequence, if the tags were at at least $150cm$, no problems would occur; unfortunately, both devices must lay on the UAV, so the maximum distance could be $45cm$ in the octocopter, causing some problems on the measures.

Initially, the DTH algorithm consisted in computing the local x, y coordinates of each tag using the *Pozyx.doPositioning* and *Pozyx.doRemotePositioning* functions; the heading angle ψ was obtained from:

$$\psi = \arccos\left[\frac{y_1 - y_2}{d}\right] \tag{6.1}$$

In 6.1 y_i are the y-coordinates of the master tag (y_1) and slave (y_2), while d is the manually measured distance between the tags. In this way, ψ was always defined between [0,180] Deg. To solve this error, the final yaw was declared as ψ in 6.1 if $x_1 <= x_2$, else the yaw was $360 - \psi$. As commented in the previously, this simple code didn't not work when the two tags were placed quite near to each other. This because of the measurement error: often the algorithm did not even return a heading angle since in the arcosine argument in 6.1 the measured numerator was

bigger than the denominator.
To solve this problem, the first approach required to use the distance between tags computed by Pozyx functions doRanging and doRemoteRangin instead of the one measured manually. Unfortunately, this didn't lead to a better result.
Another solution tries to automatically correct the difference between measured spatial coordinates, so that the algorithm returns always a heading. This is implemented considering the first tag position, computed by the Pozyx algorithms. Such position is intended as the center of a circumference of radius the actual manually measured distance between the tags. At this point, the second tag position is obtained through Pozyx methods, but this point, is projected to the first tag circumference. In this way, the arcosine's argument problem was solved. However, the algorithm didn't behave very correctly, this always because of the precision of the position measurements. In fact, even if the second tag position is projected to the right radial distance from the master tag, if it is too shifted, the heading angle will results up to 40 degrees different from the real one. These results, clear up the fact that this solution cannot be the main tool for heading estimation; something more powerful is needed.

6.2.3 Pozyx AHRS

The attitude and heading reference system has been introduced in section 1.1. This system is implemented to calculate the orientation of the drone in the space. An AHRS is already present in ArduPilot firmware, however, because of the problems concerning the heading, a new AHRS, running on the Arduino board and exploiting the Pozyx tag information, is implemented and tested. Generally, such systems are realized using very complex filters like Kalman or Extended Kalman ones, but in this case, from the moment the final algorithm will be launched through an Arduino UNO board, a custom Complementary Filter is preferred. This type of filters has been explained in paragraph 1.2.3; briefly resembling that content, complementary filters combine low pass filters and high pass filters. In fact, the first are used to operates on high frequency signals (such as vibrating accelerometers) while the second are used to cut low frequency signals (such as gyroscopes drfits).
The following AHRS is made of three parts:

1. Roll and Pitch calculation;
2. Yaw calculation;
3. Quaternion filter.

The first two parts are used to obtain a first filtering action on Euler angles. The latter, on the other hand, is used to obtain the orientation of the drone by means of quaternions, apllying an additional filter action. The input data for this program are the raw_sensor_data collected by the Pozyx tag sensors.

Roll and Pitch

For the moment, the main interest is to create a good AHRS, thus the body frame will be simply considered as the frame fixed on the Pozyx tag, without no further translations. Thus, the Roll angle is defined as one describing the rotation around the UWB tag x-axis, while the Pitch angle is the one related to the rotations around the tag y-axis. These angles are obtained using the ComputeRollPitch function, by passing as parameters the accelerometer and gyroscope raw data. For both Roll and Pitch, a suited complementary filter is realized. Roll (θ) can be obtained from the accelerations on the y and z axes or from integrating angular velocity around x axis:

$$\theta_{accelerometer} = \arctan 2(a_y, a_z)\left(\frac{180}{\pi}\right) \tag{6.2}$$

$$\theta_{gyroscope} = \theta_{gyroscope} + \left(\frac{-g_x}{16.0f}\right)\left(\frac{dt}{1000}\right) \tag{6.3}$$

In previous equations, a_y, a_z, g_x where accelerometer and gyroscope raw data, while dt is the time between consecutive operations defined in Arduino as $dt = millis() - last\_millis$ where millis() is the actual time instant, last_millis is the previous time instant. $16.0f$ is one the conversion factor discussed in paragraph 4.1.

The final Roll angle is obtained by means of the filter action in equation 6.4 , which is characterized by the parameter α_{roll}. The higher is this parameter, more the final Roll will be computed based on the gyroscope instead of the accelerometer.

$$\theta = \alpha_{roll} * \theta_{gyroscope} + (1 - \alpha_{roll}) * \theta_{accelerometer} \tag{6.4}$$

The behaviour is tested using the serial plotter of Arduino.

For what Pitch (ϕ) is concerned, a similar procedure to the one of the Roll will be followed. First, the accelerometer contribution is considered, accounting for all three axes contributions:

$$\phi_{accelerometer} = \arctan 2(a_x, \sqrt{a_y^2 + a_z^2})\left(\frac{180}{\pi}\right) \tag{6.5}$$

While the computation with angular velocities is:

$$\phi_{gyroscope} = \phi_{gyroscope} + \left(\frac{-g_y}{16.0f}\right)\left(\frac{dt}{1000}\right) \tag{6.6}$$

Eventually, the Pitch filtered is obtained using a specific parameter called α_{pitch} for the weighting of the two inertial contributions:

$$\phi = \alpha_{pitch}\phi_{gyroscope} + (1 - \alpha_{pitch})\phi_{accelerometer} \tag{6.7}$$

Yaw

Yaw angle (ψ), is obtained from magnetometer measurements on the three axes combined with the already computed roll and pitch angles. Raw magnetic data are first corrected with the biases obtained from the calibration process in section 4.3, then converted using the apposite constants listed in 4.1 and finally used as described by the following equations.

$$\psi_{magnetometer} = \arctan 2(-b_y, b_x)\left(\frac{180}{\pi}\right) \tag{6.8}$$

In details:

$$\begin{aligned} b_x &= m_x \cos(\theta) + m_y \sin(\theta)\sin(\phi) + m_z \sin(\theta)\cos(\phi) \\ b_y &= m_y \cos(\phi) - m_z \sin(\phi) \\ b_z &= -m_x \sin(theta) + m_y \cos(\theta)\sin(\phi) + m_z \cos(\theta)\cos(\phi) \end{aligned} \tag{6.9}$$

However, Yaw measurements are improved considering also the gyroscope data. This is accomplished with a new complementary filter described in 6.10.

$$\phi = \alpha_{yaw}\psi_{gyroscope} + (1-\alpha_{yaw})\psi_{magnetometer} \tag{6.10}$$

$$\psi_{gyroscope} = \psi_{gyroscope} + \left(\frac{-g_z}{16.0f}\right)\left(\frac{dt}{1000}\right) \tag{6.11}$$

Once completed the previous implementations, the filters must be tuned correctly, this is done with a trial and error evaluation, changing the $\alpha_{roll}, \alpha_{pitch}, \alpha_{yaw}$ parameters.

Quaternion filter

Euler angles have some limits that can cause errors in terms of precision. Thus, the orientation of a robot is often provided in terms of quaternions. In this case, a more complex analysis is done starting from Euler angles obtained as described in the previous paragraphs and exploiting an additional complementary filter based on $\alpha_{quaternion}$ parameter. This filter behaves similarly to the one in 6.10 but instead of the Yaw, the action has the quaternion as subject. The first component of the attitude is given by the dynamic quaternion $\boldsymbol{q}_\omega$ computed with the gyroscope added to the previous orientation $q_{t-1} = [q_w, q_x, q_y, q_z]^T$, in formulas:

$$\boldsymbol{q}_\omega = \begin{pmatrix} 1 & -\frac{dt}{2}\omega_x & -\frac{dt}{2}\omega_y & -\frac{dt}{2}\omega_z \\ \frac{dt}{2}\omega_x & 1 & \frac{dt}{2}\omega_y & -\frac{dt}{2}\omega_z \\ \frac{dt}{2}\omega_x & -\frac{dt}{2}\omega_y & 1 & \frac{dt}{2}\omega_z \\ \frac{dt}{2}\omega_x & \frac{dt}{2}\omega_y & -\frac{dt}{2}\omega_z & 1 \end{pmatrix} \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \tag{6.12}$$

Then, there is the computation of the static quaternion, from the Euler angles computed before:

$$\boldsymbol{q}_{am} = \begin{pmatrix} \cos(\phi)\cos(\theta)\cos(\psi) + \sin(\phi)\sin(\theta)\sin(\psi) \\ \sin(\phi)\cos(\theta)\cos(\psi) - \cos(\phi)\sin(\theta)\sin(\psi) \\ \cos(\phi)\sin(\theta)\cos(\psi) + \sin(\phi)\cos(\theta)\sin(\psi) \\ \cos(\phi)\cos(\theta)\sin(\psi) - \sin(\phi)\sin(\theta)\cos(\psi) \end{pmatrix} \tag{6.13}$$

Finally, the orientation is given by:

$$\boldsymbol{q}_t = \alpha_{quaternion}\boldsymbol{q}_\omega + (1 - \alpha_{quaternion})\boldsymbol{q}_{am} \tag{6.14}$$

At the end, unfortunately, this Pozyx AHRS system had the same problem of the one of the autopilot. In fact, due to magnetic disturbances indoor it is even worst than the ArduPilot. Thus, other solutions should be considered.

6.2.4 Lane Switching

The possibility to keep the EKF3 rather than EKF2 brings some advantages in terms of performances. Besides, the EKF3 instantiates multiple cores of the filter called "lanes". The primary one is used for the state estimate, the rest remains in the background and ready for switching to. The main purpose is to have multiple sensors working in separate ways but on the same subject, so that the filter can select each time the one with the best performance. However, the number of possible lanes is equal to the number of available IMUs, this means that such feature can be exploited just on the Cube Orange autopilot and not on the Pixhawk one. The lane switch is implemented so that each IMU refers to a different external magnetometer. This should partially solve the heading problem from the moment that the lane change resets the compass covariance error.
Figure 6.2 shows that when the difference between the magnetic strength computed by the two magnetometers is greater than a threshold value (in this case fixed to 0.1 Gauss) the autopilot switches the main lane. When the primary lane is stabilized again, it is restored. Nevertheless, this solution is not as good as it should be. This implementation depends on several aspects. Using a relatively low threshold (such as in this case), makes the filter more dynamic in avoiding the magnetic interference, still, this is risky if the second IMU has a low quality magnetometer. In this project configuration, the two sensors have different characteristics, thus switching from the RM3100 and the HMC833L is not always safe. In order to improve this fact, two identical and well performing sensors should be used together in the future.

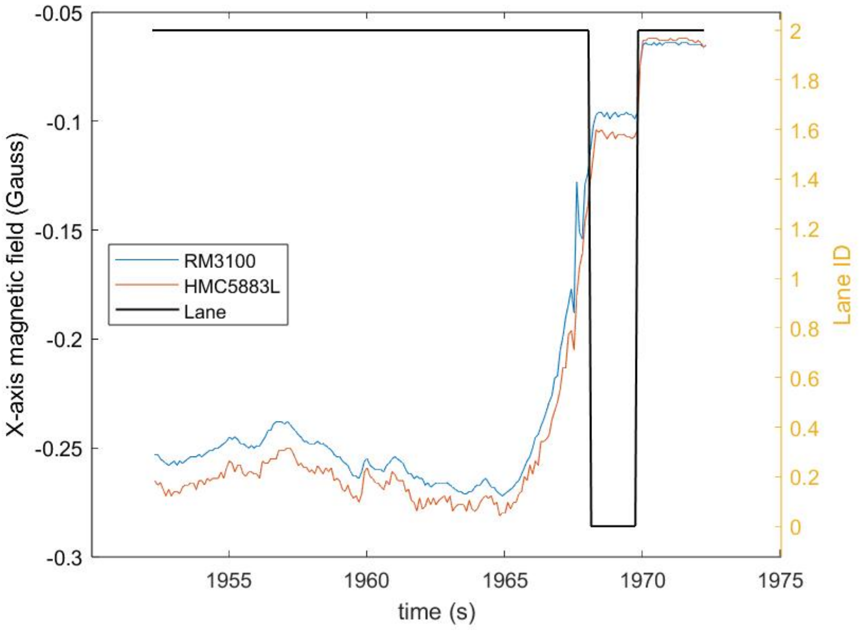


Figure 6.2: Lane switching between two autopilot cores

6.2.5 Filter Customization

Eventually, the magnetometer interference can be handle in a more custom way. The AHRS of the autopilot is provided using the EKF. In paragraph 1.2.1 this filter is briefly explained in its fundamentals. About the process and measurement noises (equation 1.13). The measurement noise has covariance R, and is something that can be easily determined depending on the sensor. The process noise has covariance Q, this matrix is tricky to estimate. The best possibility is to realize an algorithm that updates the Q matrix instant by instant. This is not available yet, so the covariance is set in such a way the magnetic distortion in the test room can be reduced. In ArduPilot each element of this matrix can be modified using MP. In particular, the interested parameters for heading are:

1. MAGB_P_NSE: body magnetic process noise;
2. MAGE_P_NSE: Earth magnetic process noise
3. MAG_M_NSE: magnetic measurement noise;
4. BCN_M_NSE: beacons measurement noise.

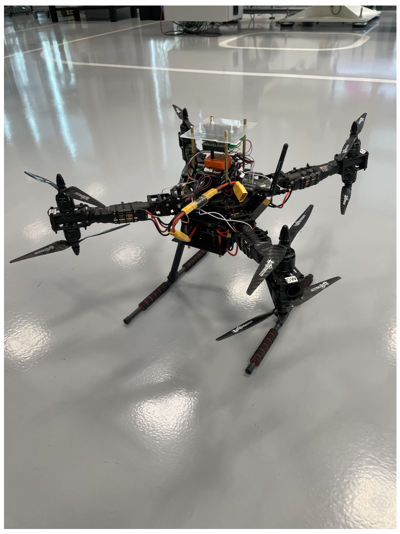
As already said, the EKF combines prediction and measurement. When the prediction is weak (very uncertain, high covariance), the result of the combination

is closer to the measurement. On the other hand, when the prediction is very good (small covariance), the combination is closer to this rather than measurements. First of all, the measurement noise matrix is adapt to the indoor use, this is mainly done analyzing the behaviour of the sensor and with a trial and error method. Referring to paragraph 5.2.2, it was noticed that the indoor compass measurements has wider variations than outdoor, so the MAG_M_NSE is increased as consequence. Then, the process noise is modified so that it can follow the vehicle dynamics. The MAG_P_NSE is decreased. In fact, the copter will fly slowly for two reasons: the first one is for safety, since there could be other people in the room. The second is that the UAV has shown the greatest problem during fast and rapid movements, thus instead of making the filter more reactive, it is decided to make is more accurate but slower.
Eventually, the filter can compute heading in two different ways. Generally, for outdoor mission, the UAV uses the yaw computed from the quaternions for a first evaluation and, once the take-off phase is completed, it switches to a three axes magnetometer innovation. Although, it is well known that indoor the magnetic disturbances are dangerous, thus this last method is not reliable. For sure, the second one is better in terms of performances for a normal use, but the critical conditions imposed by the magnetic interference forced the implementation of the first way for the entire mission. The two algorithms are evaluated, and the practice underlines that the heading computed from the yaw innovation is better and more stable than the one obtained with the magnetic field strength innovations.

6.3 Final set up and experimental tests

Eventually, each UAV is flight in both the CIM4.0 Additive and Digital laboratories. The Additive space consists in an 8x8 meters perimeter. The Digital one is a 4x4 meters area, with more electrical devices, thus with a higher magnetic disturbance. In these tests, the custom version of the EKF3 is used to provide both positioning and attitude. Both RM3100 and HMC8553L compasses are used. The final parameters for the copters in terms of EKF's noise tuning are:

- EK3_BCN_M_NSE = 1.3
- EK3_GYRO_P_NSE = 0.03
- EK3_MAG_M_NSE = 0.15
- EK3_MAGB_P_NSE = 0.00001
- EK3_MAGE_P_NSE = 0.0001
- EK3_YAW_M_NSE = 0.6


(a) Octocopter

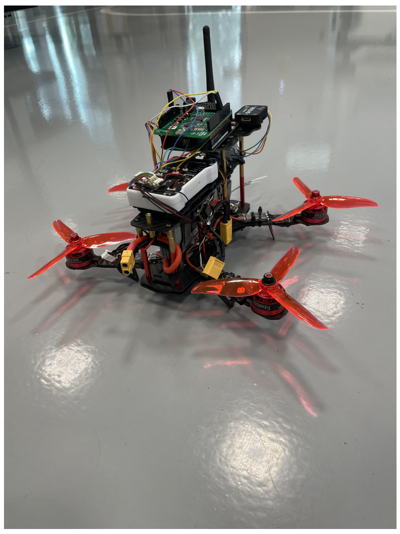

(b) Quadcopter

Figure 6.3: The two final UAVs

6.3.1 Additive mission - Quadcopter

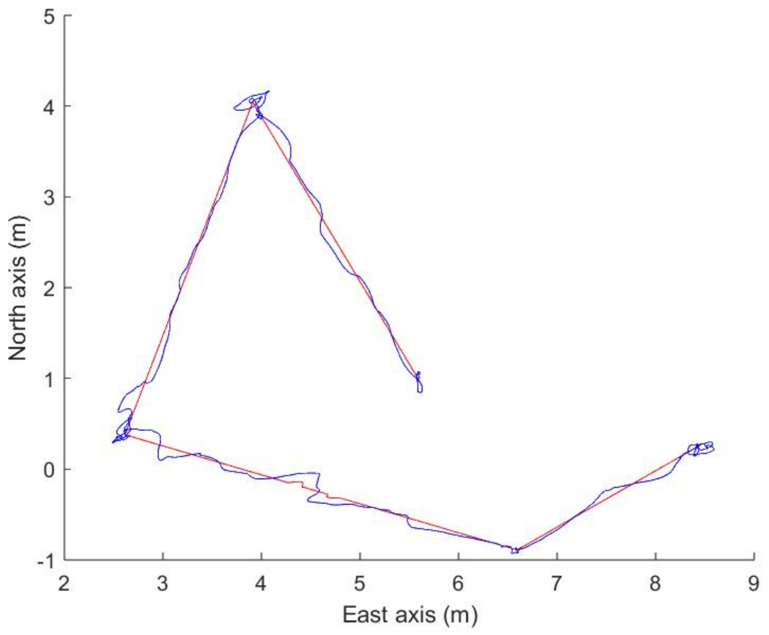


Figure 6.4: Path desired (red) and achieved (blue)

Figure 6.4 resumes the behaviour of the UAV during its autonomous mission. The copter has to follow a few waypoints defined by the operator on the map of the GCS. Such waypoints can be taken either with Google Earth or manually by the human pilot; this means the safety operator can fly the UAV and save the desired positions to be reached during the next autonomous mission, then such data are uploaded into the flight controller to be used. Figures 6.5 and 6.6 show the error between the target position and the estimated one computed by the EKF. The innovation is reported in a clearer way in 6.7 and 6.8. The mean error is less than 5cm, which is actually quite good. On the other hand, figures 6.9 and 6.10 represent the improvement between the starting position computed by the Pozyx and the one estimated by the EKF.

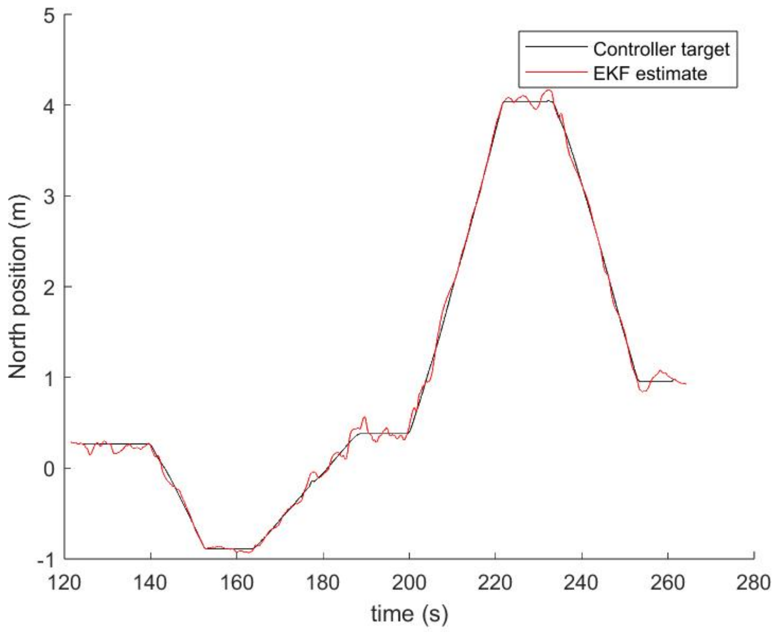


Figure 6.5: North-axis positioning difference

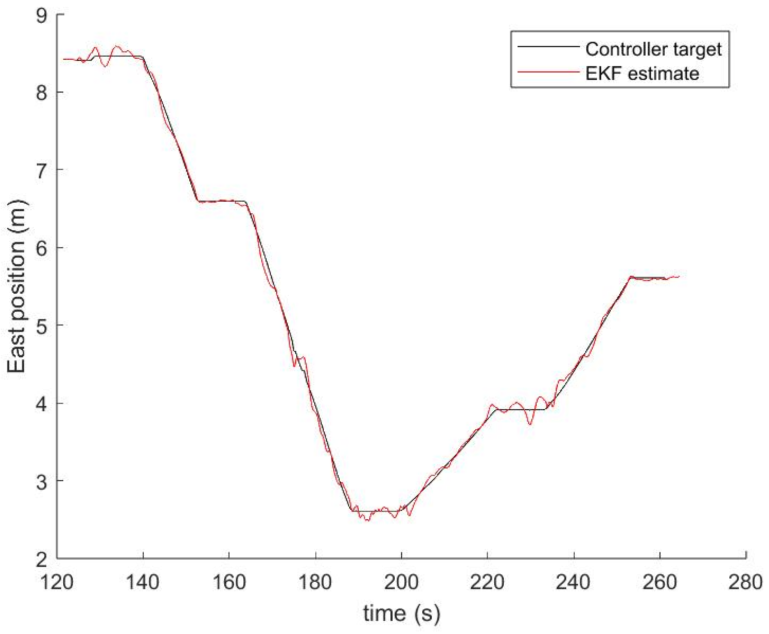


Figure 6.6: East-axis positioning difference

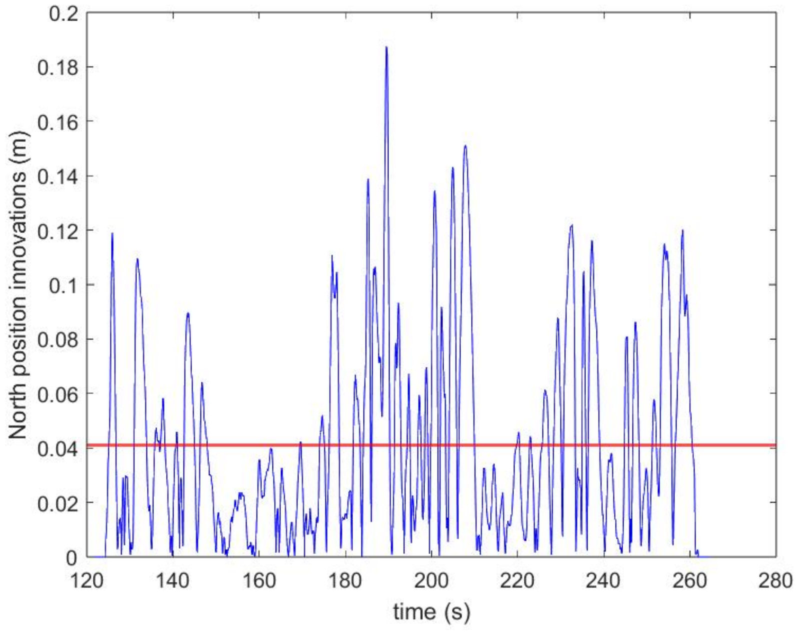


Figure 6.7: North-axis positioning error

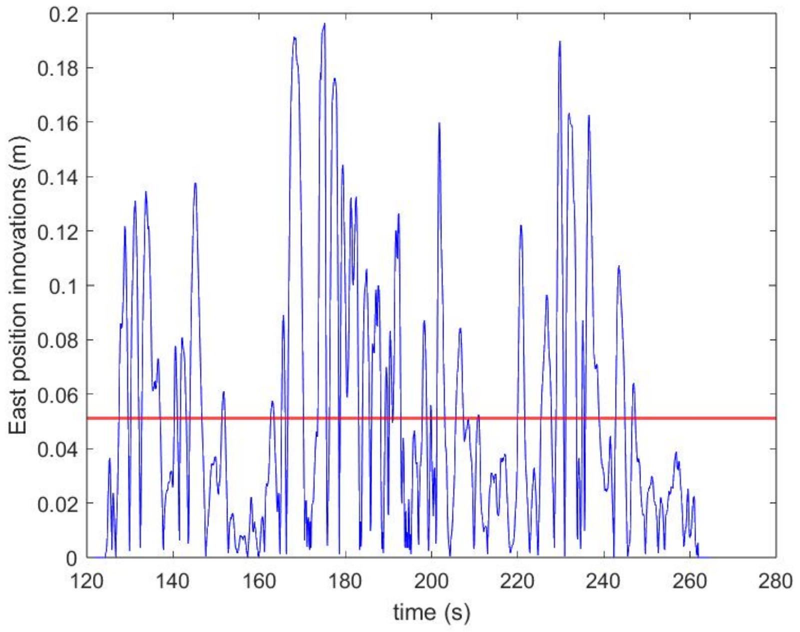


Figure 6.8: East-axis positioning error

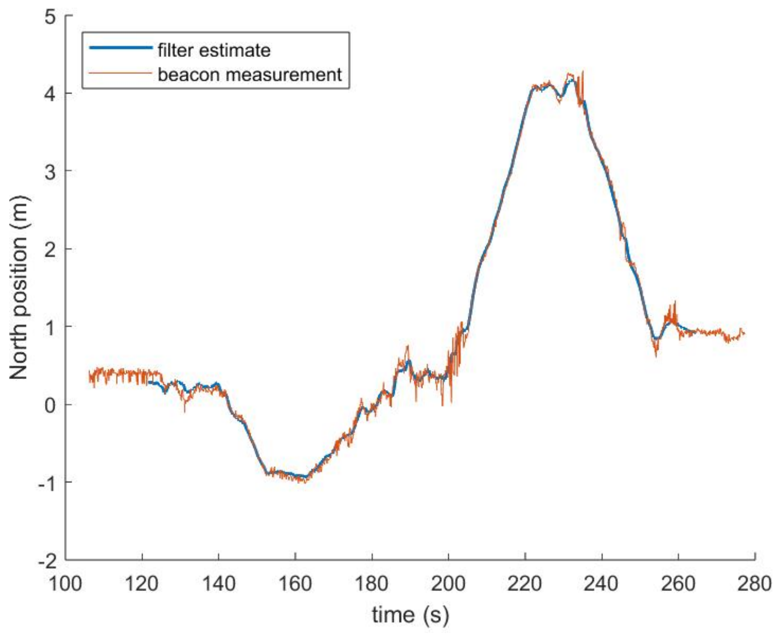


Figure 6.9: North-axis difference between Pozyx's estimate and EKF's one

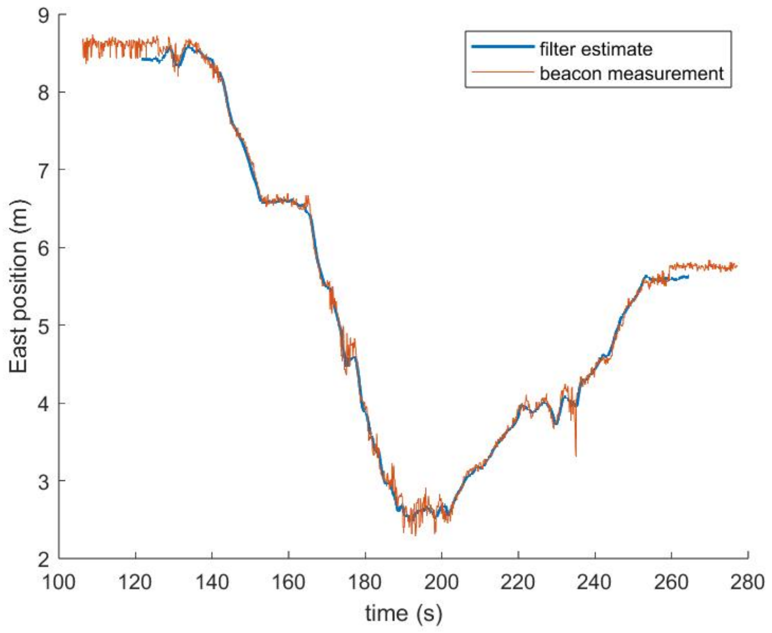


Figure 6.10: East-axis difference between Pozyx's estimate and EKF's one

It can be noticed how the estimate is more linear, which obviously increase the precision of the measure. It would be interesting to calculate the error between the estimated position and the real one in the space; however, this would be possible exploiting advanced systems such as the camera one, not available for this project.

During the flight the Yaw is stable, the UAV is programmed so that it points toward the next waypoint of the mission. The innovation between the target and estimate yaw, showed in 6.11, is less than 15deg. It is important to remember that the lower is the innovation, the better is the flight behaviour. Moreover, the magnetometer test ratio (figure 6.12), that is a parameter of the filter that describes the quality of the innovation variance, is lower than 0.3, which means the measurements coming from the sensors are qualitative good.

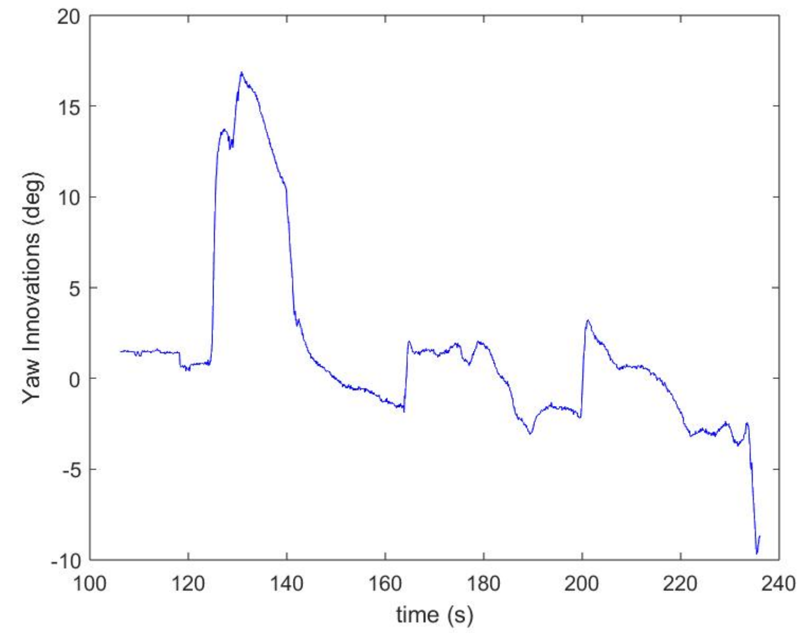


Figure 6.11: Yaw innovations

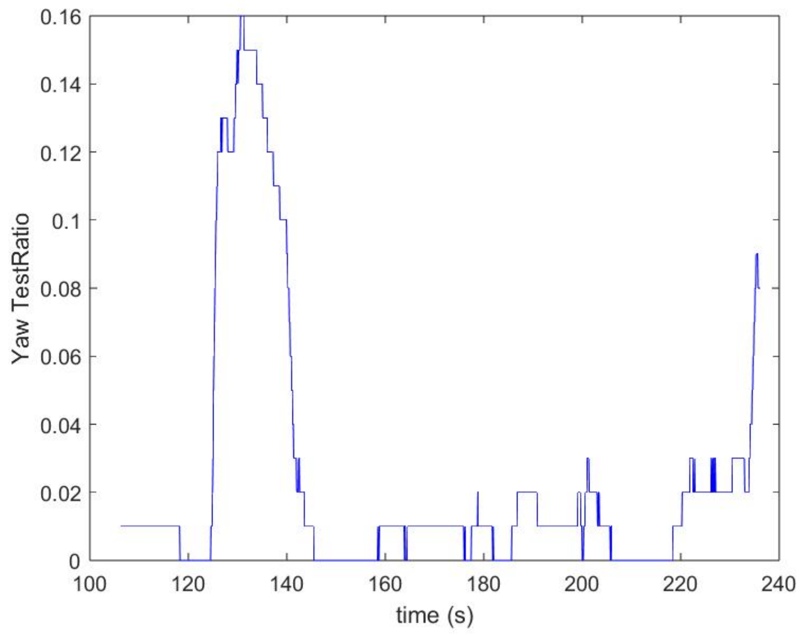


Figure 6.12: Yaw EKF's test-ratio

6.3.2 Digital mission - Quadcopter

A new flight, similar to the previous one is attempted. Its path is represented in figure 6.13.

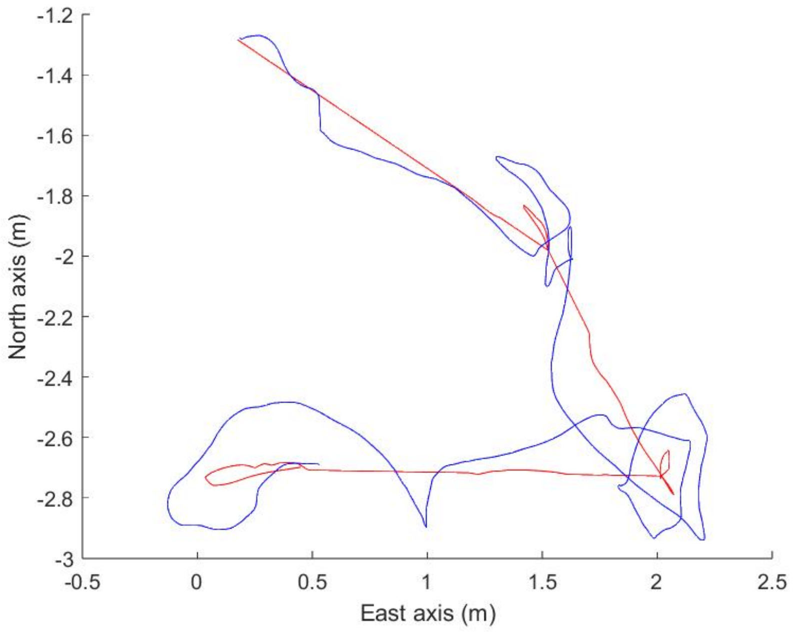


Figure 6.13: Path desired (red) and achieved (blue)

The same considerations done during the Additive flight are repeated in this case. First of all, the difference between the target position of the navigation controller and the estimated one by the filter are compared in figures 6.14 and 6.15.

Although, the UAV has, from a qualitative point of view, similar performances to the one had during the Additive mission. Figures 6.16 and 6.17 analyse the error between the target position and the filter estimate. Compared to images 6.7 and 6.8, the mean error now is around 10cm rather than 5cm. This shows how the environmental conditions in the Digital labs are more critical. Despite this fact, the copter was still able to complete its flight. For what the heading is concerned, the yaw innovations are always less than 15deg, which is an acceptable error. In addition, the yaw test-ratio of the filter is shown in figure 6.19, and is lower and 0.3, which is the classic threshold which indicates good sensor lectures.

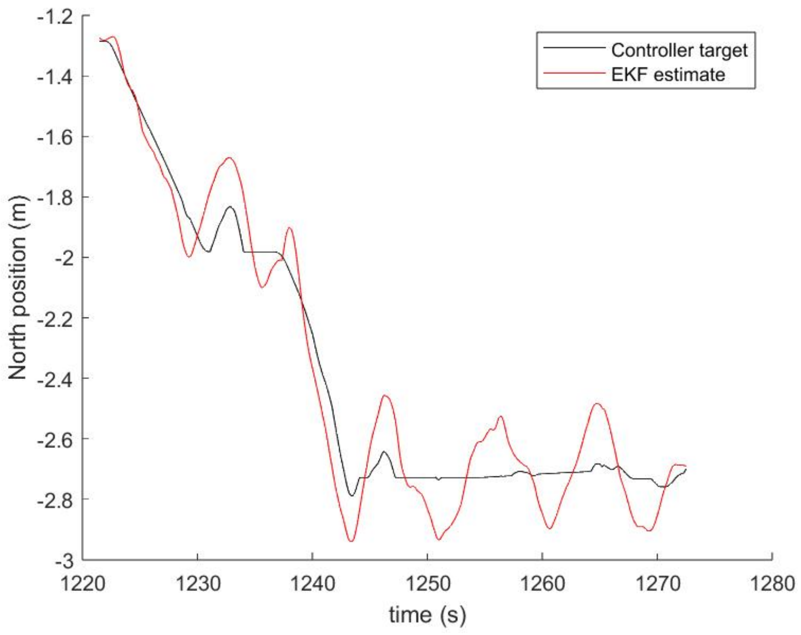


Figure 6.14: North-axis positioning difference

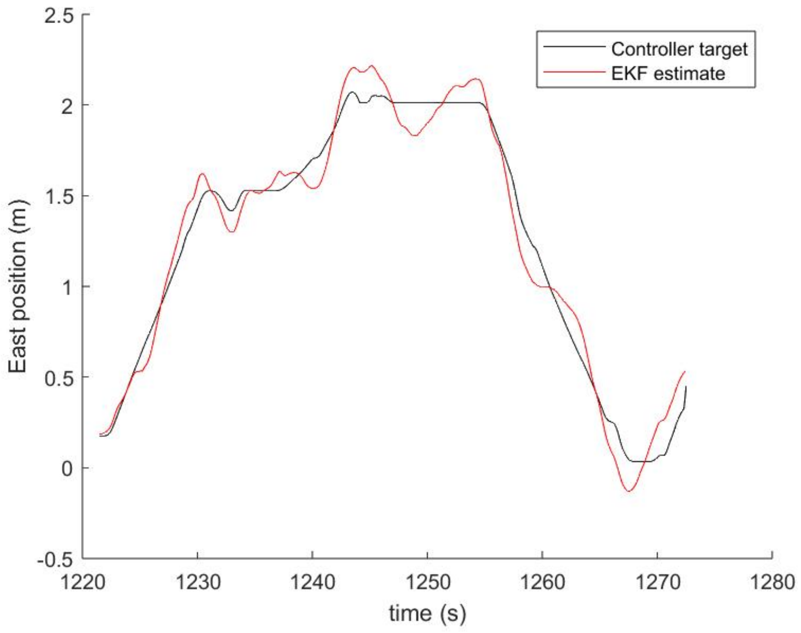


Figure 6.15: East-axis positioning difference

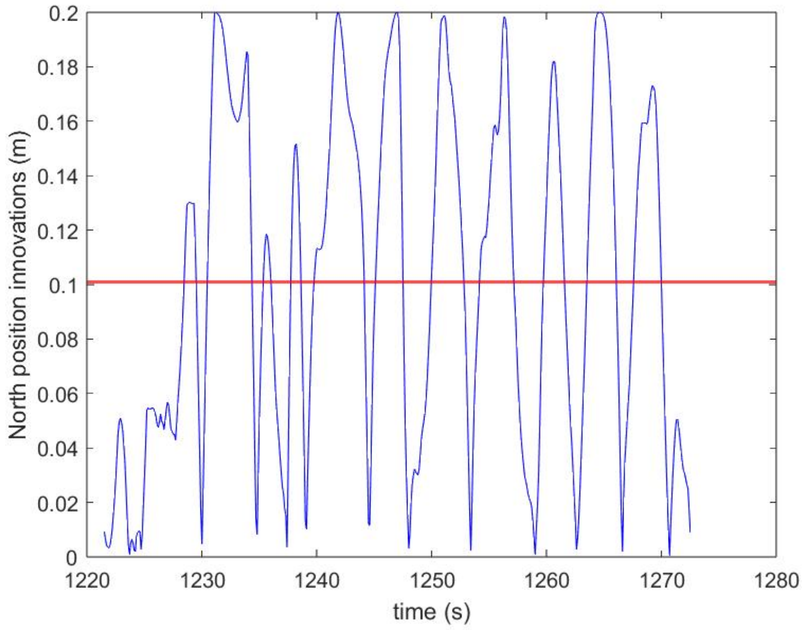


Figure 6.16: North-axis positioning error

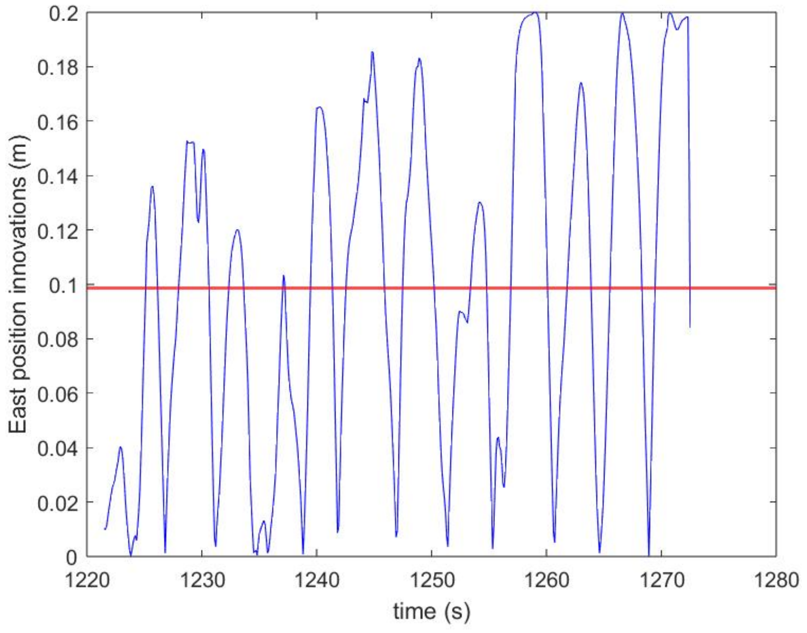


Figure 6.17: East-axis positioning error

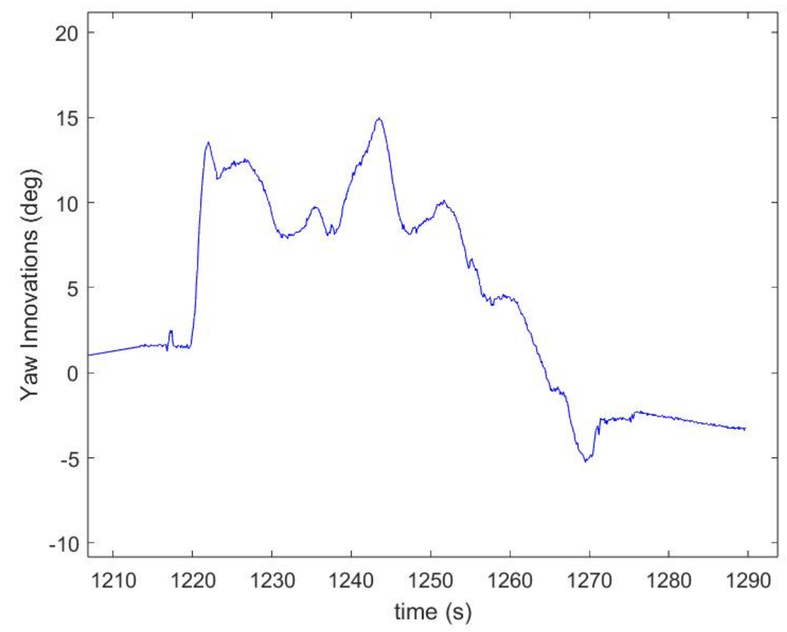


Figure 6.18: Yaw innovations

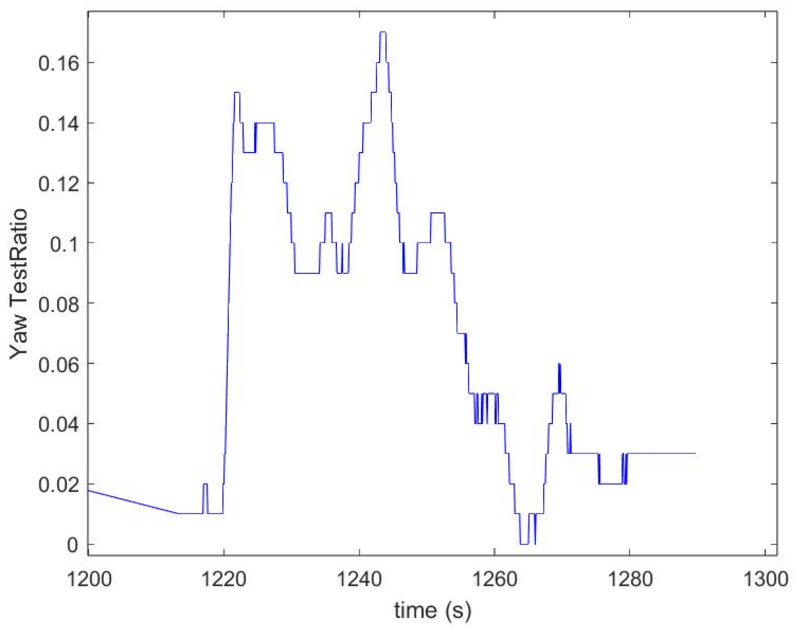


Figure 6.19: Yaw EKF's test-ratio

6.3.3 Additive mission - Octocopter

Finally, also the octocopter is tested. Because of its dimensions, it is just used in the Additive spaces. Just like previous missions, it is tested both in position hold mode and waypoints flight. This UAV has three IMUs, as a consequence, the filter's innovations and test ratios are three: the one of the main core, and the two running in the background. In this case, referring to figures 6.25 and 6.26, it is possible to notice that the error between the target position and the estimated one have very different mean values compared to the previous missions. This could be connected to the heading problem that the octocopter faces during the first phase of the mission. In fact, the UAV is configured with the compass calibration accounting for the motor interference, still it has a problem during the takeoff. In the first thirty seconds, the octocopter should have reached the desired altitude and keep the position for ten seconds. However, in figure 6.20 the first part of the mission has an undesired movement toward North.

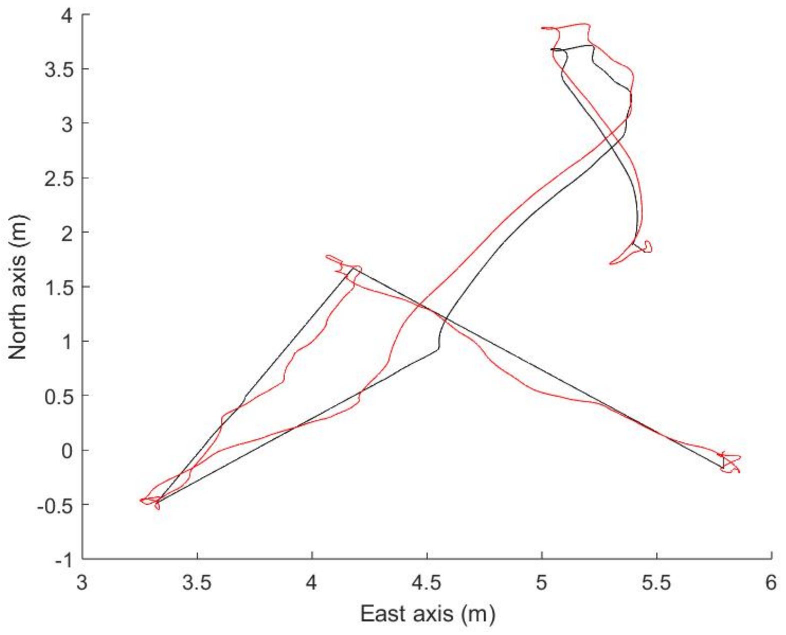


Figure 6.20: Path desired (black) and achieved (red)

This can be seen better in the graphical representations 6.23 and 6.24: in the time interval between 140 and 170 seconds there should have been an horizontal line (such as the ones at 180 and 210 seconds). In particular, observing the East position, there are rapid variations rather than the more uniform characterizing the North position; looking at 6.21 this is caused by the high innovations in terms of Yaw. Everything is confirmed by the message sent by the GCS: "EKF3 IMU MAG ground mag anomaly, yaw re-aligned", which means that during the take off, the magnetic variation affects the yaw, but this change is not felt neither by the gyroscope nor by the accelerometer, thus the heading should be re-aligned between the two sensors. Despite this fact, that causes the UAV to drift of about half a meter from the desired position, once the copter reaches the right altitude, it stabilizes and the next part of the mission is accomplished without any problems. Anyway, the difficulties affecting the heading measures are noticed also since sometimes the test-ratio overcomes the 0.3 standard value in 6.22 and the innovations are wider in 6.21 rather than in the previous flights.

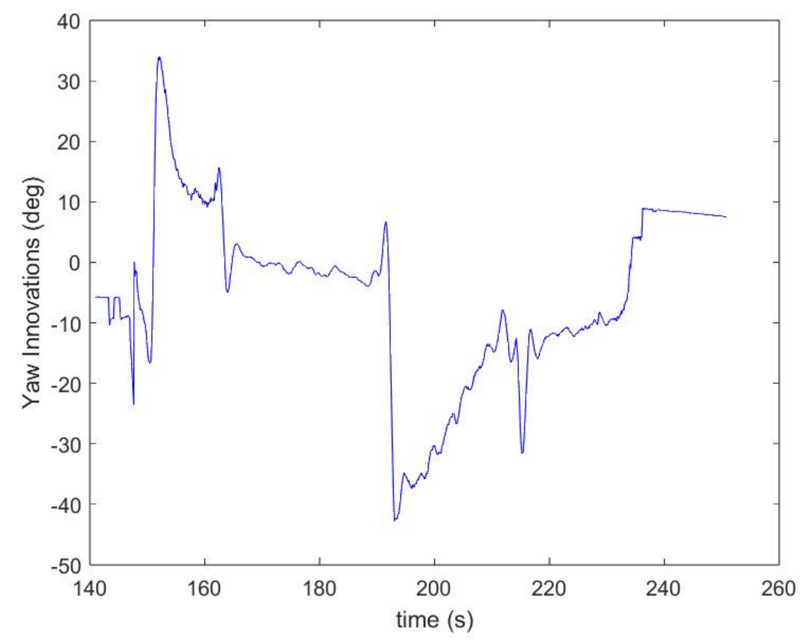


Figure 6.21: Yaw innovations

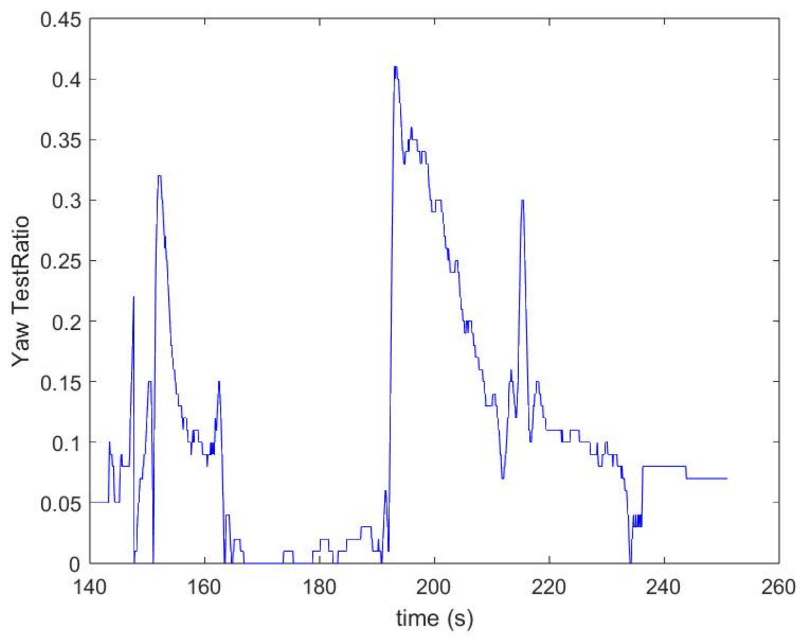


Figure 6.22: Yaw EKF's test-ratio

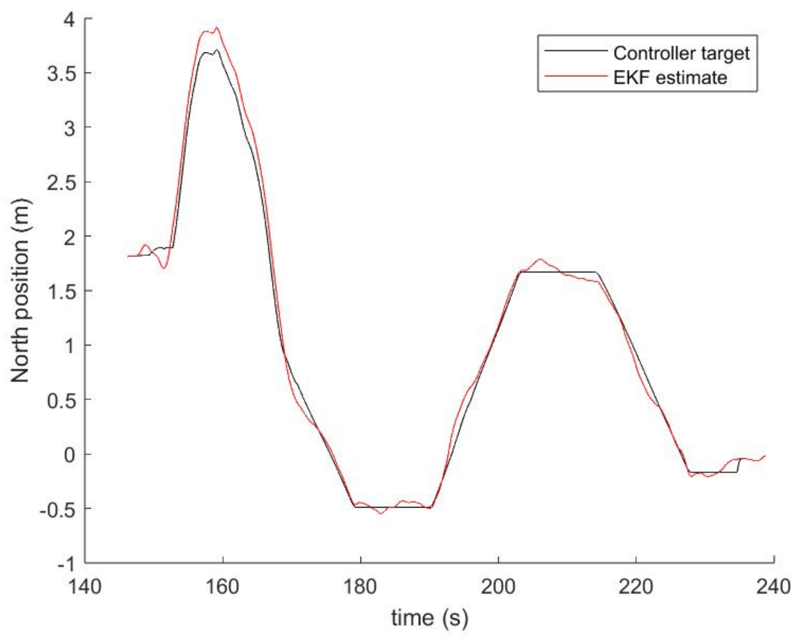


Figure 6.23: North-axis positioning difference

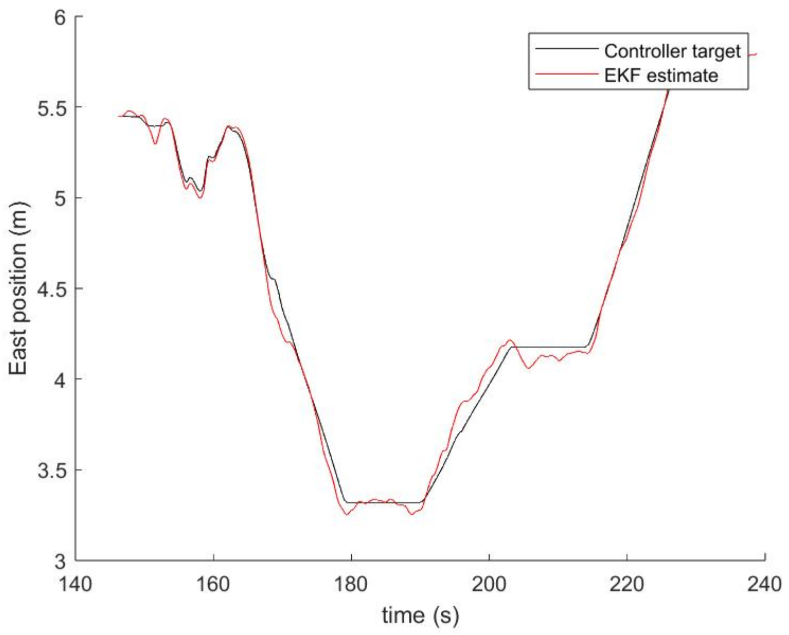


Figure 6.24: East-axis positioning difference

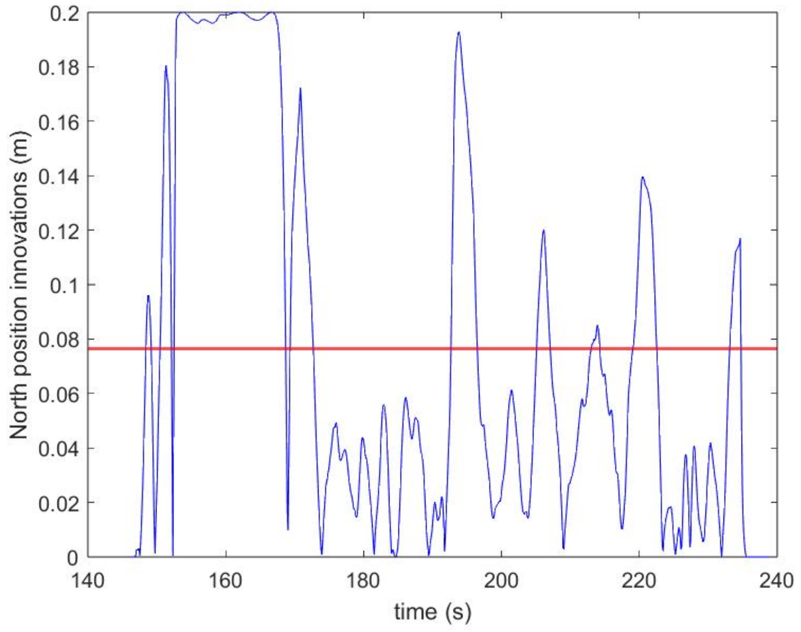


Figure 6.25: North-axis positioning error

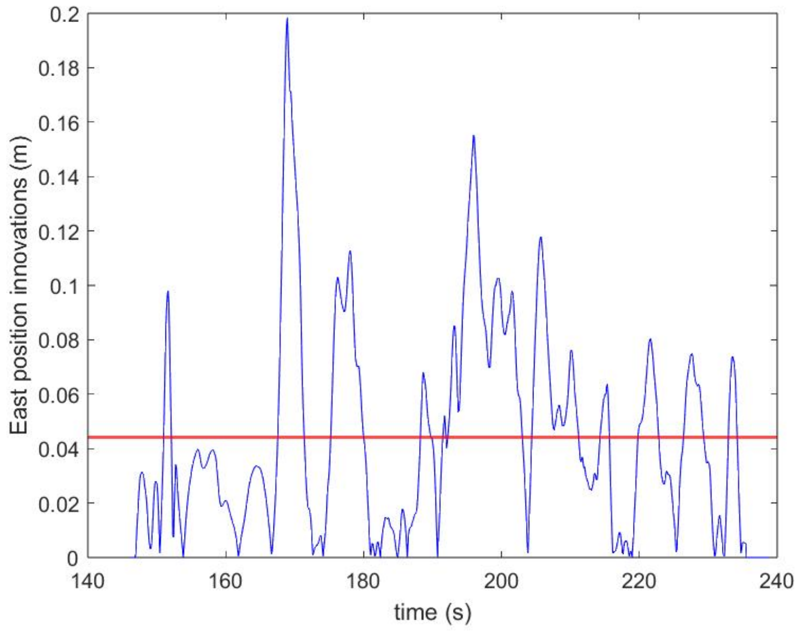


Figure 6.26: East-axis positioning error

Chapter 7

Conclusions

Autonomous robots and vehicles represent the future. Among all, drones have versatile characteristics which make them perfectly fit for many different operations. CIM4.0 wants to investigate the potential of UAV for autonomous flights in industrial environments. The final aim is the realization of a system able to explore the surroundings and which can monitor the space autonomously. The thesis required the realization of a prototype for tests: both a quadcopter and an octocopter are used. Different indoor localization technologies are available, however, Ultra Wide Band (UWB) represents a good trade off between costs and accuracy. In fact, it is relatively cheap compared to vision systems and the Pozyx version offers a 10cm accuracy in positioning. The x,y positions of the UAV are expressed as two elements of the system state vector. These are estimated exploiting an Extended Kalman Filter, which is fed with the ranges between the master tag and each anchor. At the end, the estimated accuracy of the position is lowered from 10cm to 5cm with the quadcopter. A major problem during autonomous flight indoor is the toilet bowling behavior (the UAV starts to follow circular trajectories of increasing diameter) caused by magnetic disturbances due to walls and to the presence of electrical devices (such as computers, monitors, other robots ecc.). In order to avoid this problem a good magnetometer calibration is needed; however, the real solution consists on the perfect tuning of the EKF. More in details, the measurements of the compass are not reliable, thus the filter gives priority to the system model estimation. In other words, the Kalman Gain of the filter is lowered, and this is done increasing the measurement noise covariance and decreasing the process noise covariance. While the small quadcopter flies quite good depending on the amount of disturbances present in the room, the octocopter still presents a problem during the take off. In fact, the motor interference, added to the environment one, causes first a discrepancy between inertial and magnetic sources, and thus, a certain drift of the UAV from the starting desired position. This fact could be solved adopting a more custom and well-designed frame, in order to reduce the negative influence of the motors and battery on the sensors during take off. Eventually, the final performances of the UAVs are good (especially considering that these are prototypes mounted without a well-fitting design), but could be improved in different ways. First, the process noise covariance matrix, can be established continuously each instant; this should improve the filter estimation during more dynamic maneuvers. Furthermore, the drones, in particular the quadcopter, don't have anymore too dangerous problems with the magnetometer. Although, the heading can be provided also using cameras. This solution removes completely the possibility of toilet bowling, but is more expensive. In addition, cameras enable the implementation of dynamic obstacle avoidance algorithms. At the end, for what the localization method is concerned, UWB has interesting features, but it cannot be used for exploration, since this system relies on the anchors. These must be already placed before the flight. Thus, the UWB can be exploited for industrial uses, but the situation gets more tricky in unknowkn environments. In addition, if the system is set in a human-dangerous space, for both UWB and non-onboard cameras (like VICON's motion capture), maintenance is more complex. To avoid this difficulty, other solutions such as Lidars can be exploited.

Bibliography

[1] Xuebing Yuan, Shuai Yu, Shengzhi Zhang, Guoping Wang, and Sheng Liu. «Quaternion-Based Unscented Kalman Filter for Accurate Indoor Heading Estimation Using Wearable Multi-Sensor System». In: *Sensors* 15.5 (2015), pp. 10872–10890. ISSN: 1424-8220. DOI: 10.3390/s150510872. URL: https://www.mdpi.com/1424-8220/15/5/10872 (cit. on p. 6).

[2] Mauricio A. Caceres, Francesco Sottile, and Maurizio A. Spirito. «Adaptive Location Tracking by Kalman Filter in Wireless Sensor Networks». In: *2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*. 2009, pp. 123–128. DOI: 10.1109/WiMob.2009.30 (cit. on p. 7).

[3] Amjed Al-Fahoum and Momtaz Abadir. «Design of a Modified Madgwick Filter for Quaternion-Based Orientation Estimation Using AHRS». In: 10 (Oct. 2018), pp. 174–186. DOI: 10.17706/IJCEE.2018.10.3.174-186 (cit. on pp. 8–10).

[4] Sebastian O. H. Madgwick, Andrew J. L. Harrison, and Ravi Vaidyanathan. «Estimation of IMU and MARG orientation using a gradient descent algorithm». In: *2011 IEEE International Conference on Rehabilitation Robotics*. 2011, pp. 1–7. DOI: 10.1109/ICORR.2011.5975346 (cit. on p. 8).

[5] Jin Wu, Zebo Zhou, Hassen Fourati, R. Li, and Ming Liu. «Generalized Linear Quaternion Complementary Filter for Attitude Estimation from Multi-Sensor Observations: An Optimization Approach». In: *IEEE Transactions on Automation Science and Engineering* 16 (July 2019), pp. 1330–1343. DOI: 10.1109/TASE.2018.2888908 (cit. on p. 8).

[6] Arif Tanju Erdem and Ali Özer Ercan. «Fusing Inertial Sensor Data in an Extended Kalman Filter for 3D Camera Tracking». In: *IEEE Transactions on Image Processing* 24.2 (2015), pp. 538–548. DOI: 10.1109/TIP.2014.2380176 (cit. on p. 8).

[7] Janis Tiemann, Andrew Ramsey, and Christian Wietfeld. «Enhanced UAV Indoor Navigation through SLAM-Augmented UWB Localization». In: *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. 2018, pp. 1–6. DOI: `10.1109/ICCW.2018.8403539` (cit. on p. 9).

[8] Mushfiqul Alam and Jan Rohac. «Adaptive Data Filtering of Inertial Sensors with Variable Bandwidth». In: *Sensors* 15.2 (2015), pp. 3282–3298. ISSN: 1424-8220. DOI: `10.3390/s150203282`. URL: `https://www.mdpi.com/1424-8220/15/2/3282` (cit. on p. 9).

[9] Yuanxin Wu and Shitu Luo. «On Misalignment Between Magnetometer and Inertial Sensors». In: *IEEE Sensors Journal* 16 (Aug. 2016), pp. 6288–97. DOI: `10.1109/JSEN.2016.2582751` (cit. on p. 10).

[10] Konstantinos Papafotis and Paul P. Sotiriadis. «Accelerometer and Magnetometer Joint Calibration and Axes Alignment». In: *Technologies* 8.1 (2020). ISSN: 2227-7080. DOI: `10.3390/technologies8010011`. URL: `https://www.mdpi.com/2227-7080/8/1/11` (cit. on p. 10).

[11] D. A. Turner, I.J. Anderson, J.C. Mason, and M. G. Cox. *An Algorithm for Fitting an Ellipsoid to Data.* 1999 (cit. on p. 10).

[12] Beiya Yang, Erfu Yang, and Leijian Yu. «Vision and UWB-Based Anchor Self-Localisation System for UAV in GPS-Denied Environment». In: *Journal of Physics: Conference Series* 1922.1 (May 2021), p. 012001. DOI: `10.1088/1742-6596/1922/1/012001`. URL: `https://doi.org/10.1088/1742-6596/1922/1/012001` (cit. on p. 11).

[13] Bekir Bostanci, Sercan Tekkok, Emre Soyunmez, Pinar Oguz-Ekim, and Faezeh Yeganli. «The LiDAR and UWB based Source Localization and Initialization Algorithms for Autonomous Robotic Systems». In: *2019 11th International Conference on Electrical and Electronics Engineering (ELECO)*. 2019, pp. 900–904. DOI: `10.23919/ELECO47770.2019.8990648` (cit. on p. 11).

[14] A. Salib, A. Moussa, M. Moussa, and N. El-Sheimy. «Visual Heading Estimation for UAVs in Indoor Environments». In: *2020 International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*. 2021, pp. 1–5. DOI: `10.1109/ICCSPA49915.2021.9385709` (cit. on p. 11).

[15] YuriMat. «Easy Hard and Soft Iron Magnetometer Calibration». In: (). website: https://www.instructables.com/Easy-hard-and-soft-iron-magnetometer-calibration/ (cit. on p. 34).