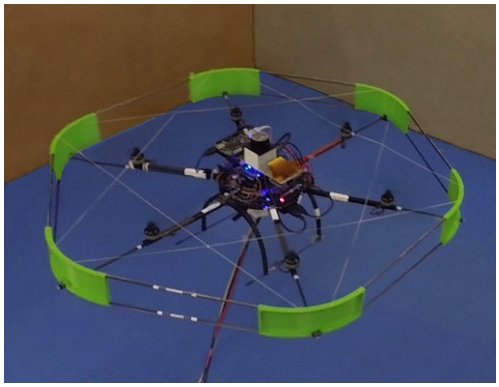
TÉCNICO LISBOA



Navigation for an Autonomous Helicopter using Onboard Sensors

José Pedro Ferreira Novo de Matos Esteves

Thesis to obtain the Master of Science Degree in

Electrical and Computers Engineering

Supervisors: Prof. Pedro Manuel Urbano de Almeida Lima
Prof. Nils Axel Andersen

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Nils Axel Andersen
Member of the Committee: Prof. Alexandra Bento Moutinho

November 2015

Dedicated to my mother and father.

Acknowledgments

The experimental work presented in this thesis was carried out in the Technical University of Denmark (DTU), at the Department of Electrical Engineering from February to July, 2015. I would like to thank Professor Nils Axel Andersen for his supervision and help during this period. I would also like to thank Professor Jens Christian Andersen for his help on this project.

The remainder of this project time was spent in Instituto Superior Técnico (IST), Universidade de Lisboa, at the Institute of Intelligent Robots and Systems group (IRSg), with the supervision of Professor Pedro Lima, whom I would like to thank for his support and his contributions to the report.

I would like to thank Carolina Loureiro for all her patience and support. She was a strong source of support throughout this project, which made me able carry on and surpass several obstacles. I would also like to thank Raul Ribeiro for refreshing my state of mind at the decisive final stage of experimental work. My sister has always supported me and has always been able to convince me that I am able to overcome life's difficulties; thank you Raquel. Finally, I would like to thank my mother for her support and for enabling me to take opportunities not only regarding this thesis, but also throughout my education and life.

Resumo

A localização e navegação de *Unmanned Aerial Vehicles* (UAVs) em espaços interiores é um tópico de investigação de interesse crescente, em que abordagens auto-contidas ainda apresentam limitações, necessitando ter em consideração o ambiente em redor, as capacidades de processamento e energia, e o peso.

Nesta dissertação, é apresentada uma solução para que um UAV execute autonomamente um caminho em linha recta num ambiente fechado, a qual é implementada num UAV de seis rotores. Esta implementação fornece a um utilizador inexperiente a capacidade de definir o caminho do hexacóptero de forma segura, sem qualquer sensor ou processamento externos. A nossa abordagem utiliza uma Inertial Measurement Unit (IMU), um sensor ultrassónico, um barómetro e um laser range finder (LRF) para calcular estimativas robustas de posição e atitude. Foram desenvolvidos modelos de Simulink para ajudar no ajuste de parâmetros dos controladores reais.

Um *Extended Kalman Filter* (EKF) tolerante a falhas é construído para estimar posição e velocidade verticais do hexacóptero; é robusto a falhas do barómetro ou sonar e consegue detectar mudanças de nível do solo. São usados dois EKFs em cascata para estimar posição e velocidade em xy; um EKF foca-se na estimativa de posição do robô usando informação do LRF, o outro EKF usa esta informação para corrigir o estado do sistema propagado usando a IMU.

Apesar desta abordagem não garantir convergência, mostrou-se sempre eficaz. O sistema final é controlado remotamente enviando referências de posição e orientação; outros métodos de controlo estão disponíveis. O hexacóptero é capaz de executar um caminho em linha recta, pairar no ar autonomamente por um período indefinido de tempo num espaço delimitado e mantém estabilidade quando são aplicadas perturbações de força e torque.

Palavras-chave: Unmanned Aerial Vehicle, localização, Extended Kalman Filter, sensores e computação a bordo, ambiente fechado, laser range finder

Abstract

The localization and navigation of Unmanned Aerial Vehicles (UAVs) in indoor environments is a research topic of interest, given self-contained approaches still present limitations that require to take into consideration the surroundings, computational capabilities and energy and weight capacities.

This dissertation introduces a solution for a UAV to autonomously execute a straight path in a known indoor environment, which was implemented in a six rotor UAV. This implementation provides to an inexperienced user the ability to safely define the path of the hexacopter, without any sensor or computation external to the UAV. Our approach relies on an Inertial Measurement Unit (IMU), an ultrasonic sensor, a barometer and a laser range finder (LRF) to compute robust position and attitude estimates. Simulink control models were developed to guide the parameter tuning of the real controllers.

A fault-tolerant Extended Kalman Filter (EKF) is built to estimate the vertical position and velocity of the hexacopter; the filter is robust to barometer or sonar outages and can detect ground-level changes. A cascaded EKF is used to estimate xy position and velocity, where both the EKFs perform prediction and update steps, but one focuses on computing a 2-D pose estimate using LRF measurements, whereas the other uses these pose estimates to correct the state propagated using the IMU. Although this approach does not guarantee convergence, it proved to be effective in practice. The final system is controlled remotely by sending position and yaw setpoints; other control methods are available. The hexacopter is able to execute a straight path, hover autonomously for an indefinite amount of time in a bounded space and is stable against force or torque disturbances.

Keywords: Unmanned Aerial Vehicle, localization, Extended Kalman Filter, on-board sensing and computation, indoor environment, laser range finder

Contents

List of Tables

List of Figures

Nomenclature

BF	Body Frame
DoF	Degree of Freedom
EF	Earth Frame
EKF	Extended Kalman Filter
IMU	Inertial Measurement Unit
KF	Kalman Filter
MAV	Micro Aerial Vehicle
NED	North East Down
OF	Odometry Frame
PF	Pixhawk Frame
RANSAC	RANdom SAmple Consensus
RC	Remote Control
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
VSLAM	Visual Simultaneous Localization and Mapping

Greek symbols

$\boldsymbol{\tau}$	Vector containing torques
$\ddot{\phi}$	Roll angular acceleration
$\ddot{\psi}$	Yaw angular acceleration
$\ddot{\theta}$	Pitch angular acceleration
$\dot{\phi}$	Roll angular velocity

$\dot{\psi}$ Yaw angular velocity

$\dot{\theta}$ Pitch angular velocity

Ω_i Angular velocity of the ith rotor

Ω_r Summed angular velocity of the rotors

ϕ Roll angle

ψ Yaw angle

ρ Air Density

θ Pitch angle

Roman symbols

$\boldsymbol{\mathcal{I}}$ Inertia Tensor

$\boldsymbol{F}$ Vector containing linear forces

$\boldsymbol{I}$ Identity matrix

A Rotor disc area

A_i Corss-sectional area of the rotor blade along the ith direction

g Gravitational acceleration

J_r Inertia associated to propeller and rotating part of the motor

l Length of multirotor's arm

Q_i Torque of the ith rotor

R_prop Radius of rotor, equivalent to half the length of the propeller

T_i Thrust force of the ith rotor

C Air friction constant

C_Q Torque constant

C_T Thrust constant

ω Angular Velocity

u, v, w Velocity Cartesian components

Subscripts

b Barometer sensor index

s Sonar sensor index

x, y, z Cartesian components.

Chapter 1

Introduction

1.1 Motivation

The popularity of Unmanned Aerial Vehicles (UAVs) has grown in the recent years, both as an academia research topic and as a commercial product, with a big acceptance and demand of the general public. The patent publication on drones has been growing exponentially [1]. The market is dominated by the military, surveillance and aerial photography, furthermore new purposes for UAVs are created along as the technology advances. The fast growth of this area is enabling a technology that is considered dangerous by many to become present in indoor environments, in close presence with humans. Multirotors are suited as aerial vehicles for indoor environments because they can perform vertical take-off and landing, and hover in a small bounded space. UAVs are preferred to Unmanned Ground Vehicles (UGVs) because of the increased mobility in the air. However, controlling the motion of these aerial vehicles is more challenging.

UAVs are becoming more common as a solution for inspection of closed places dangerous for a human being to enter, such as: contaminated houses, factories or power plants, buildings whose structure is at risk of falling down, mining sites with the risk of explosion or rupture. They are also a candidate for monitoring buildings, delivering packages inside hospitals, factories and workplaces, and, ultimately, for the enjoyment and/or aid inside the house of the everyday-user.

The motivation for this project comes from the need for the deployment of these autonomous vehicles in indoor structured environments, where common localization solutions are not available and it is not plausible to place external sensors or artificial markers on the environment, since these are not scalable options.

The most common localization solution today is Global Positioning System (GPS), but it may not be available in indoor environments; moreover, GPS accuracy is not suitable for the localization of a UAV in a small closed space and it does not take into consideration the environment the robot is inserted in. There are several alternatives to compute the localization of robots:

- Wireless Real-Time Location System (RTLS), where the different types can be Nearest Access Point, Time Difference on Arrival (TDoA), Angle of Arrival (AoA) and Received Signal Strength

Indicator (RSSI);

- Computer Vision using external fixed camera(s) [2];
- Motion Capture System with Passive or Active markers [3], [4].

Some of the approaches mentioned above are able to give position and velocity estimates accurate and fast enough to control a UAV in an indoor setting. However, these do not represent a scenario that can be expanded in large scale. Approaches relying on on-board computer vision algorithms are widely used, but they are either too computationally expensive to give real-time estimates [5] or simpler methods may not provide absolute pose estimation with bounded error, giving only relative motion estimates [6]. Advances are being made to perform computer vision tasks that can estimate absolute localization, with limited computational power [7], [8], [9].

In this work, we rely on a laser range finder, which is a growing technology that only became lightweight enough to be carried by a UAV in the recent years, for the localization in the xy-space. The reasons behind this choice are: accuracy ($\leq 1\%$ of measurement) in the measurements of the environment, low quantity of data while containing a lot of meaningful information, high-enough sampling rate (10Hz), high angular resolution (0.36^o) and range large enough for the environments considered (4m). Having a good ratio of information per size of data enables to use fast algorithms and reduces the need to pre-process information, making it possible to do all the computations on-board. A single-board computer on the lowest price range, the BeagleBone Black, is used with only the original 4GB storage capacity. This demonstrates the computational advantage in using laser sensors comparing to cameras, which have higher computational requirements to reach similar goals. An ultrassonic sensor is used to measure altitude above ground. Despite lower accuracy than a 1-D LRF, it is considerably cheaper and outperforms the LRF when the hexacopter is slightly tilted, due to the beam shape of the sensor. In the latter case, the sonar is able to give the distance to the point on the ground closer to the robot (resulting in measuring the true height), while the laser will give a measurement that is relative to a point it is directly pointing at, which is further away from the robot (resulting in a measurement bigger than the true height). Furthermore, an IMU is essential for providing high frequency estimates that allow to observe fast dynamics.

The objective of the work is to design, implement and test a system that can perform autonomous path execution in indoor structured environments using an environment map. The path is defined by the operator. For this, self-localization and control strategies are fulfilled, using solid approaches, in order to get a robust final system that can meet the initial goals of this project.

This project was also motivated by the ambition of expanding *Danmarks Tekniske Universitet's* (DTU's) software, Mobotware, to interact and control an aerial vehicle. Mobotware should become compatible with the recent technology, the Pixhawk [10], [11], which runs the PX4 autopilot. The PX4 lacks the software to control autonomously a UAV indoors, since most of the efforts have been made to work with sensors adequate for outdoor usage, such as GPS and barometer. The two software systems, Mobotware and PX4, have never functioned together before. The new software developed in this project will augment the capabilities of both software systems, in order to create a safe way for a UAV to execute a path defined by any person, regardless of his/her knowledge on aerial vehicles.

Absolute localization cannot be accomplished using the original PX4 approach, hence other algorithms are developed in this project. The results of the localization algorithms originally available on the PX4 are compared against the algorithms developed in this thesis.

The platform used is a hexacopter, but the implementation is independent from the type of multirotor. The physical system, hexacopter, and the sensors used are studied. From these models, simulators are created to study the algorithms that will be implemented. Each implementation that is developed, is tested in simulation before having a version ready for trial on the real system.

The algorithm described by Siegwart and Nourbakhsh [12] is used for self-localization, which is similar to the one described by Cox et al. [13] and Jensfelt [14]. This algorithm is feature-based, where line segments are extracted from sensor data using RANSAC.

Two Extended Kalman Filters (EKFs) are created to estimate position and velocity of the UAV, and a third one already available in Mobotware is modified to work correctly with an aerial vehicle. The Kalman Filters rely on IMU measurements to perform state propagation and information from low sampling frequency sensors - laser range finder, sonar and barometer - to estimate the error of the state and correct it. The algorithm that uses laser information to compute a position estimate requires the user to provide a 2-D metric map that describes the environment as straight lines. Outlier measurements of the environment are rejected if they are not lines or if there is no match to a specific line feature on the map in the range of sight. It is used a cascaded approach to combine two EKFs that are responsible for estimating the state in the x and y dimensions. These EKFs are designed so that the algorithm that uses the laser information can be easily changed; laser outages are tolerated and other sensors or estimators can be added in the same way, parallel to the secondary KF.

The final system has a control loop that fuses information computed on the Pixhawk and on the on-board computer, BeagleBone. The PX4 autopilot and Mobotware both contribute to the control loop of the hexacopter, making possible the autonomous navigation of the hexacopter in a known indoor environment. The operator can control the UAV using two different manners:

1. controlling the velocities in x, y, z and yaw with a Remote Control;
2. giving a setpoint of x, y, z and/or yaw from a computer connected remotely to the BeagleBone.

The hexacopter can be left hovering for an indefinite amount of time, without leaving a small confined circular area of 60cm radius, which is small for a 44cm radius hexacopter. It also withstands perturbations, such as a person pushing and rotating it, always returning to the setpoint chosen by the user.

1.2 Previous work

This project uses previously available software. DTU's control software, Mobotware, serves as a platform to interact with the laser scanner sensor. The laser-based localization algorithm [12] is already implemented for UGVs. The Pixhawk firmware is ready-to-fly, but not in indoor environments. With PX4 plug-in sensors, a flight indoors requires continuous supervision by the operator. Nonetheless, the original implementation of the controllers and low-level commands are used as a starting point. A Simulink model of a quadrotor [15] is used as a starting point for the controller's study.

1.3 Objectives

This project focuses on developing a solution on a UAV which enables it to execute autonomously a path defined by any person, in an indoor GPS-denied environment. The approach is implemented in a six rotor UAV, which becomes able to hover and move between setpoints autonomously. This work is in the domain of a localization-only strategy, in the 6DoF space. Localization is the process of finding the position and orientation of the robot, i.e., its pose. To achieve this, we assume to know a simple 2-D metric model of the environment. Estimating the pose in real-time with a good accuracy is essential to achieve an autonomous robot, especially for a flying robot in an indoor environment.

For this project, the final system should fulfill the goals:

- Use only on-board sensors,
- Use only on-board computation, limited by the computational power of a cheap on-board computer,
- Ability to perform 6DoF self-localization in a (partially) known structured environment without needing to modify it,
- Move autonomously from one point to another,
- Stay autonomously at the same pose for an indefinite amount of time, inside a confined circle of radius 35% bigger than the UAV radius,
- Withstand disturbances, such as a person pushing it.

The following tasks are set as objectives to achieve during the project:

- Assemble all additional hardware needed for this project,
- Create communication connections between Pixhawk and BeagleBone, BeagleBone and a ground-based computer,
- Create DTU software to communicate with the Pixhawk,
- Develop and implement estimation algorithms with errors smaller than 2.5% (altitude) and 5% (x and y position),
- develop estimation algorithms that have the possibility of including additional sensors and/or receiving information from more estimators,
- Develop the software to give commands to the UAV from a ground-based computer,
- Reinforce robustness to failures of autonomous flight.

1.4 Thesis Outline

The remainder of this report is divided in 4 parts: Chapter 2 introduces the notation, concepts, and algorithms necessary to understand the thesis work; Chapter 3 explains the methodology followed to develop and implement the algorithms previously described, as well as the work done to set up a fully functional system; in Chapter 4, the results of the implemented approaches are presented and discussed against the original implementation from which the initial system relied on; Chapter 5 presents the conclusions of this thesis and lays this project as a foundation that can be expanded, suggesting future work and improvements that can be made.

Chapter 2

Background

This chapter presents the concepts and notation relevant to follow up the remainder of the dissertation. The current state of research in the topics related to localization in UAVs is presented in Section 2.1, which will introduce the reader to the solutions available today to perform localization on UAVs. Some notation and basic understanding of the physical system is presented in Section 2.2, followed by a more concrete description in Section 2.3, using a mathematical formulation of forces and torques that affect the hexacopter in the 6DoF. The hardware and software used are described in Sections 2.4 and 2.5, respectively. The reader is then introduced to the localization algorithms used. Section 2.6 introduces the Extended Kalman Filter, which is used in a sensor-based approach to estimate variables in the x, y and z dimensions. Section 2.7 describes a feature-based approach to perform pose self-localization in a 2-D space, which is used in this project.

2.1 Related Work

Navigation-related problems in mobile robots can be categorized as follows [16]:

- Path planning: this topic is addressed in a completely know environment, with perfect sensing capabilities, where the problem is stated as the minimization of the cost of going from a starting point to ending point, while avoiding obstacles.
- Exploration of an unknown environment, relying on accurate range sensing and odometry information. Simultaneous Localization and Mapping (SLAM) is of great interest to solve this topic, and research is currently being done to allow for its computation to be processed solely on-board of UAVs.
- Path execution within known environments, where one of the key problems is robot self-localization. Sensors generally used to reach this goal are camera, laser, sonar and infrared. The primary issue of path execution within known environments is the matching of sensed data against map information.

In this project, the last topic is studied, focusing on the ”where am I” problem. This issue cannot be solved relying entirely on odometry, inertial information or motion measurement because it leads to unbounded position error [17]. In Unmanned Aerial Vehicles (UAVs), this error grows faster than in

Unmanned Ground Vehicles (UGVs), since it grows not only when the robot is instructed to move, but also when the robot is intended to stay put, given that it is not motionless when hovering.

Vision from on-board cameras can be fused with IMU information to estimate attitude and/or position [18], [19], [20], [21], [22], [8]. In 1992, Kosaka and Kak had already used vision to provide self-localization, relying on visible landmarks [23]. When such is not possible, the self-localization problem can be simplified by modifying the environment, adding visual markers. Artificial marker based localization can be performed in several ways, such as: using a Wii remote as a low-cost solution to estimating position by tracking an infrared blob [24]; using visual markers to track position using the Pixhawk [11]; using visual markers to estimate attitude and position information, using monocular vision (in this case, computation is done off-board) [4]; etc.

Apart from being used on-board, cameras can also be used as external sensors to estimate position and attitude [2]. To enable the tracking of extremely difficult maneuvers, such as triple flips [3], expensive motion capture systems are a commonly adopted solution. This method is very efficient to test algorithms that need accurate measurements and to serve as a ground truth reference to evaluate estimation approaches. Lorenz Meier et al. use a motion capture system with very precise estimates (error $<$1mm) at 250Hz, to report the results of the Pixhawk system [11].

Information from several sensors can be combined to achieve exploratory goals. Achtelik et al. present their achievements towards a solution that combines an on-board camera and a laser sensor to autonomously navigate in unstructured and unknown indoor environments [5]. The authors express that despite advances within the community, in 2010, SLAM algorithms were still too computationally demanding, even for powerful desktop computers, and were therefore not implementable on today's small embedded computer systems that can be mounted on-board indoor UAVs. They apply a novel algorithm that uses laser information to generate a local cost-map, from which the optimal rigid body transform that maximizes a given reward function can be found. In 2010, Blösch et al. have created the first solution where a micro aerial vehicle is able to navigate through an unknown and unstructured environment, using a single camera as exteroceptive sensor to perform Visual SLAM (VSLAM) [25]. To do so, they resort to ground computer to run the SLAM algorithm. Grzonka et al. perform SLAM using a laser sensor, but also resort to off-board computation [26].

Recently, it has become possible to do SLAM using only on-board computation. Shaojie Shen et al. present a self-contained approach to perform SLAM using a laser sensor and a camera [27]. A self-contained solution is implemented by Brockers et al., where a novel inertial-optical flow (IFO) [28] is used to estimate full attitude and almost drift-free metric distance, and a camera is used to perform VSLAM [9]. This author had previously used a camera, sonar and IMU for self-contained navigation, performing SLAM [29]. A self-contained approach to perform SLAM using only a laser range sensor, an IMU and a sonar altitude sensor is presented by Chowdhary et al. [30].

The problem of position estimation in a known (or partially known) two-dimensional polygonal environment has been solved for some time. Cox et al. match noisy laser range scans against an available metric map of the environment consisting of polygonal obstacles, in order to keep the position error as small as possible [13]. The method consists of an iterative least-squares algorithm that finds the

congruence between a range scan and the map, provided that the initial displacement is small. This method is implemented in a self-contained UGV [31]. Jensfelt discusses three laser-based localization approaches [14]: EKF feature-based localization; Multiple Hypothesis Localization, which assumes no previous knowledge of the pose, but a known environment map; Monte Carlo Localization, originally introduced as Condensation [32]. Gutmann and Schlegel also compared several approaches for self-localization using a 2-D laser range finder (LRF) [33]. The most successful techniques for state estimation are Bayesian filters, such as particle filters or extended and unscented Kalman Filters [34]. Several variations of these filters have been studied and compared [35].

To perform a feature-based localization, it is necessary to extract the features from sensor data. Nguyen et al. compare several line extraction algorithms [36], and the fastest method described, Split-and-Merge, has been modified by by Borges et al., creating the Split-and-Merge Fuzzy (SMF) algorithm [37]. In a feature-based localization algorithm, it is important not only to extract features, but also to represent them. A new algorithm for representation of structured environments with low measurement noise is developed by Harati and Siegwart [38]. Features can be separated as straight-line or, for curve segments, as triangles [39].

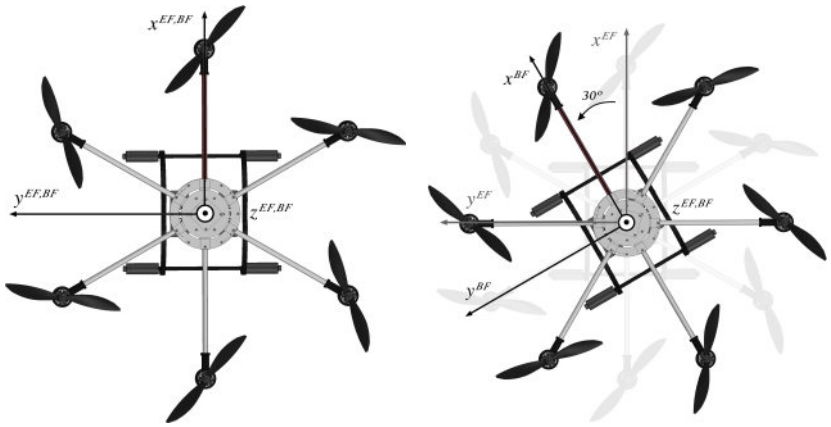
When using a laser sensor, an alternative to a feature-based approach is an iconic approach, where raw sensor data is used. Iterative Closest Point (ICP) is widely used, and several variants of this algorithm have been implemented and compared [40]. Previous laser measurements can be aligned using the ICP algorithm, which estimates a transformation between two point clouds, describing the displacement between them.

2.2 Basic Principles

2.2.1 Coordinate Frames

When describing a multirotor, it is important to establish coordinate frames. Let us consider the Earth Frame (EF) as being the fixed frame of reference shown in Figure 2.1(a), which has its origin coinciding with the origin of a map of the environment. This 2-D map is represented in the EF and needs to be provided by the user, in order to allow the hexacopter to localize its pose in the environment described by the map. In this thesis, the localization and navigation are in with respect to (w.r.t.) this frame. The Body Frame (BF) is defined as always being aligned with the helicopter as seen on Figure 2.1(b). By using this frame, matters related to the sensor data and attitude are simplified. One can observe in Figure 2.1(b) that when the hexacopter rotates 30 degrees around the z axis, the EF remains the same as in Figure 2.1(a), while the BF rotates with the hexacopter. This is true not only around the z axis, but also around the x and y axes. The EF remains the same when the UAV is translated, while the BF suffers the same translation as the UAV.

Since this project relies on already existing software, other frames of reference are used to ensure compatibility between different applications. These frames will be mentioned throughout this report, such as the Pixhawk Frame (PF) and the Odometry Frame (OF). The Pixhawk Frame is a North East



(a) (b)

Figure 2.1: Figure (a) shows the frames of reference EF and BF and figure (b) shows the same frames after a positive rotation of 30 degrees along the x axis.

Down (NED) frame, which means that the z axis is pointing downwards, in the direction of the Earth. The PF has its origin on the position x, y and z where the helicopter is firstly initialized, while its orientation is defined in relation to the map, as in the EF. The roll and pitch axes are the same as in the BF. The Odometry Frame was created as a bridge between the PF and the EF, in order to feed the hexacopter's pose correctly to the localization algorithm, explained in Section 2.7. The OF is the rotation of the PF in such a way that the x, y and z axes' directions match the EF, and has its origin in the same point as PF. Frames EF, PF and OF can be intuitively related in following way: EF is the frame that represents the robot with respect to the world in a fixed map; OF represents the robot in the same frame as EF, but translated by the initial position; PF represents the robot translated in the same way as OF and rotated to be in the NED convention. Figure 2.2 shows the each of these Frames of Reference in relation to each other. The transformations from one frame to the other are EF - translation - OF - rotation - PF and are declared explicitly in Transformation Matrix notation in (2.3), (2.4) and (2.5).

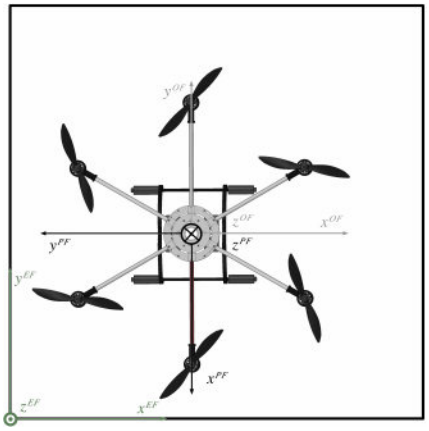


Figure 2.2: Relationship between reference frames EF, OF and PF. EF has the same origin as the Map provided by the user, while PF and OF have their origin in the position at which the hexacopter is initialized. The outer rectangle represents the Map, with origin in the lower left corner. It should be noted that the OF has the z axis pointing upwards, while it points downwards on the PF.

The attitude is defined as the rotation from the EF to the BF. It represents the rotations about the x, y and z axes (on the BF) that follow the right-hand rule, which consist of roll, pitch and yaw, respectively.

The attitude in the PF[1] and OF are defined in the same way as for the EF.

The attitude of the hexacopter is controllable by managing the differences in thrust between the different actuators. The fourth DoF, thrust, is defined as the force responsible for a translational acceleration along the z axis in the BF and is controlled by the sum of the thrusts of each actuator.

It is important to know the transformation from one reference frame to another, in order to make compatible the algorithms that use different reference frames. The Rotation Matrix that transforms a vector in the EF to one in the BF is

$$\overset{BF}{EF}\boldsymbol{R} = \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}, \tag{2.1}$$

where s_x and c_x are $\sin x$ and $\cos x$, respectively. This matrix is constructed by using the rotation matrix around each axis, shown in (2.2), by computing $\overset{BF}{EF}R = R(\psi)R(\theta)R(\phi)$. The inverse matrix corresponds to the transformation from the BF to the EF and is defined as $\overset{EF}{BF}R = (\overset{BF}{EF}R)^T$.

$$\boldsymbol{R}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \quad \boldsymbol{R}(\theta) = \begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix} \quad \boldsymbol{R}(\psi) = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

The Transformation Matrices that transforms a vector in the EF to one in the PF and from the PF to the EF are shown in (2.3).

$$\overset{PF}{EF}\boldsymbol{T} = \begin{bmatrix} 0 & -1 & 0 & y_0 \\ -1 & 0 & 0 & x_0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \overset{EF}{PF}\boldsymbol{T} = \begin{bmatrix} 0 & -1 & 0 & x_0 \\ -1 & 0 & 0 & y_0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

The Transformation Matrices that transforms a vector in the EF to one in the OF and from the OF to the EF are shown in (2.4).

$$\overset{OF}{EF}\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & y_0 \\ 0 & 1 & 0 & x_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \overset{EF}{OF}\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

The Transformation Matrices that transforms a vector in the OF to one in the PF and from the PF to the OF are shown in 2.5, which is equivalent to using only the equivalent rotation matrices, by deleting

[1] Since the PF is in the NED form, the attitude cannot be computed directly as the rotation to the BF. In this case, the attitude is computed using an analogous Body Frame, where this one is rotated in order to be in NED form. The resulting attitude is the same as in the EF or the OF.

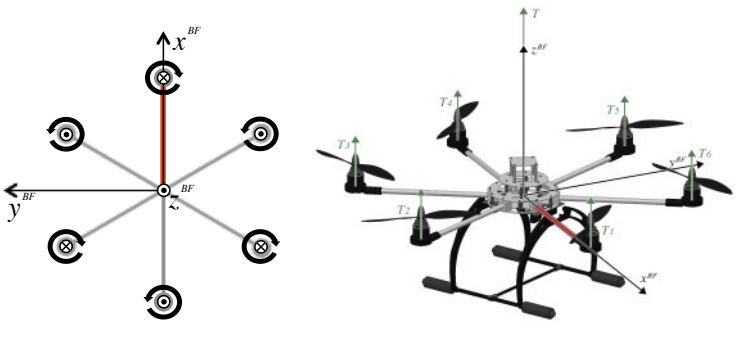
the last row and column.

$$ {}^{PF}_{OF}\boldsymbol{T} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{OF}_{PF}\boldsymbol{T} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5} $$

2.2.2 Understanding the Hexacopter

A rigid body in space has 6 Degrees of Freedom (DoF): position in x, y and z, plus rotation about x, y and z. Our system is a Unmanned Aerial Vehicle (UAV) that has 6 actuators, hence the name hexacopter. Even though the hexacopter has 6 actuators, it is underactuaded, *i.e.*, it can move in the mentioned 6 DoF, but can only control 4 DoF: thrust, pitch, yaw, and roll. Since the hexacopter is underactuated, this results in it being a non-holonomic vehicle that has to plan its maneuvers in the controllable 4 DoF space. Each actuator, also called rotor, consists of a set of motor and propeller. Since it is only possible to control 4DoF directly and there are a total of 6 actuators, there is redundancy in our model. This redundancy can be used as an advantage to increase stability, improve resistance to disturbances and deal with actuator failure. Ducard and Hua introduces an advantage of over-actuated systems as being the possibility of separating the derivation of the control laws and the design of a control allocator citeducard2011discussion, *i.e.*, it is possible to use certain actuators to deal with the control of the system, while allocating others to deal with specific objectives.

The rotors spin around their axis as shown in Figure 2.3(a), where each arrow represents the spinning direction and the corresponding rotation axis has its direction according to the right-hand rule. The motors 1, 3 and 5 spin clockwise (CW) and motors 2, 4 and 6 spin counterclockwise (CCW), in order that if all motors spin at the same angular speed, the sum of all the motor torques will be zero. Each motor spins a propeller, which generates a thrust vector as displayed in Figure 2.3(b). In order to have all the rotor with thrusts pointing upwards, CCW motors have corresponding CCW propellers that will push the air downwards, which is the same direction that the air is pushed using the remainder CW propellers. The speed of the motor rotation is related to the thrust generated by the rotor, and the thrust of each rotor will have a role in the motion of the helicopter. This will be explained in more detail in Section 2.3, but the intuition behind it will be explained here.



(a) (b)

Figure 2.3: Figure (a) shows the rotation direction of each motor. Figure (b) shows the Thrust vector associated to each rotor and the sum of them, which is the total Thrust vector.

Ideally, when all the motors are spinning at the same speed, then the total sum of all torques is zero and the helicopter will move along the z axis in the BF. If the sum of the thrusts is equal to the gravity force ($F_g = mg$) and the z axis in the BF is aligned with the z axis in the EF , *i.e.*, roll and pitch angles are zero, then the multirotor is in an equilibrium state called hover. By increasing or decreasing the total thrust, the vehicle will move upwards or downwards, respectively, along the z axis in the BF.

As explained, Figure 2.3(b) shows the thrust vectors associated to each actuator. By increasing/decreasing thrusts T_2 and T_3 over T_5 and T_6, the vehicle will tilt positively/negatively around the x axis on the BF, generating a positive/negative roll angular velocity and changing the roll angle positively/negatively. The analogous happens for the pitch when increasing/decreasing T_3, T_4 and T_5 over T_1, T_2 and T_6, which will generate a positive/negative angular velocity around the y axis on the BF, thus changing the pitch angle accordingly.

A rotation around the z axis on the BF will change the yaw angle. Each motor contributes with a torque that will contribute to spin the multirotor on the opposite way that the motor spins, *i.e.*, a CW motor will contribute with a torque that causes the hexacopter to spin CCW, and vice-versa. If the set of CW motors 2, 4 and 6 spins faster than the set of CCW motors 1, 3 and 5, then the hexacopter will rotate with a positive yaw angular velocity, *i.e.*, it will spin CCW if seen from above.

A detailed mathematical description of how the forces and torques are translated into acceleration is presented in Section 2.3, along with the contributions of each rotor to forces and torques on the system.

2.3 Model of the Hexacopter

2.3.1 Inertia Model

The inertia model is constructed based on a simplified version of the real hexacopter, which is presented in Figure 2.4. In this simplified model, the body is symmetric with respect to any plane that is orthogonal to one of the x, y or z axes and that plane contains the center point, which represents the center of mass. This simplification makes the Inertia Tensor a diagonal matrix:

$$\boldsymbol{\mathcal{I}} = \begin{bmatrix} \mathcal{I}_{xx} & 0 & 0 \\ 0 & \mathcal{I}_{yy} & 0 \\ 0 & 0 & \mathcal{I}_{zz} \end{bmatrix}. \tag{2.6}$$

It should be noticed that in order to get this simplified model, one has to admit that the protections that surround the hexacopter are also centered in the point of mass of the system, when in fact they are 4.5cm displaced upwards along the z axis.

The central part of the hexacopter comprises most of the electronics hardware, such as the motor controllers, Pixhawk, BeagleBone, laser rangefinder, dc voltage regulators, signal decodifier, ultrasonic distance sensor and others. It is modeled as a solid sphere:

$$I_{body} = \frac{2}{5} m_{body} r^2 \boldsymbol{I}_{3\times 3}, \tag{2.7}$$

where m_{body} is the mass of the set of hardware mentioned, r is the radius of the sphere and $\boldsymbol{I}_{3\times 3}$ is an

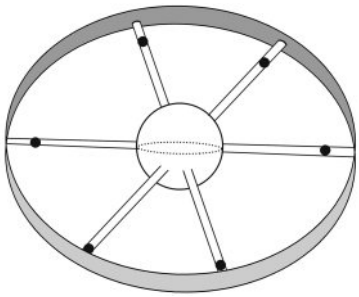


Figure 2.4: Simplified physical model of the hexacopter, with which the Inertia model is constructed.

Identity matrix. The arms of the multirotor are modeled as rods and they also include the solid carbon fiber rods that hold the vehicle protections:

$$I_{arms} = \begin{bmatrix} \frac{1}{3}m_{arm}r^2(1+2\sin{\theta_1}^2) & 0 & 0 \\ 0 & \frac{1}{3}m_{arm}r^2(\frac{1}{3}+2\sin{\theta_2}^2) & 0 \\ 0 & 0 & 2m_{arm}r^2 \end{bmatrix}, \tag{2.8}$$

where m_{arm} is the mass of one arm, r is the length of one arm, θ_1=60^o and θ_2=30^o. The protections are modeled as a hollow cylinder with 1cm thickness:

$$I_{prot} = \begin{bmatrix} \frac{1}{12}m_{prot}(3(r_{in}^2+r_{out}^2)+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m_{prot}(3(r_{in}^2+r_{out}^2)+h^2) & 0 \\ 0 & 0 & \frac{1}{2}m_{prot}(r_{in}^2+r_{out}^2) \end{bmatrix}, \tag{2.9}$$

where m_{prot} is the mass of the protections, r_{in} and r_{out} are the inner and outer radius of the protections, respectively, and h is the height of the protections. The motors are simplified as being point masses:

$$I_{motors} = \begin{bmatrix} 4m_{motor}d_1^2 & 0 & 0 \\ 0 & 4m_{motor}d_2^2 + m_{motor}d_{motor}^2 & 0 \\ 0 & 0 & 6m_{motor}d_1^2) \end{bmatrix}, \tag{2.10}$$

where m_{motor} is the mass of one motor, d_1 is the distance of motors 2, 3, 5 and 6 to the x axis, d_2 is the distance of motors 2, 3, 5 and 6 to the y axis and d_{motor} is the distance of a motor to the center of mass of the hexacopter; where motors are numbered according to notation on Section 2.2.2.

The result of computing the inertia of each component and summing them is the following Inertia Tensor:

$$\mathcal{I} = \begin{bmatrix} 0.0449 & 0 & 0 \\ 0 & 0.0431 & 0 \\ 0 & 0 & 0.0838 \end{bmatrix}. \tag{2.11}$$

This model will be used to construct a Simulink model of the hexacopter, used in Section 3.1.1.

2.3.2 Mathematical Model Formulation

In this chapter, the hexacopter is modeled using the Newton-Euler equations. These equations provide a global characterization of the dynamics of a rigid body subject to external forces and torques. The hexacopter is modeled in the BF as a rigid body, where external forces and torques are acting on it. These

forces and torques have several origins: the weight of the body of the hexacopter, the motion of the body, the actuators, the consequent airflow originated by the actuators and environment disturbances. In translational motion, the body is represented by the center of mass. A rotation is calculated when an external torque is applied (e.g. torque generated by an actuator), and that torque acts about the center of mass. It is important to notice that the linear and angular motions are coupled since the linear velocity depends on the attitude, resulting on each being associated with force and torque components.

The Ground Effect will be neglected in this model. Ground Effect is the phenomenon of lift increase and drag reduction experienced when the vehicle flies closer to the ground. According to Leishman and Hua, it can be neglected at a distance bigger than 2 times the rotor radius [41]. In our case, this distance will be around 1.5m, taking into consideration the dynamics of the 6 rotors. This phenomenon is caused by the increase in air pressure as the aircraft is flying close to the ground, which causes an interesting decrease in overall drag with an increase in total lift from the rotor. Other boundaries, such as walls, can create an effect similar to Ground Effect by creating a non symmetrical flow of air going through the hexacopter blades. Our system will be modeled as being independent from the height to the ground and the environment surroundings, thus these effects will be neglected.

As previously stated, the Newton-Euler formalism is used to model the hexacopter. Murray et al. describe the general equations of a rigid body subject to external forces and torques, in the BF, as follows [42]:

$$\begin{bmatrix} \boldsymbol{F} \\ \boldsymbol{\tau} \end{bmatrix} = \begin{bmatrix} m\boldsymbol{I}_3 & \mathbf{0}_{3\times 3} \\ \mathbf{0}_{3\times 3} & \boldsymbol{\mathcal{I}}_H \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{v}}^{BF} \\ \dot{\boldsymbol{\omega}}^{BF} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\omega}^{BF} \times m\boldsymbol{v}^{BF} \\ \boldsymbol{\omega}^{BF} \times \boldsymbol{\mathcal{I}}_H \boldsymbol{\omega}^{BF} \end{bmatrix} \tag{2.12}$$

where I_3 is the identity matrix of size 3, $\mathcal{I}_H$ is the Inertia Tensor about the center of mass of the hexacopter and v^{BF} [m/s^2] and ω^{BF} [rad/s] are the linear and angular velocities, respectively, in the Body Frame.

By expanding the cross-products and rearranging (2.12), one obtains

$$\begin{aligned} \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} &= \begin{bmatrix} \dot{\theta}w - \dot{\psi}v + \frac{1}{m}F_x \\ \dot{\psi}u - \dot{\phi}w + \frac{1}{m}F_y \\ \dot{\phi}v - \dot{\theta}u + \frac{1}{m}F_z \end{bmatrix} \\ \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} &= \begin{bmatrix} (\dot{\theta}\dot{\psi}(\mathcal{I}_{yy} - \mathcal{I}_{zz}) + \tau_x)/\mathcal{I}_{xx} \\ (\dot{\phi}\dot{\psi}(\mathcal{I}_{zz} - \mathcal{I}_{xx}) + \tau_y)/\mathcal{I}_{yy} \\ (\dot{\phi}\dot{\theta}(\mathcal{I}_{xx} - \mathcal{I}_{yy}) + \tau_z)/\mathcal{I}_{zz} \end{bmatrix}, \end{aligned} \tag{2.13}$$

where the superscript BF is omitted. We assume $\boldsymbol{w} = \begin{bmatrix} p, q, r \end{bmatrix}^T = \begin{bmatrix} \dot{\phi}, \dot{\theta}, \dot{\psi} \end{bmatrix}^T$, which holds for small and similar roll and pitch angles.

Analysis of the forces in the model

The forces acting on the hexacopter will now be discussed. A Hub force in the x and y axes is presented as being a force resulting from all the horizontal forces acting on all the blade elements [43]. This force will be neglected in our model.

• **Gravity force**

$$^{BF}F_g = {}^{BF}_{EF}R \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} = \begin{bmatrix} -(c_\psi s_\theta c_\phi + s_\psi s_\phi)mg \\ -(s_\psi s_\theta c_\phi - c_\psi s_\phi)mg \\ -c_\theta c_\phi mg \end{bmatrix} \tag{2.14}$$

• **Thrust force**

$$^{BF}F_{thrust} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{6} C_T \rho A(\Omega_i R_{prop})^2 \end{bmatrix} \tag{2.15}$$

where ρ [kg/m^3] is the air density 1.2041kg/m^3, A [m^2] is the disc area of the propellers when rotating ($A = \pi R_{prop}^2$) and $R_{prop}[m]$ is the rotor radius, corresponding to half the length of the propeller. C_T [rad^{-2}] is the thrust constant, whose value 0.0158 rad^{-2} was taken from Brogaard [15], where a study of the same motors and propellers as the ones used has been made. In the BF, the thrust force only acts along the z axis, since it is perpendicular to the propeller blades.

• **Air friction force**

$$^{BF}F_{airfriction} = \begin{bmatrix} -\frac{1}{2}CA_x\rho u|u| \\ -\frac{1}{2}CA_y\rho v|v| \\ -\frac{1}{2}CA_z\rho w|w| \end{bmatrix}, \tag{2.16}$$

where C is the dimensionless air friction constant and A_i [m^2] is the cross-sectional area of the rotor blade.

Analysis of the torques in the model

The 4 types of moments in the dynamics of the hexacopter will now be discussed. The ith rotor has an influence of $\sin(\frac{i-1}{6}2\pi)$ on roll, $\cos(\frac{i-1}{6}2\pi)$ on pitch, (-1)^i on yaw and 1 on thrust, with $i \in [1, 6]$.

• **Torque of the rotors**

$$\tau_{torque}^{BF} = \begin{bmatrix} 0 \\ 0 \\ \sum_{i=1}^{6} (-1)^i Q_i \end{bmatrix}, \tag{2.17}$$

where Q_i [Nm] is the torque of the ith rotor. Q_i can be calculated by $Q_i = C_Q \rho A(\Omega_i R_r)^2 R_r$, according to Leishman and Hua [41], and for our case it becomes $Q_i = 2.72 \times 10^7 \Omega_i^2$, according to Brogaard [15].

• **Thrust difference**

This torque component results from the sum of thrusts on one side of a given axis being bigger than the sum of thrusts on the other side.

$$\tau_{thrust}^{BF} = \begin{bmatrix} l \sum_{i=1}^{6} C_T \rho A (\Omega_i R_{prop})^2 \sin(\frac{i-1}{6} 2\pi) \\ l \sum_{i=1}^{6} C_T \rho A (\Omega_i R_{prop})^2 \cos(\frac{i-1}{6} 2\pi) \\ 0 \end{bmatrix} = \begin{bmatrix} l \sum_{i=1}^{6} T_i \sin(\frac{i-1}{6} 2\pi) \\ l \sum_{i=1}^{6} T_i \cos(\frac{i-1}{6} 2\pi) \\ 0, \end{bmatrix} \tag{2.18}$$

where l [m] is the length of hexacopter's arm and T_i [N] is the thrust force exerted by the ith rotor.

• **Body gyroscopic effect**

This effect is the inertial effect of the angular velocity, which is already taken into accound in the Newton-Euler equations. It translates the torque over a given axis that is created by the angular velocity over the remaining axis.

$$\tau_{bodygyro}^{BF} = \begin{bmatrix} \dot{\theta}\dot{\psi}(\mathcal{I}_{yy} - \mathcal{I}_{zz}) \\ \dot{\phi}\dot{\psi}(\mathcal{I}_{zz} - \mathcal{I}_{xx}) \\ \dot{\theta}\dot{\phi}(\mathcal{I}_{xx} - \mathcal{I}_{yy}) \end{bmatrix} \tag{2.19}$$

• **Propeller gyroscopic effect**

This component is mentioned by Bouabdallah [43] and results from a change in the angular velocity of the motors and takes effect over a 90 degree phase, making the roll dependent on pitch accelerations and vice-versa.

$$\tau_{propgyro}^{BF} = \begin{bmatrix} J_r \dot{\theta} \Omega_r \\ -J_r \dot{\phi} \Omega_r \\ J_r \dot{\Omega}_r \end{bmatrix} \tag{2.20}$$

where $\Omega_r = \sum_{i=1}^{6} (-1)^i \Omega_i$, with Ω_i as the velocity of the ith rotor, and J_r is the inertia of the set of propeller and rotating component of the motor.

Mathematical Model Result

The Newton-Euler (2.12) is applied to our system, as discussed, and result on the final laws of motion, in the Body Frame, defined in (2.21).

$$
\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} \dot{\theta}w - \dot{\psi}v - (c_\psi s_\theta c_\phi + s_\psi s_\phi)g - \dfrac{1}{2m} C A_x \rho u|u| \\ \dot{\psi}u - \dot{\phi}w - (s_\psi s_\theta c_\phi - c_\psi s_\phi)g - \dfrac{1}{2m} C A_y \rho v|v| \\ \dot{\phi}v - \dot{\theta}u - c_\theta c_\phi g + \frac{1}{m}(-\dfrac{1}{2} C A_z \rho w|w| + \sum_{i=1}^{6} C_T \rho A (\Omega_i R_{prop})^2) \end{bmatrix}
$$

$$
\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \dfrac{\dot{\theta}\dot{\psi}(\mathcal{I}_{yy} - \mathcal{I}_{zz}) + l \sum_{i=1}^{6} T_i \sin(\frac{i-1}{6} 2\pi) + J_r \dot{\theta} \Omega_r)}{\mathcal{I}_{xx}} \\ \dfrac{\dot{\phi}\dot{\psi}(\mathcal{I}_{zz} - \mathcal{I}_{xx}) + l \sum_{i=1}^{6} T_i \cos(\frac{i-1}{6} 2\pi) - J_r \dot{\phi} \Omega_r}{\mathcal{I}_{yy}} \\ \dfrac{\dot{\phi}\dot{\theta}(\mathcal{I}_{xx} - \mathcal{I}_{yy}) + \sum_{i=1}^{6} (-1)^i Q_i + J_r \dot{\Omega}_r)}{\mathcal{I}_{zz}} \end{bmatrix} \tag{2.21}
$$

2.4 Hardware

2.4.1 Hexacopter

The Hexacopter used in this project is the model MK-Hexa2 produced by Mikrokopter[2]. The frame, motors, propellers and motor's Ellectronic Speed Controllers (ESC) are the original ones. The motors used are Roxxy BL 2827-35[3] with 10" propellers with an advance distance[4] of 4.5". The main board from Mikrokopter, Fligh Management Unit (FMU), was replaced by the Pixhawk's PX4FMU, described in Section 2.4.6. The Remote Control (RC) used is the Spektrum DX6i, which is paired with the Pixhawk.

In the final implementation, the total mass of the hexacopter is 1.9kg without any tethering cables and 2.11kg with power and communication cables, measured at 1m above the ground. To ensure maneuverability, the multirotor should not weigh more than half of its maximum thrust. From the motor specifications, the maximum weight with which it is possible to hover is 4.92kg, equivalent to the maximum weight advised of 2.46kg. A weight of 2.11kg proved to be a high value for fast maneuvers, but with which it is possible to control the multirotor autonomously.

Other devices were added: BeagleBone, Laser Range Finder and Ultrasonic sensor, which will be introduced over the next Sections, and also a 12V-5V voltage converter.

2.4.2 Ground station

The hexacopter does not have a battery. Its power is supplied by a ground station through a tether. This station contains a 24V battery and a voltage converter that will accept an input up to 36V and will output 75V. A high voltage of 75V was previously chosen in order to reduce the current, allowing the use of a thinner transmission cable in the tether and reduce electrical noise. On the other hand, there is a need for a 75V-12V converter on the drone that is able to withstand high currents. This implementation brings

[2] `http://www.mikrokopter.de/en/home`

[3] `http://mikrokopter.altigator.com/motor-roxxy-282735-mikrokopter-special-110w-p-83.html`

[4] Advance distance is the distance perpendicular to the propeller that it ideally moves through the fluid (in our case, air) during one revolution.

some troubles, as the motors can return spikes of current back to the converter and this might burn it. Because of this problem, the ground station became nonoperational and an improvised tether with an Ethernet cable for communication and power cables connected to a 12V power supply was constructed. This approached increased the weight of the tether considerably (over 3 times), but was the only option available at the end of the project.

2.4.3 BeagleBone Black

A BeagleBone Black[5] was used as a companion computer for the Pixhawk. It is used to read and process information coming from the laser sensor. Using this information, it computes the localization in the x and y dimensions, and also provides a yaw angle estimate. The BeagleBone is also responsible for receiving and interpreting user commands. This enables the user to send commands from the ground-based computer to the BeagleBone, using high-level orders, and then these orders will be processed and converted to lower-level messages, which are sent to the Pixhawk. It communicates with the Pixhawk through a UART-UART connection using the MAVLINK protocol, explained in Section 3.1.2, and also communicates with a ground-based computer through an Ethernet cable, using an SSH session.

2.4.4 Laser Range Finder

The Laser Range Finder used is the model URG-04LX[6] produced by Hokuyo[7]. This laser has a maximum range of 4m, a measuring range of 240 degrees, 1mm resolution and gives readings at a 10Hz sampling frequency. It is adequate for indoor environment because of its high resolution and accuracy, having only 1-4cm measurement error, depending on the distance measured.

The device consists of a transmitter that illuminates the obstacle with a laser beam and a receiver that is able to detect the reflected beam, which is coaxial with the original signal. A rotating mirror sweeps the 2-dimensional plane to get the measurement of distance to obstacles in a 240^o field of view. The principle of distance measurement of this sensor is based on computing the phase difference. A laser beam with sinusoidal modulation is sent to a target. The reflected light is monitored, and the phase of the received signal is compared with that of the sent light. The phase shift obtained is 2π times the time-of-flight times the modulation frequency. From this, it is possible to obtain a stable measurement with minimum influence of the object's color and reflectance.

2.4.5 Ultrasonic Sensor

The ultrasonic sensor, also called sonar, used is I2CXL-MaxSonar-EZ4[8] (MB1242). It has a 10Hz sampling frequency with 1cm resolution. The measuring ranges from 0.2m to 7.65m. Since the sonar is put at an approximate height of 14cm above the ground, it is not able to measure the correct distance

[5] http://beagleboard.org/BLACK
[6] http://www.hokuyo-aut.jp/02sensor/07scanner/download/products/urg-04lx/
[7] http://www.hokuyo-aut.jp
[8] http://www.maxbotix.com/documents/I2CXL-MaxSonar-EZ_Datasheet.pdf

when the hexacopter is landed on the ground. It has a narrow beam, which was chosen in order not to easily detect objects on the surroudings, such as walls, but it is wide enough that it can detect the closest point in the ground when it is slightly tilted ($<$10 degrees). A filter was constructed to reduce conducted electrical noise coming from the power supply, since this is considered one of the main causes of measurement noise for this sensor. Another source of noise is the wind coming from the propellers, which can deflect the ultrasonic waves. Hence the sonar was put under the hexacopter, as further away from the propellers as possible. Unlike other sonar sensors, there is no need to add additional code to run this sensor at a low-level, as it already outputs the distance measured directly through I^2C.

2.4.6 Pixhawk

The Pixhawk is a computer that runs a Flight Management Unit (FMU) called PX4FMU and also manages several built-in sensors and, optionally, external sensors. It is open-hardware, with the information available in GitHub[9]. It originally includes accelerometer, gyroscope and barometer sensors. The sonar sensor is connected through I^2C, the BeagleBone through UART and the radio receiver is connected to the SPKT port.

2.5 Software

2.5.1 PX4 Autopilot

There is more than one FMU ready-to-use firmware available. The PX4 autopilot was the chosen one, because it is divided into modules that can be modified and replaced individually. It runs the real-time operating system NuttX and is open-source, with all the information available in GitHub[10], where the firmware is available here[11]. The firmware modules that manage the main control loop are shown in Figure C.1, where the modified modules and fields are highlighted in blue. The control loop runs at 250Hz.

The Pixhawk has different behaviors that are defined as flight modes. The flight modes addressed in this project are: Manual, ALTCTL, POSCTL and Offboard. The Manual mode provides direct control over the 4DoF: thrust, roll, pitch and yaw, using a RC. ALTCTL stands for altitude control and enables the user to control, with the RC, the altitude rate and manually control the attitude of the hexacopter. POSCTL stands for position control and provides an easy way to control the hexacopter. In this mode, the user can neglect the roll and pitch, and focus only on controlling the yaw angular velocity and the x, y and z velocities. The Offboard mode allows the UAV to be controlled from a source external to the Pixhawk. The level of control that is implemented is up to the developer. This mode can be dangerous, depending on what is implemented.

To reach the goals of this project, it is necessary to master the Offboard mode. To accomplish this, all the remaining modes have to be working correctly. It is also necessary to have the supporting software

[9] https://github.com/PX4/Hardware
[10] https://github.com/PX4
[11] https://github.com/PX4/Firmware

in charge of the navigation, which will be computed in the BeagleBone. In this project, when using the Offboard mode, part of the control loop is on the BeagleBone, but the UAV is fully controllable if the Offboard mode is disabled at any time. Part of the pose estimation is done on the BeagleBone side and also the trajectory planning. Putting all the control loop in the BeagleBone was considered, but it was concluded not to be the best solution. It is not advised to move all the control loop to the BeagleBone and only send low-level motor commands to the Pixhawk. In this case, if there is a communication issue during flight, there will be unexpected behavior on the helicopter and it will not even be possible to control the drone manually using the RC, since there is no control loop running on the Pixhawk. The Pixhawk communicates with the BeagleBone using the MAVLink protocol, which will be explained in Section 3.1.2.

In order for the flight modes to be enabled and working correctly, the position and attitude estimators had to be changed. The first was created from scratch, while the latter was slightly modified. The attitude and position controllers had to be tuned in order to function properly, but the original architecture from the PX4FMU was used. The hexacopter needs to be armed for the motors to become operational. This is achieved by pressing a physical safety switch and then by activating the arming state with the RC. The motors will then start spinning at minimum thrust, meaning that the hexacopter is fully operational.

2.5.2 Mobotware

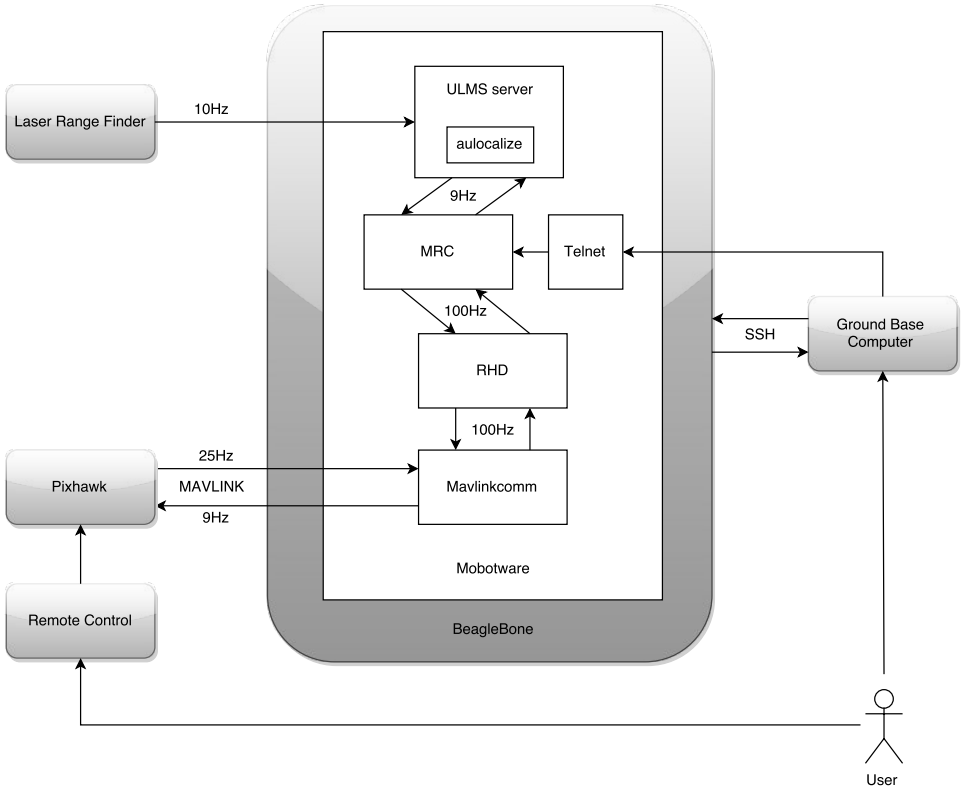


Figure 2.5: Mobotware architecture and interaction with external devices. Rectangular white boxes represent software applications and grey boxes represent hardware devices. The user is able to access the Mobotware software through a ground-based computer. The user can send commands to the Pixhawk directly through a Remote Control or by sending commands through Mobotware (accessing Mobotware through the ground-based computer).

Mobotware is DTU's real-time control software. It is used in this project and modified to the project's needs. This software consists on a central program called Robot Hardware Daemon (RHD) [44] and other applications running in parallel, where the ones used in this project are: Mobile Robot Control (MRC) and ULMS[12]. All these programs are briefly presented in this Section, and their role in the global system is stated. Further details are available in Appendix A, along with all the necessary information to run the version of this software implemented on this project's hexacopter; the implementation details that a user needs to know in order to put the software running on another UAV system, that uses the PX4 autopilot, are also explained.

Figure 2.5 illustrates the communication architecture of Mobotware, running on a BeagleBone[13], and introduces the external devices that interact with it: laser scanner, Pixhawk and a ground-based computer. The grey boxes represent hardware devices and the rectangular white boxes represent software applications, with connecting arrows representing the direction of the communication flow.

Robot Hardware Daemon

Robot Hardware Daemon (RHD) is a real-time synchronized database that contains all the relevant variables that are shared between programs, such as the position of the robot. It is defined to run at a fixed sample rate of 100Hz. This frequency will be the one at which the database is synchronized and also defines how often new information that is received/sent from/to the Pixhawk is updated, on the database. RHD is the main time-base for the low-level robot control applications, such as MRC. It is vital that RHD's periodical routines work faster than the frequency at which they are called, to ensure that other applications work at a fixed frequency. The obstruction of the RHD periodical routine can be a source of delays in the whole Mobotware system.

The RHD core framework itself does not supply an implementation for hardware interaction. To enable a specific interaction, drivers are implemented through a plugin structure. For this project, interaction with the Pixhawk is needed, hence an RHD plugin called MavlinkComm is created, which will be discussed in Section 3.1.2. The plugin MavlinkComm can interface the variable database to communicate with MRC and is also able to interface with the Pixhawk.

Mobile Robot Control

Mobile Robot Control (MRC) is a low-level application, running in soft real-time, that is in charge of tasks related to robot control. It is used in this project to:

- manage the localization algorithm running in the ULMS server,
- as an entry point for the user to enter commands,
- manage the information flow between the entities ULMS↔RHD and user↔RHD.

MRC is responsible for getting information to and from the ULMS server regarding the robot's pose. There were several approaches taken to deal with MRC; the final one uses a Telnet client to control MRC. This approach is described further in Appendix A.2.

[12] More information about ULMS in `http://rsewiki.elektro.dtu.dk/index.php/Ulmsserver`

[13] Mobotware is running on a computer located on top of the hexacopter. The computer chosen was a BeagleBone Black.

Telnet Client

A Telnet client is used to enable the user to give command orders to the hexacopter. The user can employ the commands presented in AppendixA.3 to:

- change flight mode;
- give x, y and z position setpoints for the hexacopter to go to;
- give yaw setpoints to change the hexacopter's heading.

The commands are sent from a ground-based computer to the BeagleBone, overriding the Remote Control. The Telnet Client was also created to meet other requirements, which are explained in AppendixA.3, such as controlling what is executed by the MRC.

ULMS Server

ULMS is the name of the server in charge of communicating with the laser scanner, processing its information and running the algorithms that use such information. It communicates with MRC and exchanges pose information about the robot. A ULMS plugin, *aulocalize*, is used to compute the localization algorithm presented in Section 2.7. This plugin can be viewed as a function called periodically from the MRC. It takes as input a 2-D map of the environment and pose information from the Pixhawk; it gives as output the corrections to the pose information it receives, based on the laser scanner information. The flow of information to and from ULMS is handled by the MRC.

2.6 Extended Kalman Filter

2.6.1 Introduction

In this section, a solution for the height estimation, *i.e.*, altitude above the ground rather than altitude above mean sea level (AMSL), is presented. This approach is an Extended Kalman Filter (EKF) that fuses information from several sensors. These sensors give information about a body that is free to move along the z axis with position p, velocity v and acceleration a. The approach will be based on the methodology explained by Farrell [45] and will serve as an introduction for the reader to understand the Kalman Filters implemented in this project, whose study is done in Sections 3.2 and 3.3, and the associated design choices.

2.6.2 Sensor model

In order to design an EKF, the sensors need to be modeled and design assumptions must be made. Three navigation sensors are available: an accelerometer, a barometer and an ultrasonic sensor.

The accelerometer provides acceleration information sampled at $f_1 = 250$Hz and its measurement can be modeled as

$$u = a - \alpha a - b_u - \eta_1 \tag{2.22}$$

where a is the real acceleration and η_1 is a Gaussian white noise process with Power Spectral Density (PSD) equal to $\sigma^2 = (10^{-3})^2 \frac{(m/s^2)^2}{Hz}$. The symbol α represents a constant scale factor error with distribution $\alpha(0) \sim N(0, 0.01^2)$, and it is estimated in this KF. Hence the estimate of this value will be used on the model, not posing as a non-linearity on the KF, given its value becomes a constant, not a probabilistic variable. The accelerometer output signal u contains a bias b_u that is modeled as a constant plus random walk process

$$\dot{b}_u = \omega_1. \tag{2.23}$$

The uncorrelated white Gaussian noise ω_1 is characterized by a PSD equal to $\sigma^2 = 0.01^2 \frac{(m/s^3)^2}{Hz}$. The offset b_u has initial conditions $E[b_u(0)] = \mu_{b_u}$ and $var(b_u(0)) = P_{b_u}(0) = 0.01^2$.

The barometer gives information about pressure, which is translated into an altitude measurement. The altitude AMSL is converted into height above the ground by removing an offset. The offset is included as a constant part of the bias. The altitude information is sampled at $f_2 = 100$Hz and can be modeled as

$$y_{bk} = p_k + b_y(k) + \eta_2(k) \tag{2.24}$$

where the term η_2 represents white Gaussian noise that can be split into two types of Gaussians [46]: a first-order Gauss-Markov random process and uncorrelated random process. The first arises from local environment changes in pressure, such as opening a door, which have a short term impact (a few seconds), while the latter has its origin especially on the quantization noise due to the low resolution of the sensor (10cm). The Gauss-Markov noise will be ignored as it only appears sporadically and is unimportant. The quantization noise will be treated as regular white Gaussian noise on this Section, but will be discussed later on Section 3.2.3. The discrete-time noise samples η_{2k} have variance $\sigma_2^2 = 0.1^2$, if only taking into account the quantization noise. The barometer signal has a bias $b_y(m)$ component that is modeled as a slow time-varying signal

$$\dot{b}_y = -\lambda_y b_y + \omega_2 \tag{2.25}$$

where λ_y is an inverse of correlation time of Gauss–Markov random process for the bias [47], estimated as $\lambda_y = 1/100$. The uncorrelated white Gaussian noise ω_2 is characterized by a PSD equal to $\sigma^2 = 0.1^2 \frac{(m/s^3)^2}{Hz}$. The offset b_y has initial conditions $E[b_y(0)] = \mu_{b_y}$ and $var(b_y(0)) = P_{b_y}(0) = 0.2^2$. The bias already comprises a fixed offset that arises from the difference between the altitude AMSL and the altitude of the ground. This value is calculated in the initialization step prior to running the EKF, thus it is omitted in this chapter's analysis.

The ultrasonic sensor provides position information sampled at $f_3 = 10$Hz and is modeled as

$$y_{sm} = p_m \cos(roll) \cos(pitch) + \eta_3(m) \tag{2.26}$$

The discrete-time noise samples $\eta_3(m)$ are a combination of 4 types of noise:

1. Gaussian with zero mean and variance $\sigma_3^2 = 0.01^2$.
2. spikes in the measurement, which may come from electrical noise coming from the power supply

or acoustic noise arising from wind, motion or the environment.

3. Detection of objects nearby, including walls.
4. Inaccurate measurements arising from big roll and pitch angles ($<$10degrees).

Measurements that are affected by noise of types 2 to 4 can be distinguished from ones with only noise of type 1. Thus, let us consider that readings containing types 2 to 4 can be identified by validation gates and are automatically rejected. Thus the sonar sensor can be modeled as contain only white Gaussian noise with zero mean and variance $\sigma_3^2 = 0.01^2$.

The goal of the EKF is to estimate the position and velocity at rate $f_1 = 250$Hz. Although, due to the sensor models designed, it is also needed to estimate the biases of the accelerometer and barometer sensors, as well as the accelerometer's scale factor.

2.6.3 Augmented Kinematic Model

The system dynamics are based on the kinematics of a free body:

$$\begin{aligned} \dot{p} &= v \\ \dot{v} &= a. \end{aligned} \tag{2.27}$$

Our model is sensor-based, therefore the sensor parameters need to be included into the kinematics system, creating the augmented kinematic system

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, a, \boldsymbol{\omega}) = \begin{bmatrix} \dot{p} \\ \dot{v} \\ \dot{b}_u \\ \dot{b}_y \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} x_2 \\ a \\ \omega_1 \\ -\lambda_y x_4 + \omega_2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \eta_1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{where} \quad \boldsymbol{x} = \begin{bmatrix} p \\ v \\ b_u \\ b_y \\ \alpha \end{bmatrix} \quad \text{and} \quad \boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \eta_1 \end{bmatrix}. \tag{2.28}$$

The model includes the process noise $\boldsymbol{w}$ because it represents the real system, according to the models designed; the variables b_u and b_y vary with noise. The initial value of the augmented state and its initial covariance are

$$\boldsymbol{x_0} = \begin{bmatrix} E[p(0)] \\ E[v(0)] \\ \mu_{b_u} \\ \mu_{b_y} \\ 0 \end{bmatrix} \quad \text{and} \quad \boldsymbol{P}_x(0) = diag\left(\begin{bmatrix} P_p(0) \\ P_v(0) \\ 0.01^2 \\ 0.2^2 \\ 0.01^2 \end{bmatrix} \right) \tag{2.29}$$

2.6.4 Navigation Mechanization Equations

The equations that deliver the navigation state of the vehicle (position and velocity) are often called navigation mechanization equations:

$$\dot{\hat{\boldsymbol{x}}}_0 = f(\hat{\boldsymbol{x}}, \hat{a}, \boldsymbol{\omega}), \tag{2.30}$$

where the state estimate is $\hat{\boldsymbol{x}} = \begin{bmatrix} \hat{p} \\ \hat{v} \\ \hat{b}_u \\ \hat{b}_y \\ \hat{\alpha} \end{bmatrix}$, and from (2.22) one has $\hat{a} = \frac{u+\hat{b}_u}{1-\hat{\alpha}}$, resulting in

$$f(\hat{\boldsymbol{x}}, \hat{a}, \mathbf{0}) = \begin{bmatrix} \hat{v} \\ \hat{a} \\ 0 \\ -\lambda_y \hat{b}_y \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{x}_2 \\ \frac{u+\hat{x}_3}{1-\hat{x}_5} \\ 0 \\ -\lambda_y \hat{x}_4 \\ 0 \end{bmatrix}. \tag{2.31}$$

In this equation, $\boldsymbol{w} = 0$ since the estimates do not contain noise. A possible numerical integration of (2.31) can be written for a more intuitive understanding as

$$\begin{aligned} \hat{p}_{j+1} &= \hat{p}_j + \hat{v}_j dt + 1/2 \hat{a}_j dt^2 \\ \hat{v}_{j+1} &= \hat{v}_j + \hat{v}_j \hat{a}_j dt \\ \hat{b}_{u_{j+1}} &= \hat{b}_{u_j} \\ \hat{b}_{y_{j+1}} &= e^{-\lambda_y dt} \hat{b}_{y_j} \\ \hat{\alpha}_{j+1} &= \hat{\alpha}_j \end{aligned} \tag{2.32}$$

where $dt = \frac{1}{f_1}$. The state integration is done at the f_1 rate, represented by the index j, while the measurement iterations in (2.24) and (2.26) are done at f_2 and f_3 rates, which are represented by time indexes k and m, respectively. Since $f_1 > f_2$ and $f_1 > f_3$, often there is no new position information from either the barometer or sonar. When there is a new reading from one of the position sensors, a new measurement update is made. At this time, the j index is reset to zero and the result of the measurement update, $\hat{\boldsymbol{x}}_0^+$, serves as the initial condition to the next state integration. This way, the state is propagated using the accelerometer information when there is no position observability due to lack of position information. Nonetheless, every time a new position measurement arrives from one of the position sensors, the state is corrected and used as initial state for the next integration cycle.

2.6.5 Error Models

The error state will be defined as $\delta \boldsymbol{x} = \boldsymbol{x} - \hat{\boldsymbol{x}}$. The dynamic equation for the state error vector, in the form $\delta \dot{\boldsymbol{x}} = \boldsymbol{F} \delta \boldsymbol{x} + \boldsymbol{G} \boldsymbol{\omega}$, is

$$\delta \dot{\boldsymbol{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & u \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\lambda_y & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \delta \boldsymbol{x} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \boldsymbol{\omega}, \tag{2.33}$$

which is the difference between (2.28) and (2.31). The term $a - \frac{u+\hat{x}_3}{1-\hat{x}_5}$ was linearized when x is near $\hat{x}$ using the fact that if $|\delta x| << 1$, then $\frac{1}{1-\delta x} \approx 1 + \delta x$, which lead to $a - \frac{u+\hat{x}_3}{1-\hat{x}_5} \approx \delta x_3 + u\delta x_5 + \eta_1$.

Analogously, the output error model for the barometer measurement is

$$\delta y_{bk} = \delta x_1(k) + \delta x_4(k) + \eta_2(k), \tag{2.34}$$

which is derived by linearizing the difference between (2.24) and the barometer altitude estimate

$$\hat{y}_{bk} = \hat{p}_k + \hat{b}_y(k). \tag{2.35}$$

The same linearization is applied to the 2.26 and the sonar altitude estimate

$$\hat{y}_{sm} = \hat{p}_m, \tag{2.36}$$

leading to the output error model

$$\delta y_{sm} = \delta x_1(m) + \eta_3(m). \tag{2.37}$$

The error models of each sensor measurement can be written in matrix form as

$$\begin{aligned} \begin{bmatrix} \delta y_{bk} \\ \delta y_{sm} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{H}_1 \\ \boldsymbol{H}_2 \end{bmatrix} \delta \boldsymbol{x} + I \begin{bmatrix} \eta_2 \\ \eta_3 \end{bmatrix} \\ \begin{bmatrix} \delta y_{bk} \\ \delta y_{sm} \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \delta \boldsymbol{x} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \eta_2 \\ \eta_3 \end{bmatrix}, \end{aligned} \tag{2.38}$$

where δy_{bk} and δy_{sm} are decoupled, despite the representation in one matrix equation that involves both measurement matrices $\boldsymbol{H}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \end{bmatrix}$ and $\boldsymbol{H}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}$.

2.6.6 Observability Analysis

Our model is sensor-based, which means that the estimation of the state relies on sensor outputs. When the error state is not observable, it may not be possible to determine the state $\boldsymbol{x}$ accurately at that instant. Even though it is feasible to estimate the state of the system, it is not known how precise that estimate is.

The Observability matrices of the error state when there is a measurement from the barometer, $\mathcal{O}_b$, and sonar, $\mathcal{O}_s$, are

$$\mathcal{O}_b = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -\lambda_y & 0 \\ 0 & 0 & 1 & \lambda_y^2 & u \\ 0 & 0 & 0 & -\lambda_y^3 & 0 \\ 0 & 0 & 0 & \lambda_y^4 & 0 \end{bmatrix} \quad \text{and} \quad \mathcal{O}_s = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & u \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.39}$$

The states p, v and b_u are observable when there is a position measurement, b_y is only observable when there is a barometer measurement and the observability of the state α is dependent on the accelerometer's measurement. α can only estimated if the measurement u is non-zero; furthermore, the

bigger the measurement, the more informative it can be.

2.6.7 Discrete-time State Model

As the state estimate is integrated at f_1, error between state estimate and real state will grow due to imperfections in the measurement of u. To correct the estimates at rate f_2 and f_3, the accuracy of the state estimation error has to be characterized. For this, one relies on the matrices Φ and Q_d, which will propagate the error covariance matrix P from t_k to t_{k+1} and t_m to t_{m+1}.

The discrete model, equivalent to 2.33 sampled at instants $t_k = kT$ and $t_m = mT$, is:

$$\begin{aligned} \delta \boldsymbol{x}_{k+1} &= \boldsymbol{\Phi}_k \delta \boldsymbol{x}_k + \boldsymbol{w}_k \\ \delta \boldsymbol{x}_{m+1} &= \boldsymbol{\Phi}_m \delta \boldsymbol{x}_m + \boldsymbol{w}_m. \end{aligned} \tag{2.40}$$

Since the dynamic coefficient matrix $\boldsymbol{F}$ from (2.33) is not constant between time intervals $t \in [t_k, t_{k+1}]$ or $t \in [t_m, t_{m+1}]$, these intervals need to be divided into smaller uniformly distributed intervals of length τ where u is approximately constant. The smaller the value of τ, the better performance is achieved, but the more computationally expensive the algorithm becomes; the best performance is obtained with $\tau = \frac{1}{f_1}$. (2.33) can be represented in a discrete-time model that is equivalent to the continuous-time model at the discrete-time instants $t_k = k\tau$; equivalence means that the discrete and continuous-time models predict the same system state at the specified discrete-time instants. If we find a sufficiently small time interval τ in which we consider constant acceleration, $\boldsymbol{F}$ can be considered as a constant matrix. The Discrete-time State Transition matrix $\boldsymbol{\Phi} = e^{\boldsymbol{F}\tau}$ is obtained using a second order Taylor series approximation:

$$\boldsymbol{\Phi}(\tau_i, \tau_{i-1}) = e^{\boldsymbol{F}\tau} = \boldsymbol{\mathcal{I}}_{5\times 5} + \boldsymbol{F}\tau + \frac{1}{2}(\boldsymbol{F}\tau)^2. \tag{2.41}$$

The Discrete-time State Transition matrix becomes (2.42), where the resulting term $1 - \lambda_y\tau - (\lambda_y\tau)^2$ in $\Phi_{4,4}$ was considered as being the approximation of $e^{-\lambda_y\tau}$. This matrix is accumulated over time using (2.43) until $\boldsymbol{\Phi}_{k-1} = \boldsymbol{\Phi}(\tau_N, \tau_0)$ is reached.

$$\boldsymbol{\Phi}(\tau_i, \tau_{i-1}) = \begin{bmatrix} 1 & \tau & \frac{\tau^2}{2} & 0 & u\frac{\tau^2}{2} \\ 0 & 1 & \tau & 0 & u\tau \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & e^{-\lambda_y\tau} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.42}$$

$$\boldsymbol{\Phi}(\tau_{i+1}, \tau_0) = \boldsymbol{\Phi}(\tau_{i+1}, \tau_i)\boldsymbol{\Phi}(\tau_i, \tau_0) \tag{2.43}$$

The state is propagated at f_1=250Hz by computing

$$\hat{\boldsymbol{x}}_{\tau_i} = \boldsymbol{\Phi}(\tau_i, \tau_{i-1})\boldsymbol{x}_{\tau_{i-1}}, \tag{2.44}$$

where $\boldsymbol{\Phi}(\tau_i, \tau_{i-1})$ is written in (2.42). $\boldsymbol{Qd}$ can be computed by numeric integration using (2.45), in order

to calculate $\boldsymbol{Qd}_{k-1} = \boldsymbol{Q}(\tau_N, \tau_0)$.

$$\begin{gathered}\boldsymbol{Q}(\tau_i, \tau_0) = \boldsymbol{\Phi}(\tau_i, \tau_{i-1})\boldsymbol{Q}(\tau_{i-1}, \tau_0)\boldsymbol{\Phi}^T(\tau_i, \tau_{i-1}) + \boldsymbol{G}\boldsymbol{Q}_{\boldsymbol{\omega}}\boldsymbol{G}^T\tau, \\ \text{where} \quad \boldsymbol{Q}_w = diag\left(\left[\sigma_{b_u}^2, \sigma_{b_y}^2, \sigma_1^2\right]\right)\end{gathered} \tag{2.45}$$

2.6.8 State Propagation in Time

Epoch n is defined to be the most recent discrete instant. n will be k if the last altitude measurement came from the barometer or m if a sonar measurement is received more recently.

Given $\boldsymbol{\Phi}$ and $\boldsymbol{Q}$ at time $k-1$ or $m-1$, the covariance time propagation can be performed as

$$\boldsymbol{P}_n^- = \boldsymbol{\Phi}_{n-1}^+\boldsymbol{P}_{n-1}^+\boldsymbol{\Phi}_{n-1}^T + \boldsymbol{Qd}_{n-1} \tag{2.46}$$

With the propagated covariance matrix $\boldsymbol{P}$, the Kalman Gain vector can be computed when a new position measurement is available, as in (2.47). When the measurement comes from the barometer, $R = \sigma_2^2$ and $\boldsymbol{H} = \boldsymbol{H}_1$ are used. When the measurement comes from the sonar sensor, $R = \sigma_3^2$ and $\boldsymbol{H} = \boldsymbol{H}_2$ are used.

$$\boldsymbol{K}_n = \boldsymbol{P}_n^-\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_n^-\boldsymbol{H}^T + R)^{-1} \tag{2.47}$$

Now that everything needed has been computed, the correction of the state vector is performed and the covariance matrix $\boldsymbol{P}$ is updated:

$$\begin{aligned}\hat{\boldsymbol{x}}_n^+ &= \hat{\boldsymbol{x}}_n^- + \boldsymbol{K}_n(y_n - \hat{y}_n^-) \\ \boldsymbol{P}_n^+ &= (\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}_n)\boldsymbol{P}_n^-,\end{aligned} \tag{2.48}$$

where y_n^- can be either $y_k^- = \boldsymbol{H}_1\hat{x}_k^-$ or $y_m^- = \boldsymbol{H}_1\hat{x}_m^-$, depending on the measurement coming from the barometer or sonar. If at any epoch k or m there is not a valid measurement, the state and covariance matrix are defined as

$$\begin{aligned}\hat{\boldsymbol{x}}_n^+ &= \hat{\boldsymbol{x}}_n^- \\ \boldsymbol{P}_n^+ &= \boldsymbol{P}_n^-,\end{aligned}$$

which accounts for not having a new measurement.

2.6.9 Summary

The Extended Kalman Filter can be summarized recurring to the equations already presented. The initialization is performed firstly, using (2.29) to initialize the state $\boldsymbol{x}$ and the covariance matrix $\boldsymbol{P}$.

The Filter performs a Measurement Prediction by computing

$$\hat{\boldsymbol{x}}_n^- = f(\hat{\boldsymbol{x}}_{n-1}^+, \hat{\boldsymbol{a}}_n, \boldsymbol{0}). \tag{2.49}$$

The function $f(\hat{\boldsymbol{x}}_{n-1}^+, \hat{\boldsymbol{a}}_n, \boldsymbol{0})$ was obtained by computing a discrete-time equivalent model. This model is the Discrete-time State Transition matrix $\boldsymbol{\Phi}$, computed using a second order Taylor expansion, presented in (2.41). (2.44) is computed at f_1=250Hz to propagate the system state. (2.45) needs to be computed

at f_1=250Hz to propagate the process error. The state propagation phase also includes the propagation of the covariance matrix over time, which is written as (2.46), and is computed before the Measurement Update step.

The Measurement Update is executed by computing (2.47), followed by (2.48).

Algorithm 1 Extended Kalman Filter

procedure EKF($\boldsymbol{x}_0$, $\boldsymbol{Q}_{\boldsymbol{\omega}}$)
Prediction Step:
  $\boldsymbol{\Phi}(\tau_{i+1}, \tau_0) \leftarrow \boldsymbol{\Phi}(\tau_{i+1}, \tau_i)\boldsymbol{\Phi}(\tau_i, \tau_0)$
  $\hat{\boldsymbol{x}}_{\tau_i}^- \leftarrow \boldsymbol{\Phi}(\tau_i, \tau_{i-1})\boldsymbol{x}_{\tau_{i-1}}$
  $\boldsymbol{Q}(\tau_i, \tau_0) \leftarrow \boldsymbol{\Phi}(\tau_i, \tau_{i-1})\boldsymbol{Q}(\tau_{i-1}, \tau_0)\boldsymbol{\Phi}^T(\tau_i, \tau_{i-1}) + \boldsymbol{G}\boldsymbol{Q}_{\boldsymbol{\omega}}\boldsymbol{G}^T\tau$
  if new_sonar_measurement() = false **and new_barometer_measurement()** = false **then**
    goto *Prediction Step*
  $\boldsymbol{P}_n^- \leftarrow \boldsymbol{\Phi}_{n-1}^+\boldsymbol{P}_{n-1}^+\boldsymbol{\Phi}_{n-1}^T + \boldsymbol{Q}(\tau_i, \tau_0)$
Update Step:
  $\boldsymbol{K}_n \leftarrow \boldsymbol{P}_n^-\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P}_n^-\boldsymbol{H}^T + R)^{-1}$
  $\hat{\boldsymbol{x}}_{\tau_i}^+ \leftarrow \hat{\boldsymbol{x}}_{\tau_i}^- + \boldsymbol{K}_n(y_n - \hat{y}_n^-)$
  $\boldsymbol{P}_n^+ \leftarrow (\boldsymbol{I} - \boldsymbol{K}_n\boldsymbol{H}_n)\boldsymbol{P}_n^-$
  $\boldsymbol{\Phi}(\tau_i, \tau_0) \leftarrow \boldsymbol{0}$
  $\boldsymbol{Q}(\tau_i, \tau_0) \leftarrow \boldsymbol{0}$
  goto *Prediction Step*

The Prediction Step is followed by the Update Step every time a measurement from the sonar or barometer is received, *i.e.*, at epoch k or m. At this time, the index relative to the state propagation is reset to zero and the result of the measurement update, $\boldsymbol{x}_0^+$, serves as the initial condition to the next state integration. While a new measurement is not received, the Prediction Step propagates the state and its associated covariance matrix.

The initial estimate of the system's state should be according to what is expected from the system at the initialization time. If the system starts when the vehicle is motionless on the ground, one suggestion is: $E[p(0)] = 0$, $E[v(0)] = 0$, $E[\alpha(0)] = 0$, $E[b_u(0)] = \mu_{b_u}$ and $E[b_y(0)] = \mu_{b_y}$, where μ_{b_u} and μ_{b_y} are the mean value of 100 measurements from the accelerometer and barometer, respectively. The initial covariance diagonal values are dependent on the degree of confidence of the values select for $\boldsymbol{x}_0$.

2.7 Kalman Filter localization with line feature extraction

In order to reduce the tracking error, a feature-based[14] self-localization algorithm is used, where the features are line segments. The algorithm described by Siegwart and Nourbakhshin [12] is used to handle the laser range measurements in order to predict the current absolute pose of the hexacopter in a 2-D space. A brief explanation of this method is presented. This algorithm takes as inputs a map $\mathcal{M}$ and the initial pose of the robot in map coordinates with the corresponding variance. In each iteration, the algorithm uses as inputs: pose estimates (optional) and laser scanner measurements. $\mathcal{M}$ is a 2-D metric map of the environment described as $\mathcal{M}$=$\{L_1,L_2,...,L_N\}$, where L_m represents the line segment with origin O_m, angle α_m and length l_m in the EF. In this Section, when mentioning the previous localizer

[14]González at al. [48] define feature-based approach as a method where a set of features is extracted from the sensed data and then matched against the corresponding features in the model.

iteration, t_{i-1}, it should be understood as the last iteration where the whole algorithm was run, and t_i as the time of the current iteration.

The algorithm is divided in 5 sequential steps:

1. Pose prediction

   1.1. Translational Motion: The distance traveled since the last localizer iteration, t_{i-1}, is estimated. The result of this step is the displacement of x and y, in odometry coordinates. Let us consider that we have x_0 and y_0 as the position at t_{i-1} and that J measurements of position, ${}^{OF}x$ and ${}^{OF}y$, were received since t_{i-1}, then the displacement is computed as:

   $$\Delta x_i^- = \sum_{j=1}^{J} {}^{OF}x_j - {}^{OF}x_{j-1} \quad , \quad \Delta y_i^- = \sum_{j=1}^{J} {}^{OF}y_j - {}^{OF}y_{j-1}. \tag{2.50}$$

   1.2. Rotational Motion: Analogously to the Translational Motion computation, a yaw displacement, $\Delta\psi_i^-$, is calculated based on all the yaw estimates received, ${}^{OF}\psi$, since t_{i-1}:

   $$\Delta\psi_i^- = \sum_{j=1}^{J} {}^{OF}\psi_j - {}^{OF}\psi_{j-1}. \tag{2.51}$$

   1.3. Pose Computation: The new pose is computed by adding the computed pose displacement of the robot $\Delta\boldsymbol{p_i}^- = \begin{bmatrix}\Delta x_i^- & \Delta y_i^- & \Delta\psi_i^-\end{bmatrix}^T$ to its pose at t_{i-1}:

   $$\hat{\boldsymbol{p}}_i^- = \hat{\boldsymbol{p}}_{i-1}^+ + \Delta\boldsymbol{p_i}^- = \begin{bmatrix}\hat{x}_i^- & \hat{y}_i^- & \hat{\psi}_i^-\end{bmatrix}^T. \tag{2.52}$$

   An alternative to the approach mentioned is to predict the pose based on a deterministic model, which relies on the pose known at t_{i-1} and the robot's movement due to the control input $\boldsymbol{u}(i-1)$:

   $$\hat{\boldsymbol{p}}_i^- = \hat{\boldsymbol{p}}^-(i|i-1) = f(\hat{\boldsymbol{p}}^+(i-1|i-1), \boldsymbol{u}(i-1)) = \begin{bmatrix}\hat{x}_i^- & \hat{y}_i^- & \hat{\psi}_i^-\end{bmatrix}^T. \tag{2.53}$$

   The Covariance is propagated as:

   $$\boldsymbol{P}(i|i-1) = \nabla_p f \cdot \boldsymbol{P}_p(i-1|i-1) \cdot \nabla_p f^T + \nabla_u f \cdot \boldsymbol{P}_u(i-1) \cdot \nabla_u f^T. \tag{2.54}$$

   In our case, the variables x, y and ψ are uncorrelated, which makes the Jacobian ∇_p equal to the identity matrix $\boldsymbol{I}_{3\times3}$. The measurement inputs are given as position and orientation estimates, which were previously computed without being correlated, hence $\nabla_u = \mathcal{I}_{3\times3}$. The independence between x, y and ψ can be assumed because the vehicle is considered to be holonomic.

2. Observation

   The information from the laser scanner is gathered and features are extracted from it, using RANSAC[15]. By using RANSAC, outlier measurements are spontaneously rejected, such as small

[15]RANSAC is an iterative algorithm that estimates a set of parameters in a model that best define the data set. By randomly choosing points from the data set and considering them as inliers, a different model is created and evaluated against the whole data set in each iteration. After some iterations, the model with biggest support is considered as the one that best fits the data and its parameters are chosen

round features. The features correspond to line segments represented by angle and length. The features are transformed from the EF to the local Sensor Frame, which coincides with BF. Features are represented in polar coordinates as

$$\boldsymbol{z}_m(i) = \begin{bmatrix} \alpha_m \\ l_m \end{bmatrix}^{BF} \tag{2.55}$$

After extracting each feature and their uncertainties, a covariance matrix for each line is calculated based on the uncertainty of each point belonging to the line feature:

$$\boldsymbol{P}_m = \begin{bmatrix} \boldsymbol{\sigma}_{\alpha\alpha} & \boldsymbol{\sigma}_{\alpha l} \\ \boldsymbol{\sigma}_{l\alpha} & \boldsymbol{\sigma}_{ll} \end{bmatrix} \tag{2.56}$$

3. Measurement prediction:

Using the predicted pose from 1) and the given the map $\mathcal{M}$, multiple predicted feature observations are generated, in the BF, which correspond to the features that the hexacopter should see if it was on the predicted pose. These features, existing on the map in the EF, are transformed to the BF, using h_i instead of (2.1) because the lines are in polar coordinates.

$$\begin{aligned} \boldsymbol{z}_n(i) = \begin{bmatrix} \alpha_{i,n} \\ l_{i,n} \end{bmatrix}^{BF} &= h(\boldsymbol{z}_{i,n}, \hat{\boldsymbol{p}}^-(i|i-1)) \\ &= \begin{bmatrix} {}^{EF}\alpha_{t,i} - {}^{EF}\,\hat{\psi}_i \\ {}^{EF}r_{i,n} - ({}^{EF}\hat{x}_i^- \cos({}^{EF}\alpha_{i,n}) + {}^{EF}\,\hat{y}_i^- \sin({}^{EF}\alpha_{i,n})) \end{bmatrix} \end{aligned} \tag{2.57}$$

The transformation's jacobian $\nabla \boldsymbol{h}_n$ is

$$\nabla \boldsymbol{h}_n = \begin{bmatrix} 0 & 0 & -1 \\ -cos({}^{EF}\alpha_{i,n}) & -sin({}^{EF}\alpha_{i,n}) & 0 \end{bmatrix} \tag{2.58}$$

4. Matching

Now that there is one set of observed features and another of predicted features, both in the BF, the matching step can start.

4.1. Identify all observations that match specific predicted features.

4.2. Calculate the *innovation*[16], $\boldsymbol{\nu}_{nm}$, for each measurement prediction, m, matched with an observed prediction, n.

4.3. Validate the pairs of features using a validation gate and reject the invalid ones. The matched pair will be the one with the lowest Mahalanobis[17] distance, and the validation gate used is the threshold value of 9.

[16] *Innovation* is a measure of the difference between predicted and observed measurements

[17] Mahalanobis distance is a unitless scale-invariant measure of distance that takes into consideration a distribution D. The distance is computed as the number of standard deviations from the point to the mean of D.

$$\begin{aligned} \boldsymbol{\nu}_{nm}(i)^T \cdot \boldsymbol{P}_{IN,nm}(i) \cdot \boldsymbol{\nu}_{nm}(i) &\leq g^2, \\ \text{with } g &= 3, \\ \boldsymbol{\nu}_{nm}(i) &= \left[\boldsymbol{z}_m(i) – h_n(z_n, p(i|i-1))\right], \\ \boldsymbol{P}_{IN,nm}(i) &= \nabla \boldsymbol{h}_n \cdot \boldsymbol{P}_p(i|i-1) \cdot \nabla \boldsymbol{h}_n^T + \boldsymbol{P}_{R,n}(i), \end{aligned} \tag{2.59}$$

where $\boldsymbol{P}_{R,n}$ is the covariance matrix associated to the measurement error of α_n and r_n.

5. Estimation

The last step is to compute the best estimate of the pose $\hat{p}^+(i|i)$, using the information gathered at time i.

5.1. Stack the validated observations $\boldsymbol{z}_m(i)$ in a vector $\boldsymbol{z}(i)$, along with the associated *innovations* $\boldsymbol{\nu}(i)$.

5.2. The measurement Jacobian for each validated measurement, $\nabla \boldsymbol{h}_n$, is put in a vector $\nabla \boldsymbol{h}$, with the measurement error noise $\boldsymbol{P}_R = diag(\boldsymbol{P}_{R,n})$.

5.3. Get the composite *innovation* covariance $\boldsymbol{P}_{IN}(i)$ from (2.59).

5.4. Compute the Update Step of the Kalman Filter in the following way [12]:

$$\boldsymbol{K}(i) = \boldsymbol{P}_p(i|i-1) \cdot \nabla \boldsymbol{h}^T \cdot \boldsymbol{P}_{IN}^{-1}(i). \tag{2.60}$$

And, ultimately, the current estimate for pose $\hat{p}_i^+ = \hat{p}(i|i) = \begin{bmatrix} \hat{x}_i & \hat{y}_i & \hat{\psi}_i \end{bmatrix}^T$ in the map coordinate system EF is computed, along with the associated covariance P_p.

$$\hat{\boldsymbol{p}}_i^+ = \hat{\boldsymbol{p}}(i|i) = \hat{\boldsymbol{p}}(i|i-1) + \boldsymbol{K}(i) \cdot \boldsymbol{\nu}(i) \tag{2.61}$$

$$\boldsymbol{P}_p(i|i) = \boldsymbol{P}_p(i|i-1) – \boldsymbol{K}(i) \cdot \boldsymbol{P}_{IN}(i) \cdot \boldsymbol{K}(i)^T \tag{2.62}$$

5.5. The transformation $\Delta\hat{\boldsymbol{p}}_i^+$ is computed by (2.63), using the Kalman Filter's pose estimate $\hat{\boldsymbol{p}}_i^+$. This transformation consists on a displacement between the pose estimate $\hat{\boldsymbol{p}}_i^+$ computed in this step (step 5) and the predicted pose $\hat{\boldsymbol{p}}_i^-$ computed in step 1. Both the displacement and the predicted pose are in odometry coordinates (OF), while the pose estimate is in map coordinates (EF), consequently it is converted to the OF in (2.63). This displacement is computed relative to the time of the LRF scan.

$$\Delta\hat{\boldsymbol{p}}_i^+ = \hat{\boldsymbol{p}}_i^- - \begin{bmatrix} \cos(\Delta\hat{\psi}^+) & -\sin(\Delta\hat{\psi}^+) & 0 \\ \sin(\Delta\hat{\psi}^+) & \cos(\Delta\hat{\psi}^+) & 0 \\ 0 & 0 & 1 \end{bmatrix} \hat{\boldsymbol{p}}_i\,, \quad \text{with } \Delta\hat{\psi}^+ = \hat{\psi}_i^- - \hat{\psi}i^+ \tag{2.63}$$

If $\Delta\hat{\psi}^+$=0, the rotation matrix in (2.63) is the identity matrix. This represents the ideal case, where the predicted orientation $\hat{\psi}_i^-$ is very accurate (there is no prediction error in orientation), consequently the odometry frame (OF) has the same orientation as the world frame (EF).

The goal of having this transformation $\Delta\hat{\boldsymbol{p}}_i^+$ is to have the measurement error of the 2-D pose of the robot at the time of the LRF scan. The alternative would be to have the absolute pose

estimate computed in Step 5.4. It is preferable to have the measurement error because, if the only information available is an absolute pose and this value is delayed, then it is impossible to compute a correct measurement error of that delayed pose. The output of this algorithm will be used as a measurement input on another Kalman Filter, which will be explained in Section 3.3. The latter Kalman Filter can be designed to accept either a measurement error or an absolute measurement; if an absolute measurement is sent, it will eventually be converted into a measurement error, which will certainly be less accurate than the displacement computed in Step 5.5.

The output of this algorithm can be either the absolute pose $\hat{\boldsymbol{p}}_i^+$ computed in Step 5.4 or the displacement computed $\hat{\boldsymbol{p}}_i^+$ in Step 5.5.

2.8 Cascaded Extended Kalman Filter

In this thesis, the approach taken to perform localization in the x and y dimensions involves a cascaded EKF. Figure 2.6 shows how the two EKF approaches described in Sections 2.6 and 2.7 can be combined, which are named PEKF and BEKF, respectively. The implementation of these algorithms will be presented in Section 3.3. The reasons behind this implementation choice are also presented in the latter Section. The equations in grey are the ones excluded from the final implementation; this will be discussed in Section 3.3. Even though the EKF presented in Section 2.6 was applied to the altitude localization, the formulas extracted from that Section that are presented in Figure 2.6 are general and can be applied to estimate variables in x and y dimensions.

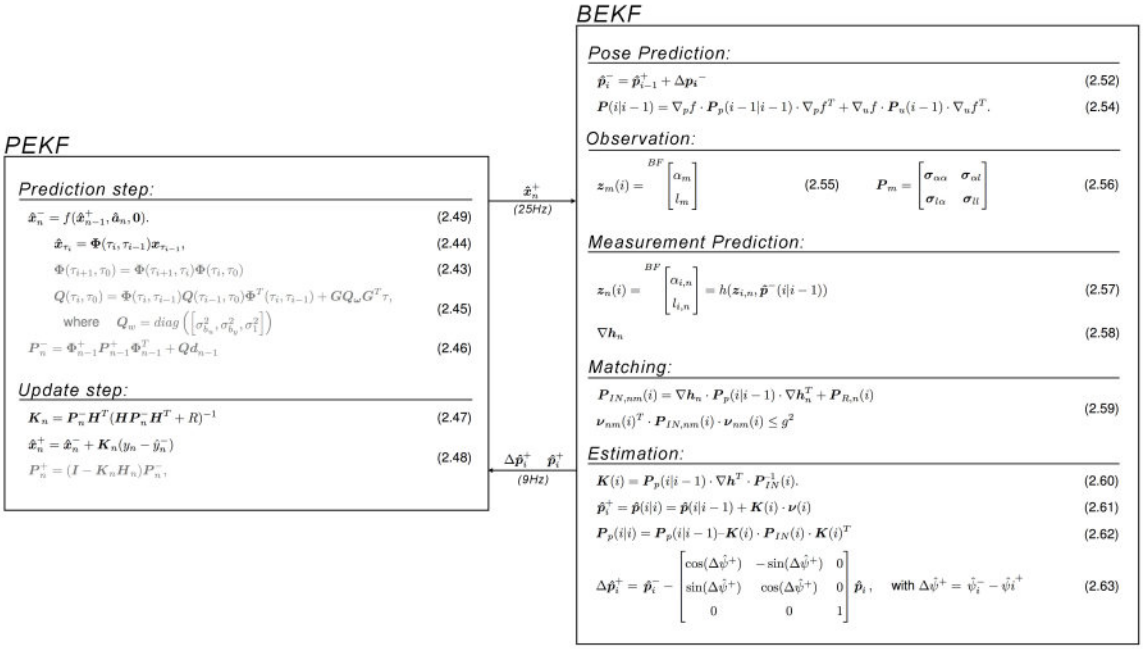


Figure 2.6: Here is presented the mathematical description of the combination of the two localization algorithms in a cascaded approach. The PEKF represents the first algorithm studied, while the the BEKF represents the feature-based localization algorithm.

Chapter 3

Methodology

This Chapter addresses the approaches implemented in order to achieve the proposed goals of our project. The main focus is on the localization algorithms, but other points were tackled in order to have a functional system control loop that does not require human feedback. In Section 3.1, the attitude controllers will be addressed, the communication software developed is presented and the simulators developed are discussed, especially their functionality, assumptions and limitations. In the latter Section, it is also presented the changes performed on the attitude estimation. Furthermore, the simulation setup for the localization algorithms is explained, where the assumptions and limitations are stated, and the map setup that serves as a testbed for our experimental work is also presented.

The instruments and implementation of the altitude localization are addressed in Section 3.2, following the algorithm presented in Section 2.6, but specifying the design choices. The procedures of the implementation are explained, followed by a more specific discussion of the implementation details.

In Section 3.3, the localization in x and y dimensions is tackled. The design choice of combining the algorithms from Sections 2.6 and 2.7 is discussed, followed by the description of the procedures to implement them together. Finally, specific details of the implementation are discussed and the algorithm's limitations are stated, along with implementation aspects that can be improved in the future.

3.1 Preliminary Work

3.1.1 Attitude Controller

The attitude controller used is a cascade controller, with a PD in the inner loop and a P on the outer loop. This controller's behavior is similar to a PD angular position controller. In this Section, the architecture and tuning of this controller is discussed, both theoretically and how it was achieved in practice.

The angular position loop must have sufficient bandwidth to track the angle reference and the angular velocity loop must have sufficient bandwidth to track the velocity reference. Nevertheless, there are some basic guidelines that our cascaded system must follow. From control theory it is known that in a multi-loop system the slowest response is always the bottleneck. Thus, in our system, the velocity loop response needs to be around one order of magnitude faster then the angle loop around it.

Model of the controller

In order to study the controller and simulate its response, a Simulink model of the hexacopter was constructed, using a quadrotor model developed by Brogaard [15] as a baseline. The Inertia model constructed in Section 2.3.1 was used in this Simulink model. The model of the hexacopter system linearized around the hover point used is shown in (3.1), where the body and the propeller gyroscopic effects are neglected. Since the linearization of a system locally determines the stability of the full system, this class of controllers is guaranteed to be locally stable [42]. The original model [15] was transformed in order to match the hexacopter model described in Section 2.3.2 and the controllers used in the real system. The controllers are designed in Simulink in such a way that they best match the C++ code implementation used on the real system. The motor model corresponds to the real system, which was studied by Brogaard [15], but the gyroscope model does not. The estimation of angular velocity and position that occurs in the real system, where an attitude EKF is used, was not replicated. Instead, a gyroscope model is used, which returns delayed measurements of angular velocity. The measurement signal is integrated over time to get angular position. The controller has access to the true state, with a certain delay and integration error, since discrete samples are integrated at 250Hz and not the continuous state. The Simulink models developed are presented in Appendix B.

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} \dfrac{l \sum_{i=1}^{6} T_i \sin(\dfrac{i-1}{6})}{\mathcal{I}_{xx}} \\ \dfrac{l \sum_{i=1}^{6} T_i \cos(\dfrac{i-1}{6})}{\mathcal{I}_{yy}} \end{bmatrix} \tag{3.1}$$

The inner loop of our cascade controller analysis is firstly presented, followed by the analysis of the outer loop angle controller, for the roll and pitch angles.

Roll

- **Inner loop (velocity loop)**

Figure 3.1 shows the Bode diagram of the open inner loop using a PD controller. K_P^{vel}=0.2 and K_D^{vel}=1.8 were chosen as the ideal gain values because they give a Phase Margin around 50 degrees while keeping the Bandwidth over 14rad/s and the Gain Margin at a reasonable value (14.4dB).

The bandwidth of the closed loop system can be increased by raising the P gain, but this will decrease the Phase Margin and Gain Margin. Increasing the P gain will also result in an increase of the oscillation of the velocity response shown in Figure 3.2. By increasing the D gain, it is possible to gain back some Phase Margin, while sacrificing bandwidth of the system.

The response of the closed loop system is depicted in Figure 3.2 and the response of the same controller to a position reference (adding a position feedback loop) is shown in Figure 3.2. A small overshoot is desired on the velocity response, since it will make the position tracking faster.

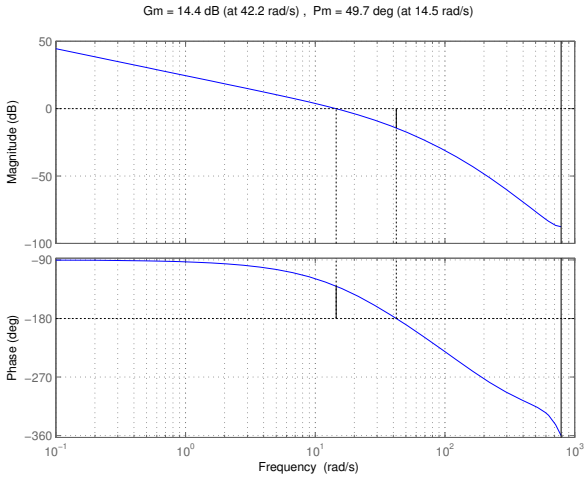


Figure 3.1: Bode diagram of Roll angular velocity open loop control.

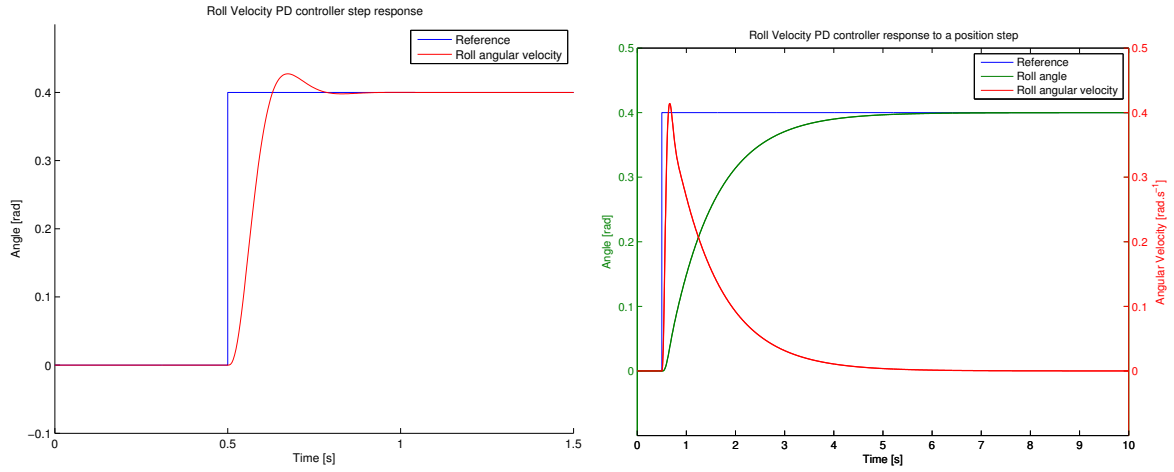


Figure 3.2: Figure (a) shows the response to a Roll angular velocity step using the Roll angular velocity PD controller, while figure (b) shows the response to a Roll angle step, using the same PD angular velocity controller and closing the position feedback loop.

• **Outer loop (position loop)**

A Proportional component is added around the existing controller to tune the response speed to a position reference. It is selected K_P^{pos}=6 for the position P gain. Our final controller does not have a large bandwidth (6rad/s), but has a large stability margin, with a 66.5 degree Phase Margin and 11.6dB Gain Margin. The response has a 10%-90% rising time of 181ms.

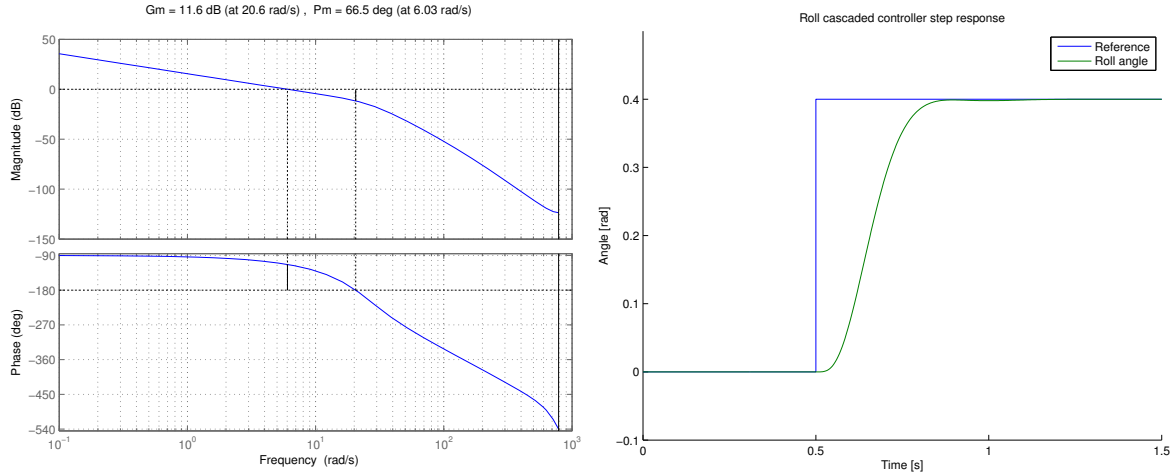


Figure 3.3: Figure (a) shows the Bode diagram of the final cascaded controller, while figure (b) shows its response to a Roll angle step.

Pitch

• **Inner loop (velocity loop)**

Figure 3.4 shows the Bode diagram of the open inner loop using a PD controller. The same gains as the ones used on the roll velocity controller, K_P^{vel}=0.2 and K_D^{vel}=1.8, were chosen. Even though the hexacopter is not symmetric, the roll and pitch dynamics are very similar. Choosing the same gains will not lead to the same response, but the result will be identical. The system has a 60.7 degree Phase Margin, Bandwidth of 15 rad/s and the Gain Margin at a reasonable value (14.5 dB).

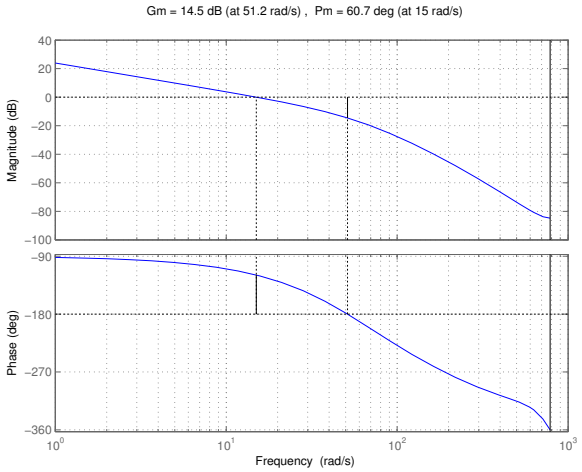


Figure 3.4: Bode diagram of Pitch angular velocity open loop control.

The response of the closed loop system is depicted in Figure 3.5 and the response of the same

controller to a position reference is shown in Figure 3.5, by adding a position feedback loop. A small overshoot is desired on the velocity response, since it will make the position tracking faster.

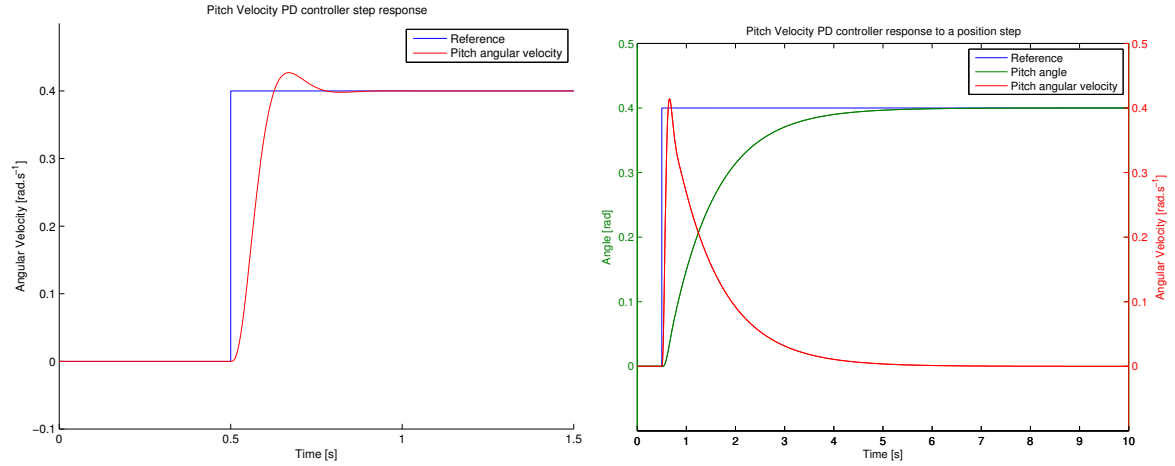
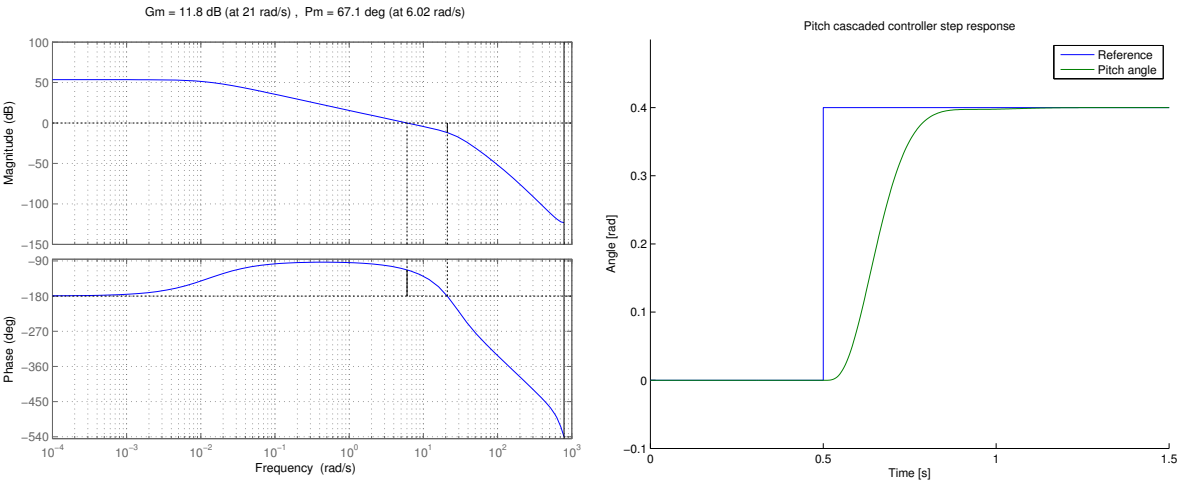


Figure 3.5: Figure (a) shows the response to an angular velocity step using the Pitch angular velocity PD controller, while figure (b) shows the response to an angle step, using the same PD angular velocity controller and closing the position feedback loop.

- **Outer loop (position loop)**

A Proportional component is added around the existing controller to tune the response speed to a position reference. It is selected K_P^{pos}=6 for the position P gain. Our final controller has a 6rad/s bandwidth, a Phase Margin of 67.1 degrees and an 11.8dB Gain Margin. The response has a 10%-90% rising time of 183ms.



(a) (b)

Figure 3.6: Figure (a) shows the Bode diagram of the final cascaded controller, while figure (b) shows its response to a Pitch angle step.

Tuning the real attitude controller

The roll and pitch controllers designed and described above are used as a starting point to calibrate the real controllers. The model used is only a linear approximation of the real system and the state feedback

is not accurately reproduced on the Simulink. However, it gives a good notion of adequate values for the controller parameters.

A heuristic method was adopted to tune the attitude controller, as recommended on the PX4 autopilot's webpage[1].

A PD roll/pitch rate controller is preferred over a PID because the integrator component will not only slow down the response, but also accentuate the errors that are intrinsic to the sensors, such as small offsets. There is always a small offset on sensors, such as on the accelerometer, that result in a slight offset on the angle estimation, regardless of how carefully the calibration of the sensors is done.

The final values selected are the same for pitch and roll controllers, due to their identical behavior:

$$K_P^{pos} = 6.5$$
$$K_P^{vel} = 0.11$$
$$K_D^{vel} = 0.01$$
$$K_I^{vel} = 0$$

One can notice the disparity on the D gain values between the simulated, K_D^{vel}=1.8, and used, K_D^{vel}=0.01. On the real system, the velocity's P component was tuned to have the biggest value that would not lead to oscillations, K_P^{vel}=0.11. This value proved to be smaller than the simulated, K_P^{vel}=0.2, which meant that the bandwidth would not be as high as the one expected from simulations. For this reason, there was no benefit on having a big D gain value, since that would decrease the bandwidth of the controller.

3.1.2 Communication Software: MavlinkComm

The RHD plugin MavlinkComm manages both the RHD variable database and the communication with the Pixhawk. The details specific to this program can be found in Appendix A.4.

The Pixhawk uses the MAVLINK message marshaling[2] protocol. This communication protocol is designed to be lightweight, with only 8 bytes overhead per packet, and has an inbuilt packet-drop detection. For more details on MAVLINK, refer to Meier et al. [10], [11].

Firstly, a C++ program was built to communicate with the Pixhawk from an offboard computer, using the MAVLINK protocol. This program was built on top of an already existing class *Autopilot_Interface*[3], which already had some of the features needed to communicate with the Pixhawk: reading and decoding incoming MAVLINK messages and sending two types of messages.

RHD is in C language to ensure a lower level control of the system and avoid memory leaks, and, as a consequence RHD, plugins must also be C programs. Consequently, MavlinkComm is created as a C plugin that is compatible to interact with C++ applications by using wrapper[4] functions. This allows the

[1] `https://pixhawk.org/users/multirotor_pid_tuning`

[2] Marshalling is the procedure of transforming a message to a data format suitable for transmission.

[3] Original class can be found in the git repository `https://github.com/mavlink/c_uart_interface_example`

[4] A wrapper function enables calling member functions or changing the values of attributes of a C++ class object, from a C environment.

interaction with the *Autopilot_Interface* class.

MavlinkComm creates and initializes all the variables in the RHD database, hence all the necessary variables in the communication Pixhawk↔RHD↔MRC are created. The final part of the initialization is establishing communication with the Pixhawk. RHD will only be fully running when this communication is secured. For this to be possible, it is necessary that the Pixhawk is running and sending MAVLINK messages regarding attitude and local position information, otherwise the program will wait until it receives these MAVLINK messages. After that, other clients can establish a connection to RHD: RHDTEST[5], ULMS and MRC, keeping in mind that MRC should only be started after ULMS is running, in order for it to be recognized by the ULMS server.

MavlinkComm was designed with a set of features that enables the on-board computer (BeagleBone) to be part of the hexacopter's control loop system. These features are:

- establish a MAVLINK connection.
- receive information regarding the hexacopter's attitude and local position, Pixhawk system status and other kinds of information, if needed.
- send position and attitude error estimates.
- send position and attitude absolute estimates.
- send command to enable/disable Pixhawk's flying modes.
- send attitude and position setpoints.

The communication is achieved using a serial (UART-UART) connection, with 57600 baudrate.

3.1.3 Algorithm Simulators

Before testing the localization algorithms on the real system, simulations in Matlab were conducted to evaluate if the implementation of these algorithms is correct and its performance. The estimation algorithms used in this project are sensor-based, hence the accuracy of the simulator in representing reality depends greatly on how accurate the model of the simulated sensors is. Two simulators are created in Matlab to study the implementations of the algorithm presented in Section 2.6.2: one for the altitude estimation, another for the estimation in x and y dimensions. The simulators are decoupled because the state variables in the z axis can be estimated independently from the state variables in the x and y dimensions.

The sensors are modeled as described in Section 2.6.2, where the real measurements were mimicked as best as possible. The noise specifications described in Section 2.6.2 were unfavorably aggravated to make sure that the simulator does not present better conditions than reality. The resolution of each sensor is taken into account, as well as the delay inherent to each sensor, since it is impossible to have instantaneous measurements. The main differences from the real sensor measurements are:

- Sonar: a fixed attitude is always considered, with zero pitch and roll angles, and obstacle detection noise is discarded, because it was decided that including random obstacle detection as noise did

[5]RHDTEST is an application that enables to vizualize the values in the RHD database in real-time.

not make sense. All other sources of noise are considered: white Gaussian noise and random spikes. The spike noise follows a random uniform distribution with an average of 1 spike per 80 measurements, which is far greater than what was observed experimentally.

- Barometer: temporary short-term effects arising from changes on the environment, such as opening a window, are neglected. The bias is modeled as a linear function of time, where the slope is randomly chosen from an interval of numbers, and is affected by a random walk process plus a fixed offset randomly chosen.

- Accelerometer: the scaling factor of the accelerometer is ignored. Since it is usually very close to 0, it is considered irrelevant.

Neglecting the attitude of the hexacopter is not very relevant, in altitude estimation, for two reasons: 1) attitude does not have a big effect on the measurements, since they are compensated in the accelerometer, rejected in the sonar (for big attitude values) and no effect on the barometer; 2) attitude cannot be high for more than a very short period ($<$0.5s) in indoor environments (if the control of the vehicle is done correctly), since that would mean that the hexacopter has high acceleration values continuously and that is not a scenario in an indoor environment.

The LRF measurements are not modeled because they are never used directly by any Kalman Filter. As explained in Section 2.7, features are extracted from the LRF measurements using RANSAC. The output of the localization algorithm described in Section 2.7 is modeled, since it is used as an input of the algorithm described in Section 3.3. Section 2.8 shows that the pose estimate is used as a low-frequency sensor input, *i.e.*, as a measurement. These position estimates, produced by the localization algorithm, are modeled as:

$$y_k = p_{k-2} + \eta(k) \tag{3.2}$$

where $\eta(k)$ is a Gaussian white noise process with Power Spectral Density (PSD) equal to σ^2=0.13^2 $\frac{m^2}{Hz}$. No correct way of modeling the estimate of an Extended Kalman Filter was found. The measurement does not have the same value of position, so it was decided to model this estimation error as a white Gaussian noise process. This is not a correct procedure, but it was the most convenient. The error of the estimation has approximately zero mean experimentally, when the robot is still. The experimental variance of this error is $\sigma^2_{experimental}$=0.0686^2 $\frac{m^2}{Hz}$, which is approximately half the one considered in simulation, in order to represent a worse scenario. The measurement has a 0.2s delay and the velocity that is computed from these measurements has a 0.35s delay. These delay values correspond to the reality, which will be explained in Section 3.3. In simulation, it is assumed that the localization algorithm is always able to extract a pose estimate from laser measurements, but that is not guaranteed to be true in reality.

Attitude has a big effect on the measurements of the LRF. Neglecting the attitude effect in the model of the laser-based position estimates leads to better estimates than reality. Because of neglecting the attitude and not being able to accurately model the estimation error, the variance of the noise distribution in our model was increased to σ^2=0.13^2 $\frac{m^2}{Hz}$, while the experimental one is $\sigma^2_{experimental}$=0.0686^2 $\frac{m^2}{Hz}$,

measured when the robot has the same pose.

In simulation, there is access to the ground truth state of the system at every instant, unlike on the real system. This provides the ability to test the performance of the algorithms in terms of accuracy and delays, and also to tune parameters in the algorithm.

One can observe that the sonar velocity errors are approximately 10 times bigger than the sonar measurement errors because the differentiation of the measurement is done with a 0.1 time interval (f_{sonar}=10Hz). The velocity estimation errors do not increase tenfold, which is a good indicator that the EKF is performing a good estimation.

One aspect to be considered is that the noise of the sonar in the simulator was increased to σ^2 =0.06^2, instead of the real value σ^2=0.01^2, in order to represent a worst case scenario.

3.1.4 Attitude Estimation

The attitude estimator running on the Pixhawk was studied and the yaw angle ψ estimation was slightly modified. Originally, it uses a gyroscope to measure angular velocity and magnetometer to measure the strength of the Earth's magnetic field, which is converted into a yaw angle. The yaw angle and angular velocity are estimated from the combination of these measurements in a Kalman Filter. In indoor environments, magnetic fields change abruptly due to objects in the surroundings and especially due to materials in the building's structure. This makes the magnetometer a nonviable choice to provide yaw angle measurements. To correct this handicap, the yaw angle is estimated using the localization algorithm presented in Section 2.7, by measuring the orientation of the robot in relation to the environment map. This yaw estimate is fed to the Pixhawk's attitude Kalman Filter, which was changed to use these estimates instead of magnetometer information.

The Kalman Filter can track fast yaw angular motion because of the gyroscope measurements. These measurements are integrated in time and allow to accurately track fast changes of the yaw angle. Using only the gyroscope, the angle estimates would drift, but they are corrected with very accurate angle measurements based on the laser information; this way, the yaw angle estimates have a small bounded angle error.

If the laser-based yaw estimates computed by the localization algorithm are not available, magnetometer measurements will be used instead, using the original approach.

3.1.5 Map Setup

The map used as a testbed in all the localization experiments is a corner with two walls with 3m width and height close to 2.2m. These walls do not touch the ceiling, hence it is common to loose track of the walls for 2 or 3 laser range measurements if the hexacopter is very tilted at some point; this depends on the altitude of the hexacopter and its distance to the walls. More walls can be added to the map and this will improve the localization algorithm. The reason for choosing a corner as a testbed is to test how the system behaves when using the minimum features required for the localization algorithm to work; this is the worst case scenario. There are other worst case scenarios that are not considered, such as having

a map with only very small features. This case is not considered, because in a structured environment, which is an assumption of this project, it is likely that the environment can be described by relatively long lines.

3.2 Study 1: Altitude Estimation

3.2.1 Introduction

In the Section 2.6, a formal description of the methods used to estimate altitude variables was presented. The Kalman Filter designed for this purpose is based on these methods, and its implementation is handled in this Section. The goal of the implemented KF is the same as the one presented on Section 2.6, *i.e.*, to determine the position and velocity of the hexacopter along the vertical axis. The complexity of the design is relaxed in order to get a more lightweight KF, with the requisite of achieving good results with the less complex approach. The reader is advised to read Section 2.6 in order to fully understand this Section.

3.2.2 Objectives

The implementation of this estimator aims at being a version of Section 2.6 as inexpensive as possible, where good results are obtained. Pre-processing information from the ultrasonic sensor is proposed, in order to get a signal that only contains white Gaussian noise. Having good measurements from this sensor will reinforce the possibility of still achieving good estimation results when lowering the computation demands.

The objectives for this estimator are:

- providing state estimates at a 250Hz frequency,
- estimating altitude position with 4cm accuracy,
- estimating altitude velocity with 0.1m.s^{-1} accuracy,
- estimating x and y velocity with a delay lower than 0.1s,
- withstand sonar outages with bounded position error,
- withstand barometer outages with bounded position and velocity error.

3.2.3 Sensor model

As explained on Section 2.6, the sensors at our disposal are accelerometer, barometer and ultrasonic sensor. A simpler model than presented in (2.22) is used for the case of the accelerometer:

$$u = a - b_u - \eta_1. \tag{3.3}$$

The accelerometer's scale factor is neglected. The reason for this decision lies on the fact that the scale factor might not be estimated precisely and a bad estimate can cause more harm than not estimating it at

all. The concept is rather simple: the scale factor is the proportionality constant between the measured acceleration and the actual acceleration given from the sensor. Apart from the difficulty in estimating it precisely, the scale factor for a positive acceleration may be different than the scale factor for a negative acceleration [49]. The difference between the two scale factors arises from the torque rebalance loop in the accelerometer. Therefore, there are generally two separate scale factors for a given accelerometer. The main problem with neglecting the scale factor is that the bias estimation will often suffer. This arises from the fact that an acceleration measurement that is bigger or smaller than reality will lead to error when integrating once (for velocity) or twice (for position). This error will be misinterpreted by our Kalman Filter as the presence of an offset on the accelerometer measurement. Despite the possible incorrect estimate resulting from this case, the bias estimate will always converge to the correct value when the acceleration measurements have small values (and the velocity of the vehicle is low). The other variables in this model are explained in Section 2.6.2.

The barometer model is written on (2.24) and explained on Section 2.6.2. A Low-Pass Filter is applied to the barometer's signal in order to reduce the quantization noise, and this will introduce a delay to this signal. This delay should not be considered relevant, because the role of the barometer will be to bound the position error during periods at which sonar measurements are not available, *i.e.*, the barometer is used as a backup in a faulty situation. The role of the barometer will be explained in more detail in Section 3.2.5.

The sonar model is a slightly simplified version of (2.26) and is now described as

$$y_{sm} = p_m + \eta_3(m) \tag{3.4}$$

The model does not depend directly on the roll and pitch angles, which are considered to be zero at all times. This simplification does not have any practical influence, and the reason will now be explained. Since measurements taken when the sonar is tilted more than 10 degrees will be ignored, as explained on Section 2.6.2, the maximum accepted roll and/or pitch angles are 10 degrees. The difference between the measurement y_{sm} and the real altitude p_m is negligible for measurements under 2 meters (maximum error under 6.3cm). Because the sonar's beam shape is approximately a cone, tilting the sonar by a small angle (such as 10 degrees) will actually give the correct height measurement p_m. The sonar will detect the nearest point to the robot on the ground, inside the beam cone, instead of the point in the ground that the sonar is directly pointing at. As a result, this simplification does not have a practical negative impact on the height measurements. The sensor model designed is only accurate if the measurements are subjected to validation gates, which reject noisy measurements that cannot be modeled as white Gaussian noise. The different kinds of noise components were discussed in Section 2.6.2. The Figure 3.7 illustrates the validation gates that will reject noisy measurements. It is important to notice that when a measurement that differs more than 20cm from the most recent valid measurement is received, this can represent one out of two cases: 1) the measurement is a spike originating from motor noise or an object (such as a wall) is detected, thus the measurement should be ignored; 2) the ground is at a new altitude, e.g. the drone is over a table. This measurement will be classified as invalid, but an auxiliary variable called *accepted change* will grow with each one of these invalid measurements. If in

reality there is a new ground level, invalid measurements will be received continuously, making the variable *accepted change* grow, and the validation gate will end up accepting this new altitude measurement as a new ground level. A practical example of the latter scenario is shown in Figure 4.7 and discussed in Section 4.3.1. It is important to notice that the sonar measurements will only be used to correct the velocity state if two consecutive measurements are considered valid. The velocity estimate provided by the sonar is computed by differentiating the sonar signal.

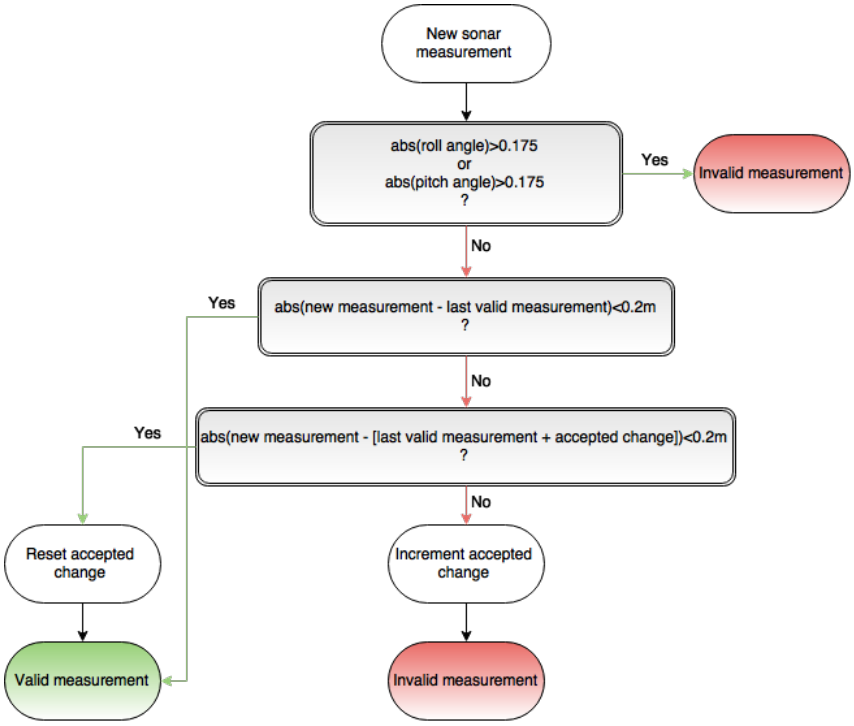


Figure 3.7: Sonar measurement validation gate.

3.2.4 Implementation

Our kinematic based system has state $\boldsymbol{x} = \begin{bmatrix} p, v, b_u, b_y \end{bmatrix}^T$. The navigation mechanization equations are

$$\dot{\hat{\boldsymbol{x}}} = f(\hat{\boldsymbol{x}}, \hat{a}, \boldsymbol{\omega}) = \begin{bmatrix} \hat{v} \\ \hat{a} \\ 0 \\ -\lambda_y \hat{b}_y \end{bmatrix} = \begin{bmatrix} x_2 \\ u + \hat{x}_3 \\ 0 \\ -\lambda_y \hat{x}_4 \end{bmatrix}. \tag{3.5}$$

To achieve a discrete model sampled at instants $t_k = kT$ of the form $\delta \boldsymbol{x}_{k+1} = \Phi_k \delta \boldsymbol{x}_k + \boldsymbol{w}_k$, (3.5) is integrated numerically. Admitting that the acceleration is constant between integration time instants of length $\tau = \frac{1}{f_1}$, one gets the Transition State Model in (3.6) and Control Input Matrix in (3.7).

$$\boldsymbol{\Phi}(\tau_i, \tau_{i-1}) = \begin{bmatrix} 1 & \tau & \frac{1}{2}\tau^2 & 0 \\ 0 & 1 & \tau & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{-\lambda_y \tau} \end{bmatrix} \tag{3.6}$$

$$\boldsymbol{B} = \left[\tfrac{1}{2}\tau^2, \tau, 0\right]^T, \tag{3.7}$$

where our input is considered as being the accelerometer measurement in the z direction. From (3.3), one gets $a = u + b_u$, which results in having the same terms multiplying by b_u in matrix $\boldsymbol{\Phi}$ (column 3, rows 1 and 2) and by u in matrix $\boldsymbol{B}$ (rows 1 and 2).

The observation matrix can be written in matrix form, for the barometer, as $\boldsymbol{H}_1$ and, for the sonar, as $\boldsymbol{H}_2$, as presented in (3.8).

$$\boldsymbol{H}_1 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{H}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.8}$$

Hence the measurement correction, when correcting with barometer information, is shown in (3.9) and, when correcting with sonar information, is shown in (3.10).

$$\boldsymbol{z}_n = \begin{bmatrix} y_{bk} \\ 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \hat{\boldsymbol{x}}_k^- \tag{3.9}$$

$$\boldsymbol{z}_n = \begin{bmatrix} y_{sm} \\ \dot{y}_{sm} \\ y_{sm} \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \hat{\boldsymbol{x}}_m^- \tag{3.10}$$

In (3.10), $\dot{y}_{sm}$ is used as being the velocity measurement given by the ultrasonic sensor. In fact, it is computed as the derivative of the sonar measurement for consecutive valid measurements. In the approach taken on (3.10), b_u correction can be intuitively thought as being how wrong the state propagation is compared to the sonar measurement. You can notice that b_y is not being corrected with the measurements. The obvious approach would be to correct b_y estimate from the divergence of the position estimate and the barometer measurement. This way proves to be unreliable if the barometer is the only sensor correcting the position estimate. Another approach was taken, but it cannot be expressed in this mathematical formulation; in this implementation, b_y is corrected when a valid sonar measurement is received by computing

$$z_{b_y} = (y_{bk} - \hat{b}_y^-) - y_{sm}, \tag{3.11}$$

where y_{bk} is the latest barometer measurement, $\hat{b}_y^-$ is the barometer bias estimate prior to update and y_{sm} is the new sonar measurement. This equation is equivalent to saying that the divergence of measurements between sonar and barometer lies on an incorrect b_y estimate. This is not true at every epoch m, because of the noise in the sonar or barometer measurements. Since this correction will be done

with a low weight, only the slow time-varying part of this divergence is used to correct b_y. This low weight acts as a Low-pass Filter, making this approach independent from sonar and barometer measurement noise, since the measurement noise has a frequency very high comparatively to the growing rate of b_y. Since it is the sonar that is mainly responsible for correcting b_y, the Kalman Gain associated with this correction will be treated as part of the sonar's Kalman Gain vector, which will be presented later in this Section in (3.16).

In order to keep the estimator computationally lightweight, a static Kalman Filter is used. Using fixed Kalman gains may lead to a worse solution, but certainly saves a lot of computation power that was to be spent on propagating the state covariance, since the computational load of aided navigation systems is typically dominated by the time update of the covariance matrix [45]. The chosen Kalman gains will be discussed in Section 3.2.5.

When a new position measurement from either sensor is received, at epoch n, the current estimate is corrected as follows

$$\begin{aligned} \hat{\boldsymbol{x}}_n^+ &= \hat{\boldsymbol{x}}_n^- + \boldsymbol{K}\boldsymbol{z}_n, \\ \text{with } \boldsymbol{z}_n &= \boldsymbol{y}_n - \hat{\boldsymbol{y}}_n^- \end{aligned} \tag{3.12}$$

where $\boldsymbol{y}_n^-$ can be either $\boldsymbol{y}_k^- = \boldsymbol{H}_1\hat{\boldsymbol{x}}_k^-$ or $\boldsymbol{y}_m^- = \boldsymbol{H}_2\hat{\boldsymbol{x}}_m^-$, in case the position is corrected with a barometer measurement or the whole state vector is corrected with sonar information, respectively. In the case where the sonar measurement cannot be used to give a proper velocity estimate, the second row of $\boldsymbol{H}_2$ is substituted by zeros.

3.2.5 Implementation Discussion

The details of the implementation presented will now be discussed. The approach described in Section 3.2 was implemented and showed good results. Even though the results looked promising (position errors smaller than 3cm) when holding the hexacopter by hand and simulating a flight with the motors turned off, the result was not as satisfactory when flying the hexacopter. When giving altitude position setpoints and/or altitude velocity setpoints, the result showed small perturbations, not on the estimated state, but on the controller outputs. The estimator was causing these perturbations. Correcting the state estimates when a new measurement arrives, makes the state change abruptly from the iteration before the correction to the iteration after. These small discontinuities were being amplified on the controller, causing the system to react not as smoothly as it should. The solution to this problem was to spread out the state correction along the propagation stage. To reach this, the measurement correction, $\boldsymbol{z}_n$, that should be applied (in the same way as (3.12)) is firstly computed, then that correction is divided into smaller fractions, $\Delta\boldsymbol{z}_n$, and then these smaller corrections are applied in the future iterations. Notice that the sum of smaller corrections equals the total correction, $\boldsymbol{z}_n$, that should be applied. The measurement correction, $\boldsymbol{z}_n$, is computed in every epoch n, *i.e.*, every time there is a new position measurement. The difference is that a smaller correction, $\Delta\boldsymbol{z}_n$, is now applied in every state propagation iteration, instead of applying $\boldsymbol{z}_n$ only when a new position measurement is received. Instead of correcting with (3.12) at a

frequency f_2 (barometer) or f_3 (sonar), it is used

$$\hat{\boldsymbol{x}}_n^+ = \hat{\boldsymbol{x}}_n^- + \boldsymbol{K}(\boldsymbol{y}_n - \hat{\boldsymbol{y}}_n^-)\tau \tag{3.13}$$

with $\tau = \frac{1}{f_1}$, in order to spread the correction in a discrete manner at the higher frequency f_1. But our goal is to introduce the same correction over the interval of time between sensor updates $t \in [n_i, n_{i+1}[$, not along 1 second. Hence, the sum of these smaller corrections over the interval $t \in [n_i, n_{i+1}[$ should have the same value as the correction done with the previous approach, (3.12). The scaling factor α that transforms approach 3.12 to 3.13 can be calculated:

$$\boldsymbol{z}_n = \alpha \sum_{i=0}^{f_1/f_n-1} \Delta\boldsymbol{z}_n \Leftrightarrow \boldsymbol{z}_n = \alpha \sum_{i=0}^{f_1/f_n-1} \boldsymbol{z}_n\tau \Leftrightarrow \boldsymbol{z}_n = \alpha\boldsymbol{z}_n\tau\frac{f_1}{f_n} \Leftrightarrow \alpha = \frac{f_n}{\tau f_1} \Leftrightarrow \alpha = f_n, \tag{3.14}$$

where f_n represents the sampling frequency of one of the position sensors and α represents the proportionality between approaches 3.12 and 3.13, taking into consideration the different frequencies at which they are computed. α is applied to the Kalman Gain vector, leading to a new Kalman Gain vector that relates to the original one as $\boldsymbol{K}_{new} = \alpha\boldsymbol{K}_{old}$, with $\alpha = f_2$ for the barometer case and $\alpha = f_3$ in the ultrasonic sensor case.

The barometer and sonar Kalman Gains, $\boldsymbol{K}_b$ and $\boldsymbol{K}_s$, were manually tuned taking into consideration the measurement noise and experimental data.

$$\boldsymbol{K}_b = \begin{bmatrix} 0.1 & 0 & 0 & 0 \end{bmatrix}^T, \quad \text{equivalent to the old } \boldsymbol{K}_{b,100} = \begin{bmatrix} 0.001 & 0 & 0 & 0 \end{bmatrix}^T \text{ at } f_2 \tag{3.15}$$

$$\boldsymbol{K}_s = \begin{bmatrix} 4 & 1.5 & 4 & 0.5 \end{bmatrix}^T, \quad \text{equivalent to the old } \boldsymbol{K}_{s,10} = \begin{bmatrix} 0.4 & 0.15 & 0.4 & 0.05 \end{bmatrix}^T \text{ at } f_3 \tag{3.16}$$

The barometer Kalman Gain vector is considerably lower comparing to the sonar. This is due to the characteristics of the barometer sensor explained on section 2.6: the signal is noisy, has low resolution and its error grows with the error of the b_y estimate. The Kalman Gain related to b_y is rather low but could be set even lower, because this signal varies slowly in time, so its variation should not vary abruptly with the corrections introduced by the sonar. A low Kalman Gain can be intuitively seen as a Low-Pass Filter on the correction applied to b_u, filtering high frequency noise that could come from sonar and barometer measurements.

Before starting the estimator, the initial state is initialized as

$$\boldsymbol{x}_0 = \begin{bmatrix} E[p(0)] & E[v(0)] & \mu_{b_u} & \mu_{b_y} \end{bmatrix}^T, \tag{3.17}$$

where $E[p(0)] = 0$, $E[v(0)] = 0$ and the biases μ_{b_u} and μ_{b_y} are initialized with the mean of the first 500 samples of the barometer and accelerometer measurements, respectively. This initialization takes 2 seconds and greatly reduces the initial error of the state estimate and, consequently, of the near future states. It is not possible to predetermine a good initialization of the biases because μ_{b_u} depends on unpredictable factors such as temperature and μ_{b_y} depends on the altitude AMSL of the ground the robot is initialized in, temperature (even though a compensation is performed), humidity and other issues that will affect the local indoor air pressure. It is safe to do the initialization this way because the filter

always starts when the hexacopter is turned on and, at that time, it is motionless on the ground.

Due to the characteristics of the barometer signal, this sensor was introducing more noise to the estimate than relevant information. It was decided to use only the barometer measurements on the estimator if the sonar is not valid for more than 0.3s, which is equivalent to 3 sonar sample periods with invalid measurements. The barometer corrections will be deactivated again when a new valid sonar measurement is received. The bias of the barometer signal is always being corrected in (3.13), using (3.11), so that the bias is always properly estimated at any time the barometer may be needed to estimate position. The role of the barometer is to bound the position error when the sonar is not available. It is not possible to have good altitude estimates indoors with the sensor used, so it is only used as a last resource as a fault tolerant approach. A signal with lower noise was generated by applying a Low-Pass Filter, introducing a delay of up to 0.2s, depending of course on the variation of the signal.

3.3 Study 2: XY Estimation

3.3.1 Introduction

A cascaded Extended Kalman Filter (CEKF) is used to estimate the position and velocity of the hexacopter in the x and y dimensions. To reach this goal, other state variables have to be estimated, such as the biases of the accelerometer sensor and the heading of the robot. Figure 3.8 shows a representation of the CEKF approach, which combines one EKF running on the Pixhawk, Pixhawk's EKF (PEKF), and another running on the Beaglebone, BeagleBone's EKF (BEKF). The first one, PEKF, is based on the model presented on Section 2.6, where the state propagation is done at a high frequency of 250Hz and the correction of the current state computed at a lower frequency of 9Hz. It provides dead-reckoning navigation for a short period of time if the BEKF is not running. The second, BEKF, uses the algorithm presented in Section 2.7; it provides low-frequency position estimates to the PEKF at 9Hz, and is responsible for using the LRF measurements to estimate the pose of the hexacopter, taking into account the state propagation that is sampled and sent from the PEKF at 25Hz. The PEKF only uses the x and y position estimates from the pose estimate.

Firstly, the reasons for choosing this CEKF approach are presented and then the objectives that this approach should meet are introduced. The explanation of the approach follows, where the instruments used are described and it is discussed the implementation procedures of the PEKF and the BEKF.

A working implementation of a laser-based localization algorithm for ground vehicles was available, which could run on a linux-based system. Hence this implementation could not be computed on the Pixhawk, but only on the BeagleBone. In order to use this algorithm in the project, two options were available:

1. Use only one EKF for xy localization, which would be computed on the BeagleBone. To make this possible, one needs to send the IMU information from the Pixhawk to the BeagleBone, where it can be used to propagate the state and compute the laser-based position.

2. Design an EKF running on the Pixhawk, PEKF, that computes high frequency position and velocity

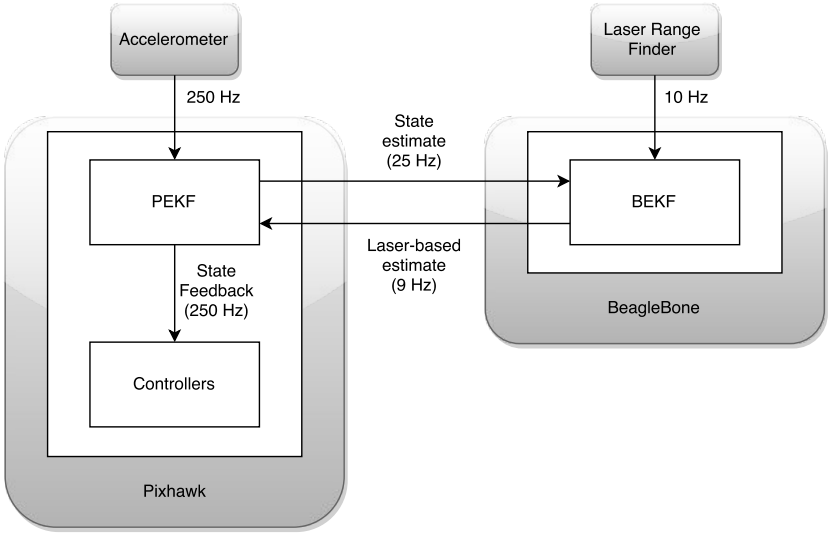


Figure 3.8: xy position Cascaded Extended Kalman Filter architecture. The grey boxes represent hardware devices, whereas white boxes represent software modules.

estimates and receives laser-based position estimates from the EKF running on the BeagleBone, BEKF.

A few problems may arise when using the approach 1:

- Position estimation becomes completely dependent on the BeagleBone;
- Problems with communication between Pixhawk and BeagleBone become extremely dangerous, since the estimation, which is is a critical part of the control loop of the system, would only be computed only on the BeagleBone;
- Delays may arise from the communication, both when sending IMU information (Pixhawk to BeagleBone) and when sending the state estimates (BeagleBone to Pixhawk), which is not desirable in real-time control. Since IMU information cannot be accessed directly from the BeagleBone, the measurements are received at the Pixhawk and then would have to be sent to the BeagleBone. If a varying delay is created on the IMU measurements received on the BeagleBone, this can make the sampling frequency vary on the Kalman Filter running there, which would deteriorate the state propagation and possibly invalidate the assumption of constant acceleration in between iterations.

Approach 2 was chosen not only because of the disadvantages of approach 1, but also due to the advantages:

- have a functional KF on the Pixhawk that is not designed specifically to the LRF.
- Provide modularity on our system. By designing the PEKF non dependent from the BEKF, the algorithm used to extract position information from the laser scanner information can be changed easily, completely independently from the core KF (PEKF), which is estimating the states of the system.
- Ease of expansion. The PEKF can be modified without needing to change the BEKF. It is provided the capability to add other sensors or estimators that can complement the position information

given by the BEKF, without the need to change the BEKF. Moreover, the BEKF's output is also improved when the PEKF is improved, since the results of the BEKF improve with the quality of the information being fed by the PEKF.

- Ability to withstand short laser information outages.

3.3.2 Objectives

The cascaded approach CEKF poses the problem of not guaranteeing convergence of the estimated state to the real solution. Hence our main objective is to verify that this approach works and that it does not pose a threat on the stability of the system. This cannot be verified theoretically, thus it will be analyzed in the results Section 4.3.2. Assuming that state convergence is observed at all times, the objectives of this estimator are:

- providing state estimates at a 250Hz frequency,
- estimating x and y position with 10cm accuracy,
- estimating x and y velocity with 0.15m.s^{-1} accuracy,
- estimating x and y position with a delay lower than 0.1s.
- estimating x and y velocity with a delay lower than 0.05s.

3.3.3 Sensor model

The sensors used in the CEKF are an accelerometer that measures acceleration along the x and y axes and a LRF measuring the distances of points along a 240^o angular range of view. The accelerometer is responsible for propagating the position and velocity states of the system along both x and y dimensions, at $f_1 = 250$Hz. The accelerometer's model is presented as (3.3) and will be used for both x and y axis, with the same characteristics as explained in Section 3.2.3. The raw information of the laser scanner is not fed into any Kalman Filter; instead the BEKF uses information associated to line features extracted from laser range readings. Hence, there is no need to design a model for the LRF measurements. On the other hand, the position estimates from the BEKF need to be modeled, since they are fed to the PEKF as a measurement input in this Kalman Filter. The BEKF position estimates are modeled as (3.2), which is discussed in Section 3.1.3, and have a sampling rate of $f_2 = 9$Hz.

3.3.4 Implementation

Pixhawk EKF

The EKF running on the Pixhawk, PEKF, is responsible for the estimation of all the state variables related to the x and y dimensions, and these estimates will be fed to the position controllers. The algorithm does sensor fusion of the accelerometer and of information based on LRF readings. The accelerometer's raw information is computed in the PEKF, while the raw laser distance information is computed outside this

filter, in the BEKF. The PEKF receives estimates of the x and y position of the hexacopter, from the BEKF.

The system dynamics are based on the kinematics of a free body, as in (2.27). In our sensor-based model, the kinematic-based system has the state $\boldsymbol{x} = \left[p_x, v_x, b_{ux}, p_y, v_y, b_{uy}\right]^T$. The navigation mechanization equations are

$$\dot{\hat{\boldsymbol{x}}} = f(\hat{\boldsymbol{x}}, \hat{a}, \boldsymbol{\omega}) = \begin{bmatrix} \hat{v}_x \\ \hat{a}_x \\ 0 \\ \hat{v}_y \\ \hat{a}_y \\ 0 \end{bmatrix} = \begin{bmatrix} x_2 \\ u + \hat{x}_5 \\ 0 \\ x_4 \\ u + \hat{x}_6 \\ 0 \end{bmatrix}. \tag{3.18}$$

To achieve a discrete model sampled at instants $t_k = kT$ of the form $\delta\boldsymbol{x}_{k+1} = \Phi_k \delta\boldsymbol{x}_k + \boldsymbol{w}_k$, the (3.18) is integrated numerically. Admitting that the acceleration is constant between integration time instants of length $\tau = \frac{1}{f_1}$, one gets the Transition State Model in (3.19) and Control Input Matrices $\mathbf{B}_1$ for the x axis accelerometer and $\mathbf{B}_2$ for the y axis accelerometer, in (3.20).

$$\boldsymbol{\Phi}(\tau_i, \tau_{i-1}) = \begin{bmatrix} 1 & \tau & \frac{1}{2}\tau^2 & 0 & 0 & 0 \\ 0 & 1 & \tau & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \tau & \frac{1}{2}\tau^2 \\ 0 & 0 & 0 & 0 & 1 & \tau \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

$$\boldsymbol{B}_1 = \left[\tfrac{1}{2}\tau^2, \tau, 0, 0, 0\right]^T \qquad \boldsymbol{B}_2 = \left[0, 0, 0, \tfrac{1}{2}\tau^2, \tau, 0\right]^T, \tag{3.20}$$

where our input in $\mathbf{B}_1$ and $\mathbf{B}_2$ is considered as being the accelerometer measurements in axes x (u_x) and y (u_y), respectively.

In our cascaded approach, the output of the BEKF is used as a measurement input in the PEKF, using the observation matrix written in (3.21). The measurement correction is computed as shown in (3.22), with an associated output error model presented in (3.23).

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.21}$$

$$\boldsymbol{z}_n = \begin{bmatrix} y_{xk} & \dot{y}_{xk} & y_{xk} & y_{yk} & \dot{y}_{yk} & y_{yk} \end{bmatrix}^T - \mathbf{H}\,\hat{\boldsymbol{x}}_k^- \tag{3.22}$$

$$\begin{bmatrix} y_{xk} & \dot{y}_{xk} & y_{xk} & y_{yk} & \dot{y}_{yk} & y_{yk} \end{bmatrix}^T = \mathbf{H}\,\delta\boldsymbol{x} + \begin{bmatrix} 1 & f_1 & 1 & 1 & f_1 & 1 \end{bmatrix} \eta_1 \tag{3.23}$$

The measurement model described by the Observation matrix in (3.21) accounts for a velocity estimate provided by the BEKF. The computation of this estimate will be discussed in Section 3.3.5.

The decision of using static Kalman Gains was taken for two reasons:

1. not having access to the covariance matrix associated to the BEKF's estimates, which means that the covariance matrix of the system cannot be propagated in the best way.

2. the computational complexity decreases greatly if the covariance matrix is not propagated.

Since this approach proved to bring good results, the complexity of the estimator was not increased.

BeagleBone EKF

The IMU information is very important to propagate the system state, since it provides very fast estimates and can detect high-frequency dynamics; moreover dead reckoning is simple and inexpensive. Although, the uncertainty of the robot's pose increases with time and, especially, as it moves. In order to reduce the tracking error, the localization approach described in Section 2.7 is used to get an absolute position measurement, based on the environment. This algorithm is computed in the BeagleBone, hence being named as BeagleBone's Extended Kalman Filter, BEKF. It is taken advantage on the fact that man-made indoor environments are generally structured, commonly containing straight walls and fixed obstacles with straight surfaces, which can be well represented by line segments; obstacles that cannot be well represented as such are neglected, and the measurements of these obstacles, collected by the laser sensor, will be rejected as outliers in our model.

There was an already implemented version of this algorithm in DTU's real-time control software Mobotware. This algorithm is processed in soft real-time by the module called *aulocalize*. Some changes were made to this method in order to make it work with the hexacopter.

The EKF running on the BeagleBone, BEKF, is responsible for processing the information from the LRF and estimating the pose of the robot in the 2-D space. The algorithm receives information from the PEKF and the Pixhawk's attitude estimator, in order to have an initial prediction of the pose of the robot. The Pixhawk sends estimates of yaw and x and y position at a 25Hz rate. The computation of the yaw estimates was explained in Section 3.1.4 and the computation of the position estimates is explained in Sections 3.3.4 and 3.3.5. The BEKF is responsible for creating estimates of the same variables and sending them back to the Pixhawk, in order to correct the state that will be fed to the position controller.

Two approaches were taken considering the position prediction step, which will now be discussed.

1. Only rotational motion is predicted.

   By using the yaw estimate sent from the Pixhawk, the displacement of yaw since t_{i-1} is computed, $\Delta\psi$. This displacement is added to the yaw estimate resulting from the last localizer iteration, resulting in the wanted giving the yaw prediction, ψ_i.

The translational motion is neglected, which means that the x and y prediction is the corresponding localizer's position estimate from last iteration. The way to deal with this bad prediction is to increase the measurement uncertainty, which gives rise to a prediction which is a Gaussian distribution centered around the last known position and with variance high enough to reach the current position. The covariance matrix is updated as

$$P_t = \begin{bmatrix} 0.11^2 & 0 & 0 \\ 0 & 0.11^2 & 0 \\ 0 & 0 & 0.088^2 \end{bmatrix},$$

where the standard deviation equivalent to 11cm in x and y is chosen in order for the localization algorithm not to trust that the position of the hexacopter remained the same since the previous iteration, which was 0.11s ago.

2. Both translational and rotational motion are predicted.

   On this approach, the whole pose of the robot is predicted in the analogous way as the one explained for the yaw on the previous paragraph. The displacements of the state since the last localizer iteration are computed using Pixhawk's information. These displacements are added to the previous estimates given by the localization algorithm, resulting in the pose prediction. The covariance matrix is updated as

$$P_t = \begin{bmatrix} 0.1^2 & 0 & 0 \\ 0 & 0.1^2 & 0 \\ 0 & 0 & 0.088^2 \end{bmatrix}.$$

The first approach was used initially, before having a trustworthy Kalman Filter estimating position running on the Pixhawk. This implementation showed as good performance as the second method, which proved that the algorithm provides good results when using only rotational prediction. But the second approach was preferred to the first one because there is a difference on the output of the localization algorithm. When feeding the position prediction with zeros, the position error will be relative to (0,0), which is equivalent to having an absolute measurement of position. On the other hand, when giving the current position to the localizer algorithm, its output will be the position error relative to the Pixhawk's position, which proved to be of more value. Another favorable point to the second approach is that it makes the localization algorithm run faster, because the predicted features (computed in the Measurement Prediction step) are more accurate. There is, however, a drawback to the second method: by depending on Pixhawk's estimates to predict the current position, the localization algorithm takes the risk that the Pixhawk's estimate is too different from reality, making it not able to accurately match predicted features with observed ones, which may lead to a bad localizer position estimate.

3.3.5 Implementation Discussion

Pixhawk EKF

The approach presented in Section 3.3.4 will be further discussed in this Section, where important implementation details will be addressed.

In our cascaded approach, the output of the BEKF is used as a measurement input in the PEKF.

Ideally, the position estimate provided by the BEKF would come with the covariance matrix associated to the estimates. Instead, due to the lack of these values, a fixed variance value for each variable estimate received from the BEKF is assumed. This solution simplified the initial cascaded EKF approach, but should be improved in the future, to ensure that laser-based estimates do not worsen the estimates of the PEKF, in the case that the BEKF's associated covariance matrix becomes higher than expected. This improvement was not implemented because good results were achieved with the solution described.

As explained in Section 3.2.5, the same approach of correcting the state continuously at a high frequency using low-frequency information is used. This eliminates small discontinuities in the state variables that originated in state updates. The correction step is executed at a 250Hz rate with (3.13), instead of being executed in one step at the lower frequency of 9Hz, at which the BEKF sends the position estimates. The Kalman Gain vectors used in this approach are presented in (3.24) and (3.25), along with the equivalent Kalman Gains that would be used in a conventional update step $\boldsymbol{K}_9$, executed at 9Hz. Keep in mind that the Kalman Gains used at the higher frequency have higher values than the low-frequency Kalman Gains, but they will affect the state correction with a lower weight than the latter due to the factor τ in (3.13), as it would be expected.

$$\boldsymbol{K}_x = \begin{bmatrix} 2.8 \\ 0.49 \\ 16 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \text{equivalent to the conventional } \boldsymbol{K}_{x,9} = \begin{bmatrix} 0.311 \\ 0.054 \\ 1.778 \\ 0 \\ 0 \\ 0 \end{bmatrix} \text{ at } f_1 \tag{3.24}$$

$$\boldsymbol{K}_y = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 2.8 \\ 0.49 \\ 16 \end{bmatrix}, \quad \text{equivalent to the conventional } \boldsymbol{K}_{y,9} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.311 \\ 0.054 \\ 1.778 \end{bmatrix} \text{ at } f_1 \tag{3.25}$$

The algorithm used on the hexacopter had a Kalman Gain associated to the accelerometer biases that is 4 times higher than wanted, which is shown in (3.24) and (3.24). This made the acceleration bias estimation very susceptible to the accuracy of the laser-based position estimates received from the BEKF, *i.e.*, the bias estimate will easily suffer if these estimates are inaccurate or delayed. The result of the over-correction of the bias can be detected in the position estimation as an oscillation after the robot moves, and will be discussed in Section 4.3.2.

The filter is initialized with

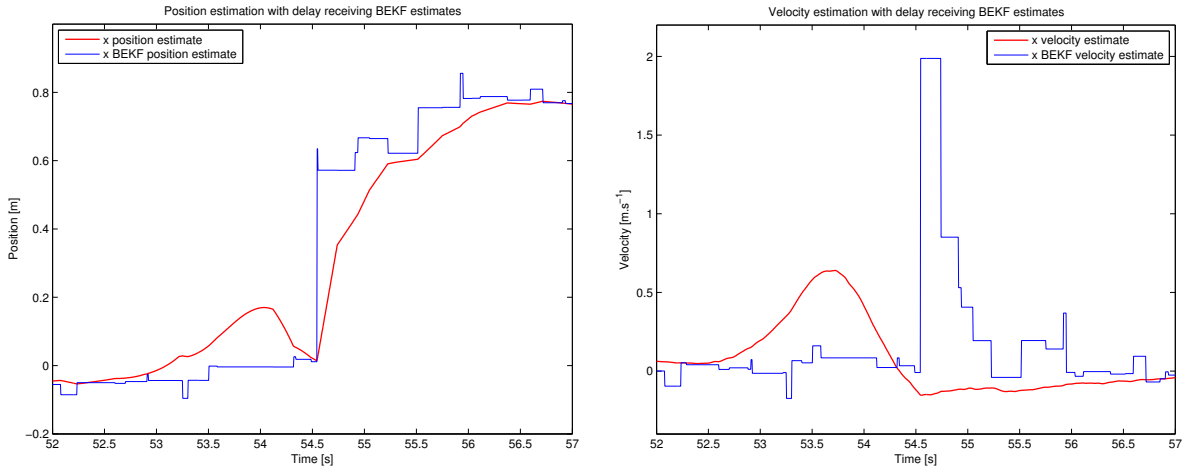
$$\boldsymbol{x}_0 = \begin{bmatrix} E[p_x(0)] \\ E[v_x(0)] \\ \mu_{b_{ux}} \\ E[p_y(0)] \\ E[v_y(0)] \\ \mu_{b_{uy}} \end{bmatrix}, \tag{3.26}$$

where $E[p_x(0)] = 0$, $E[v_x(0)] = 0$, $E[p_y(0)] = 0$ and $E[v_y(0)] = 0$, since the hexacopter will always start in the origin of the PF. The biases are initialized as $\mu_{b_{ux}}$=0 and $\mu_{b_{uy}}$=0. There is no need for a special process to initialize these variables, since their values will converge to a good estimate in less than 5 seconds, while the robot is motionless on the ground. As explained previously on Section 3.3.5, it is not possible to predetermine a good initialization of the accelerometer biases because they vary especially with ambient temperature.

The output position estimates of the BEKF arrive to the PEKF with a delay bigger than expected. Figure 3.9(a) shows the poor position and velocity estimates when using delayed information from the BEKF, at this point the delay could reach 1s. An intensive inspection of all the possible origins of the delay was conducted and several modifications were made. The communication program that enables the interaction between BeagleBone and Pixhawk, MavlinkComm (addressed in Section 3.1.2), was redesigned to have the fastest communication possible in both directions, since the same (serial) communication buffer is used for reading and writing. The time between each of the steps of the communication executed was analyzed, from the output of the BEKF until writing the estimates on the communication buffer. The time that the BEKF took to converge to the correct estimate was roughly analyzed and showed at most a 2 iteration delay, corresponding to a maximum delay of 0.22s from reality. The hardware used for the communication was also changed from an FTDI-USB converter to a UART-UART cable. After all the modifications, the delay was greatly decreased, varying from 0.1s to 0.25s, as can be observed experimentally in Section 4.3.2. A delay of this magnitude brings additional complexity to the correct estimation of the system's state, which will be further debated.

From Figure 3.9(a), one can conclude that the laser-based absolute position estimates (BEKF estimates) may weaken the estimates if they arrive with delay, being the impact more negative when the helicopter moves. On the other hand, they are an essential part of the estimation because they guarantee that the PEKF estimates have a bounded error.

At first, the delay posed a big problem. One can exploit the fact that the accelerometer is able to give reasonable measurements with virtually zero delay, even though the resulting estimates drift. One approach is to correct this drift. This could be achieved by making the BEFK give position error estimates as output instead of absolute position estimates. Let us consider that a position correction is computed at epoch t_1 in the BEKF. This value will arrive delayed to the PEKF at epoch $t_1 + \Delta t$, where Δt represents the delay associated to the estimate. Let us correct the accelerometer position estimates at $t_1 + \Delta t$ with the correction estimated at t_1. The correction received gives the drift of the position estimate at epoch t_1.



(a) (b)

Figure 3.9: PEKF innacurate estimation of x position, Figure (a), and velocity, Figure (b), when being fed BEKF information arriving with delay around 1s.

Relying on the fact that the accelerometer's position estimate has drift of Δx_1 at epoch t_1, than it still has the same drift at epoch t_2 ($t_2 > t_1$) plus the drift it accumulated between t_1 and t_2, which is Δx_2, giving a total drift $\Delta x_1 + \Delta x_2$ at t_2. This way, it is possible to correct the drift Δx_1 (drift existing at t_1) on epoch t_2 with the delayed correction, decreasing the drift at t_2 only to Δx_2, instead of $\Delta x_1 + \Delta x_2$. If the past drifts are corrected at each laser estimate epoch, then the position estimate error is bounded. By doing these corrections at each epoch, the only drift in the estimates should be the one accumulated since an instant in the past as big as the delay. This approach results in estimates with the fast response of the accelerometer and the bounded error of the laser scanner. This approach proved to be unsuccessful because it can only correct position. If the velocity is not corrected, then the velocity estimate error is unbounded and the position estimate will not converge. If position error is used to correct the velocity estimate, the position estimate will not converge as well and velocity estimates, which are an essential input in the position controller of our control system, get even worse.

After lowering the delay to the minimum possible, $<$0.25s, it became pointless to have a special approach to estimate position.

Velocity estimates are the most important for the stability of our control system, since they will be responsible for the control of the faster responses of the system. Hence the velocity state has to be accurate in real-time, with the minimum delay possible. Position estimates are obviously important, but their accuracy is not as important for the stability of the system. The most important concern regarding the PEKF's position estimates is for them to be accurate enough when providing them to the BEKF, so that this filter is able to converge to the real position of the robot. If the PEKF's position estimates are too far from the true value (rule of thumb would be more than 0.3m), the BEKF may not be able to use the laser information to compute an accurate position estimate (on that iteration) or even converge to the correct position estimate.

The computation of noise-free velocity estimates from the BEKF position estimates aggravated the delay issues. It is desired an estimation approach that gives a response as fast as the one from the accelerometer, hence a technique was used to fuse delayed measurements in the EKF. Several tech-

niques can be used, while the ones that are the most straight forward are filter replay [50], [51], and measurement extrapolation [52]. In the first approach, the filter goes back to the time in which the measurement was taken, the measurement correction is applied and the filter is replayed from that moment, using the corrected state. In our case, this approach requires storing the information received from the accelerometer, at 250Hz, during an interval whose length is the upper bound of the lag, in order to be able to replay the state propagation. An approach similar to the one presented by Larsen et al. [52] is used, but neglects the concern of the updating the covariance matrix.

To explain the approach used, let us consider that delayed position information is received from the BEKF at t_{now}, which will be used to compute a velocity estimate from the instant of time $t_{now} - \Delta t$, *i.e.*, the velocity estimate computed at t_{now} is the real velocity of the robot at $t_{now} - \Delta t$. The approach is as follows:

1. at t_{now}, compute a velocity estimate (relative to $t_{now} - \Delta t$) by differentiating delayed position estimates received from the BEKF;

2. determine the elapsed time Δt;

3. compare the computed velocity estimate (relative to $t_{now} - \Delta t$) with the velocity state of the PEKF from $t_{now} - \Delta t$, and calculate the correction based on that past estimate;

4. use this correction to update the velocity state of the PEKF.

In summary, the velocity state of the PEKF is corrected at t_{now}, based on measuring how wrong the state was in the past, at $t_{now} - \Delta t$. This approach is based on the fact that an error that existed in the past (error from drift originating on acceleration integration) still exists in the present, thus it can be corrected.

The value of Δt is far greater than the delay that the BEKF position estimates have, because Low-Pass Filters are applied in the computation of the velocity estimates, described in step 3, in order to remove noise in BEKF estimates that was being amplified when it was differentiated. A Low-Pass Filter is applied to the position BEKF estimates and then another LPF is applied to the resulting velocity estimates. This results in delayed velocity estimates with scaled value, being slightly smaller than reality.

BeagleBone EKF

The approaches mentioned in Section 3.3.4 were the ones adopted, because of their simplicity and good results. However these can be improved and it will now be discussed how that is accomplished for both approaches, on the position prediction step. When mentioning the previous localizer iteration, t_{i-1}, it should be understood as the last iteration where the whole algorithm was run.

1. Only rotational motion is estimated.

   Yaw estimates are sent from the Pixhawk every epoch k, which occur at 25Hz. This estimate is the result of an attitude EKF running at 250Hz. In this attitude EKF, the yaw is propagated with

gyroscope information and corrected at approximately 9Hz, where the correction is done using the yaw estimate computed in the localization algorithm presented in Section 2.7. An iteration of the localization algorithm takes place in an epoch t. In the position prediction step, a yaw prediction is calculated and also the corresponding measurement error. The yaw prediction is achieved by adding the displacement since the last localizer iteration, $\Delta\psi_{total}$, to the yaw estimate from the last localizer iteration, ψ_{t-1}, and finally computing the associated error of this prediction. The measurement error is proportional to the cumulative yaw displacements since the last localization iteration, plus a term that represents the uncertainty of the Pixhawk's estimate. This means that even in the case that the Pixhawk's yaw in time t is the same as in $t-1$, i.e., $\Delta\psi_{total} = 0$, but the estimates have changed and gone back to the same value, i.e. $\sum_k |\psi_k| > 0$, then this motion of the hexacopter is taken into account to grow the uncertainty of the yaw prediction.

The translational motion is predicted as zero. In reality, this is not true when the hexacopter moves, so in order for the localizer not to take this information into consideration, the error measurement is chosen to be how much the hexacopter is able to move in between iterations of the localizer, i.e., in a time interval of 0.11s. The value should be sufficiently big so that it presents a margin for the localizer to give a correct estimate, if the position changes.

The covariance update, in each iteration k while going through the poses received from the Pixhawk, since the last localizer iteration t_{i-1}, is:

$$P_k = P_{k-1} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \Delta\psi_k * 0.2 \end{bmatrix},$$

where $\Delta\psi_k$ is the displacement of the yaw angle in between estimates sent from the Pixhawk and $P_0 = P_{t-1}$.

After going through the pose history received from the Pixhawk, the covariance matrix is updated to account for the uncertainty of the estimate using

$$P_t = P_k + \begin{bmatrix} 0.1^2 & 0 & 0 \\ 0 & 0.1^2 & 0 \\ 0 & 0 & 0.088^2 \end{bmatrix},$$

which allows the localizer algorithm to accept a motion of 10cm on the x and y axis since the last localizer iteration and also allows the algorithm to accept estimates 5 degrees off from $\psi_{t-1} + \Delta\psi_{total}$.

2. Both translational and rotational motion estimated.

   If the Pixhawk provides x, y and yaw estimates, the current pose can be predicted. Rotational motion is treated in the same manner as explained in 1) and translational motion is treated analogously.

   The covariance update is done, while going through the Pixhawk's pose history since the last localization:

$$P_k = P_{k-1} + \begin{bmatrix} \Delta x_k * 0.2 & 0 & 0 \\ 0 & \Delta y_k * 0.2 & 0 \\ 0 & 0 & \Delta \psi_k * 0.2 \end{bmatrix},$$

and, after going through all the pose history, the estimate's uncertainty is added:

$$P_t = P_k + \begin{bmatrix} 0.07^2 & 0 & 0 \\ 0 & 0.07^2 & 0 \\ 0 & 0 & 0.088^2 \end{bmatrix},$$

which gives an uncertainty of 7cm in x and y and of 5 degrees in the yaw estimate about the predicted pose.

Notice that the covariance matrices are not being multiplied by the corresponding Jacobians matrices. Since x, y and yaw estimates from the Pixhawk are uncorrelated and there is no motion model involved that correlates these entities, the Jacobian is the identity matrix.

Jensfelt [14] presents an approach similar to the one presented in Section 2.7 and compares it to two global localization strategies: Multiple Hypothesis Localization and Monte Carlo Localization. It was found to be more efficient than the other two, and to be operational even in dense clutter. Even though no direct performance comparison is made, the pose tracking algorithm similar to ours has less limitations, since the other two turn out to have bigger requirements in terms of features.

One handicap of the algorithm presented in Section 2.7 when using it to represent multiple rooms, with parallel walls, is that these may be mistaken. The algorithm erroneously is not able to distinguish between the features from the map that the robot can physically observe with the laser scanner, from all the features that are close enough to be considered inside the range of sight. This means that if the robot is facing a wall, it cannot see the features behind that wall. Nonetheless, the algorithm is not able to reject those features and will predict, on Step 3 of the algorithm, features that are behind the wall, as long as they are within the range of sight of the robot. This may lead to problems in the matching step, if these wrongly predicted features are matched to observed features.

As described in Section 2.7, this algorithm is feature-based, where line segments are extracted from sensor data using RANSAC. Among many geometric primitives, line segment is the simplest one and accurately describes most office environments [36]. RANSAC is evaluated as having bad speed and correctness, but produces more precise lines than other line extraction algorithms and its accuracy can be increased by performing more iterations [36]. By using RANSAC, outlier rejection of small curved features is done spontaneously. This is especially good in our thesis, since that people are expected to be around the MAV and RANSAC will automatically reject them as features. Even if incorrect small lines representing people are extracted with RANSAC, they are not likely to be matched to any feature in the map.

The laser has a limited range of 4m. This means that this algorithm will not work if the robot is flying in the middle of big empty open spaces, without any surfaces that the laser can detect. Other approaches should be used to bound the position error in these cases.

It is required to have features along more than one axis for the localization algorithm to work. It is necessary to have line features which create a certain angle between them, ideally bigger than 20^o, in

any circle of 4m radius in the area that the localization algorithm is computed.

Chapter 4

Results

In this Section, the experimental results will be presented and discussed, along with relevant simulation results. The results of the localization originally provided by the Pixhawk are shown, in an indoor environment, in Section 4.1. The results of the implementation discussed in Sections 3.2 and 3.3 are presented in Section4.3. Due to the lack of a source for ground truth in the experiments, simulations were conducted to examine the performance of the algorithms in estimating the state of the system. The original and propose approaches are compared in light of the goals of this project. An equivalent experiment is performed in similar conditions for each pair of original-developed approach, so that the comparison between approaches is possible. In these experiments, the UAV is held manually.

4.1 Problem Description

The Pixhawk system has a localization algorithm available. Although, it is not proper for indoor environments.

The altitude estimator is not appropriate for the goals of this project, since it only uses barometer and accelerometer measurements for altitude estimation. It is targeted for outdoor use, where the barometer can be combined with GPS to provide altitude estimates. In indoor environments, GPS is not available and the barometer by itself does not provide results with enough accuracy or resolution; moreover it is not able to give absolute height measurements and its error grows with time due to its varying bias. Consequently, the barometer is not suitable. Moreover, there is the need for estimation of absolute altitude above ground. Consequently, an ultrasonic sensor is added. The Pixhawk's algorithm is ready to use information from the ultrasonic already, but does not estimate altitude with it, only measures distances to the nearest object.

An Extended Kalman Filter is created to estimate altitude position and velocity by fusing information from IMU and sonar, using also barometer measurements in a sonar failure situation.

The original Pixhawk's localization algorithm is able to perform localization in x and y dimensions if one of the sensors is provided: GPS, PX4FLOW module (added in the middle of this project) or a motion capture system. GPS is unavailable, as explained, and a motion capture system is not a self-

contained approach, but the PX4FLOW module is a possible approach that is able to provide velocity estimates. There are two disadvantages of using the latter solution: the integration of velocity leads to drift in position and there is no absolute position measurement, consequently localization with absolute position is not possible.

In this thesis, two Extended Kalman Filters are used cascaded: one fuses IMU information with position estimates; the other provides these absolute position estimates based on LRF information.

In order to interpret the results correctly, a few details should be discussed. In the following experiments, a source for ground truth is not available.

- In the altitude solution results, the reader can rely on the sonar position measurements to have a notion of reality, since this sensor gives very good measurements with high resolution. On the other hand, the sonar velocity estimates are not as reliable, since they correspond to the differentiation of the sensor's position measurements. The sonar measurement has the lowest sampling frequency ($f_s = 10$Hz), thus a new measurement is not available on every KF iteration; a new measurement is indicated with a circle and the corresponding line indicates the latest measurement value known.

- In the xy localization solution, handheld experiments consist of moving the drone from a still position in one point to another; the robot is held as motionless as possible in the starting and ending points. On experiments with the flying robot, it moves from one setpoint to another.

Due to the lack of ground truth, simulations of the algorithms are executed in order to perform error analysis. The simulation scenario has been discussed in Section 3.1.3.

On the xy localization results, it will only be shown results from the y dimension, since x and y variables are estimated exactly in the same way and exhibit the same behavior.

4.2 Original Solution

4.2.1 Altitude Estimate

The original solution for altitude's position and velocity estimation is shown in Figures 4.1(a) and 4.1(c), respectively. This is the experimental result of the original EKF that comes with the PX4 autopilot. By comparing to the sonar measurements, one can easily see that it does not accurately measure absolute altitudes. This handicap comes from the use of the barometer as the only sensor that corrects the altitude estimate; the sonar information is not used. This may seem non sense, but the logical reason is that this system is originally designed for outdoor use, where the sonar range is easily exceeded, thus the algorithm is not used to be dependent on the sonar information; also in outdoor environments, it does not make sense to measure altitude with sonar in the presence of obstacles such as buildings and trees. In this algorithm, the sonar information is used only to give an idea of the vertical distance to obstacles, not as a sensor that affects the control loop of the system. Figure 4.1(b) shows a zoom-in of Figure 4.1(a) to show a bad case where the barometer measurements suffer from a perturbation and that greatly affects the position state. The barometer signal looks different from what will be shown later on Section 4.3.1, because the barometer measurements are not filtered in this algorithm.

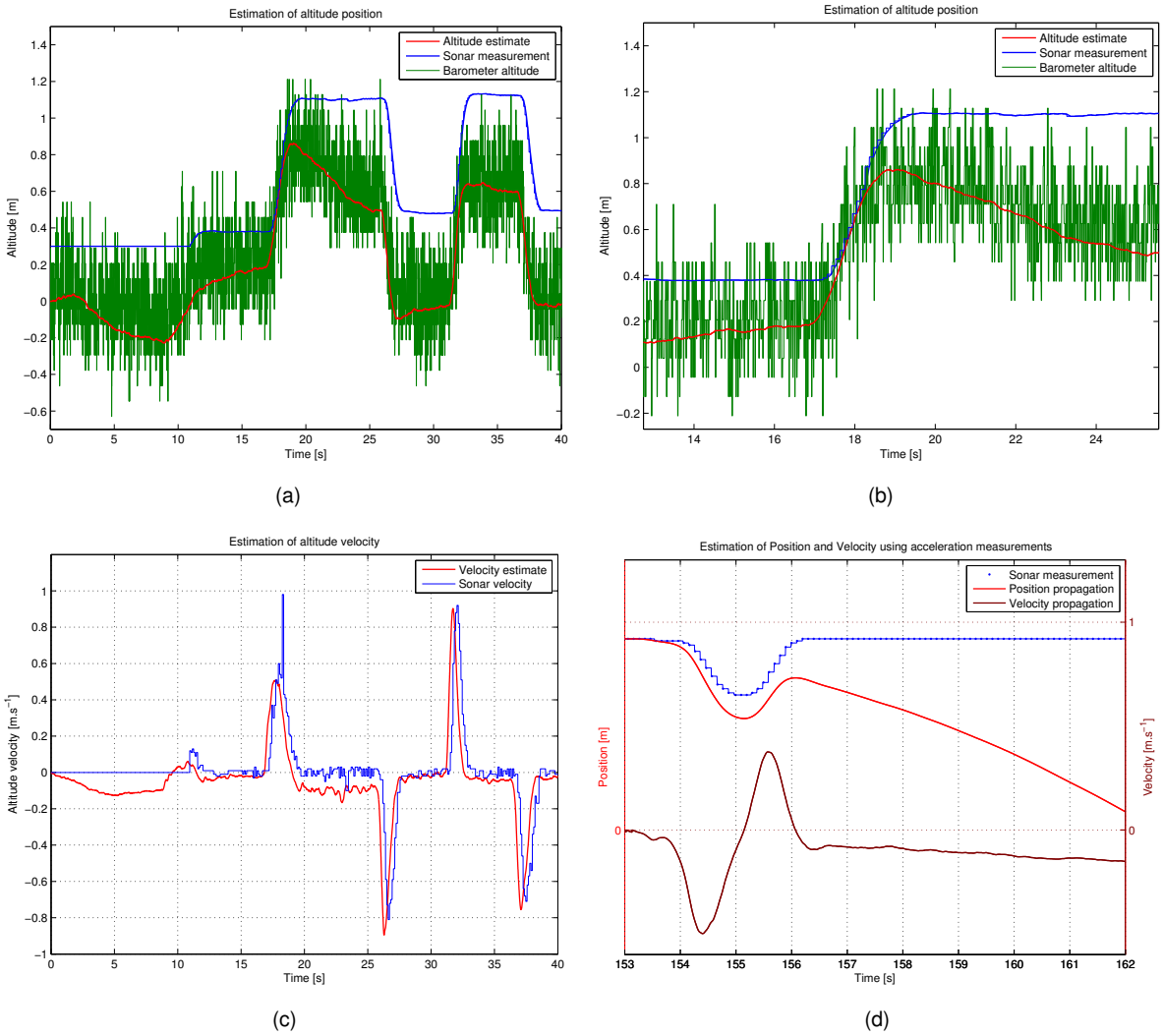


Figure 4.1: In this Figure, two different experiments are performed holding the hexacopter manually: one on (a), (b) and (c); another on (d). Figure (a) shows the estimation of altitude position, while Figure (c) shows the altitude velocity estimation, both using the original EKF from the PX4 Autopilot. Figure (b) shows a zoom-in of Figure (a), at a situation where the error of the position estimate increases due to drift in the barometer measurements. Figure (d) shows the result of estimating position and velocity only with accelerometer measurements; they are computed by the double integration and integration of accelerometer measurements, respectively. Both experiments were done performing the movement showed by the sonar's position measurement curve, which can be seen as a guide.

Figure 4.1(d) shows the position and velocity estimates that the accelerometer provides. A small deviance in the bias b_u estimation is equivalent to a static error in the acceleration. One can see that this leads to the velocity estimate growing linearly when it should remain zero, from $t = 156.5$s to $t = 162$s. As a result, the position error will grow quadratically. This proves that the accelerometer cannot be used by itself to estimate position or velocity.

4.2.2 xy Estimate

The result of the original Pixhawk's localization algorithm is shown in Figure 4.2. Several tests were performed where the MAV is handheld and moved back and forth between positions 0 and 1m. The test where the best results were achieved is shown in Figure 4.2. A PX4FLOW module is added in order

to perform this experiment. The Optical Flow module provides velocity measurements in the x and y dimensions, based on tracking features on the ground observed by the camera of the module. This measurements are used to correct the velocity state and the accelerometer's bias state.

Drift when the robot is motionless should not occur, because the Optical Flow sensor can estimate low velocities accurately, which means that it can detect if the robot is motionless. In Figure 4.2(b), we can observe that the position estimate drifts 10cm in the interval $t \in [17, 22]$s and in $t \in [24, 28]$s. The problem is that the bias of the accelerometer is not estimated well-enough using only this sensor, so the state is propagated incorrectly.

A study of Optical Flow sensors used for UAV navigation is done by Gageik et al. [53], from where we can witness several problems in position and velocity estimation:

- the integration of velocity leads to a bias in the position estimate every time the robot moves; this bias greatly increases with the speed of motion;
- the measurements might have scaling issues depending on algorithms used and the accuracy of the distance to features calculation.

Honegger et al. present the need for illumination when using Optical Flow sensor, and the PX4FLOW sensor illumination requirements are described [54]. Not only lack of illumination, but also lack of features detected by the camera may present an accuracy problem. It is always assumed that no moving objects appear in the camera's field of view, but that is an issue to be taken into consideration, since it may lead to measurement error. A few more limitations are pointed to Optical Flow [55]: sub-pixel precision, inaccuracy in presence of image deformation due to rotation and possible trade-off between efficiency and the maximum image displacement that can be computed. The most obvious problem related to the goal of this project is that it is not possible to extract absolute position measurements from Optical Flow measurements.

4.3 Proposed Solution

4.3.1 Altitude Estimate

The results of different experiments will be presented and discussed in this Section. Firstly it is discussed whether or not it is possible to handle sonar outages, relying on the fact that the accelerometer and barometer biases were already estimated using sonar measurements in previous iterations. Then the results of the final EKF version will be presented, where different scenarios that are likely to happen are explored. An error analysis is performed, in order to have performance study of the algorithm.

On Figure 4.3(a), the sonar is disabled and the estimator is using information from the accelerometer to propagate the position and velocity estimates, while correcting the position with barometer measurements. The sonar was only disabled shortly before the experiment, in order to estimate properly b_u and b_y. Having a good estimate of b_u and b_y is very important to compute good estimates from the accelerometer and barometer measurements. Throughout this experiment, the sonar is unavailable, which means that both the accelerometer and barometer biases are not being corrected, but since they

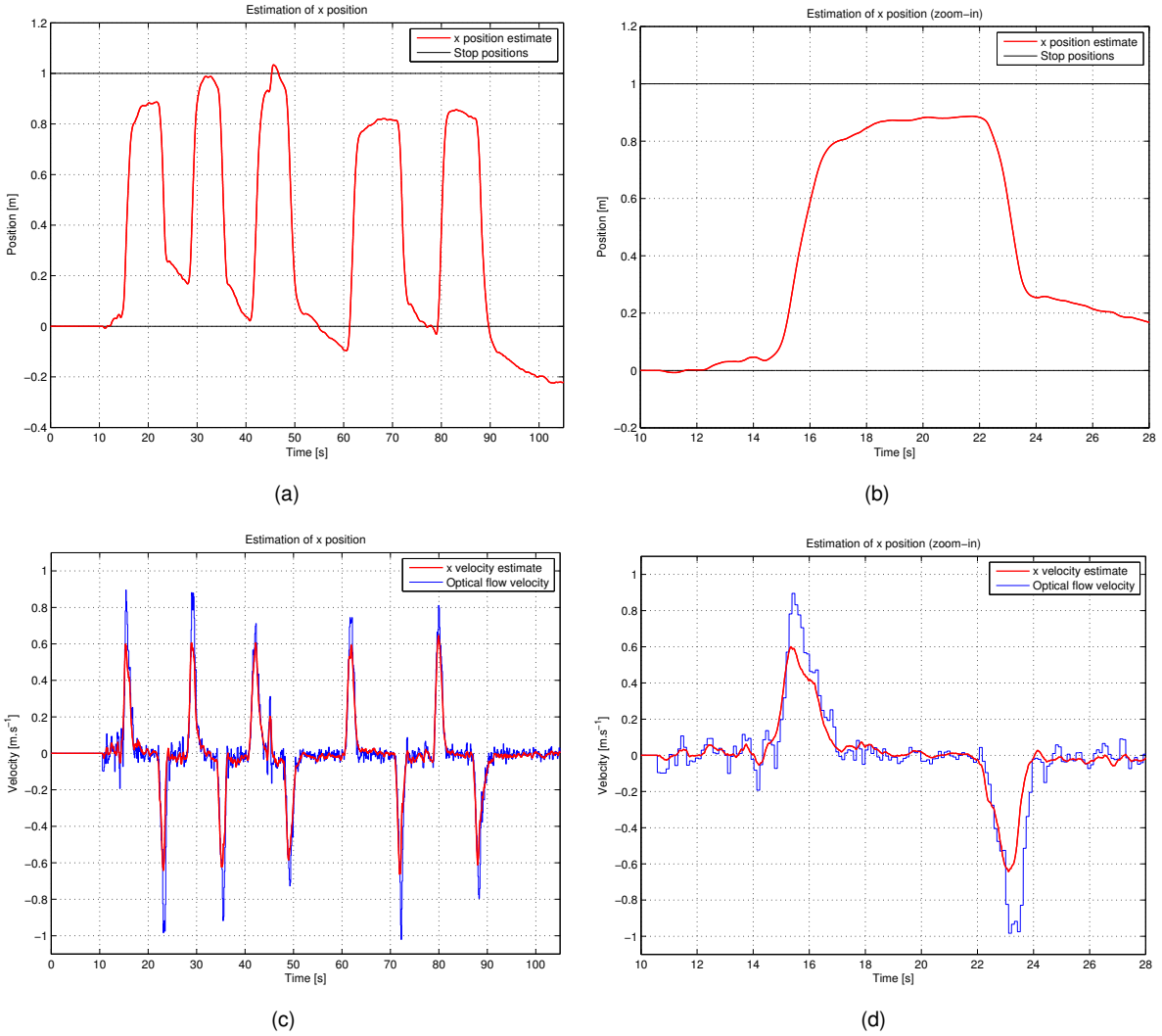


Figure 4.2: Figure (a) shows the estimation of position, while Figure (c) shows the velocity estimates, both in the x-dimension, using original EKF from the PX4 Autopilot. Figure (b) and (d) show a zoom-in of the same experiment. The experiment is performed holding manually the MAV from the starting point and a point 1m away multiple times. The MAV is left motionless at each point for at least 5s, but drift in the position estimate can be observed.

are slow time-varying signals, they are not likely to change by any considerable amount on an interval of the order of 10^1 seconds. The goal of this experiment is to demonstrate that the estimator can withstand short periods of sonar outages with bounded position error. We can see that the barometer signal will grow at a fast rate in the interval $t \in [57.6, 68.2]$, but becomes more delayed as the position starts to vary slower. This behavior is explained by the response of the Low-Pass Filter applied to the barometer signal. As explained in Section 3.2.5, the delay of the barometer signal is very small for fast varying barometer measurements ($t \in$[57.6 , 58.2]), but the delay will grow if the measurements change at slower rates ($t \in$[58.7 , 59.5]). A position error that can go up to 20cm is very noticeable; this error and the delay of the signal are the reasons why the barometer is not used when sonar measurements are available.

Figure 4.3(c) shows the altitude velocity estimation during the same period as Figure 4.3(a) was captured. This velocity is the integration of the accelerometer signal, without any correction. One can easily see that the velocity estimate appears to be approximately the same as the sonar velocity curve,

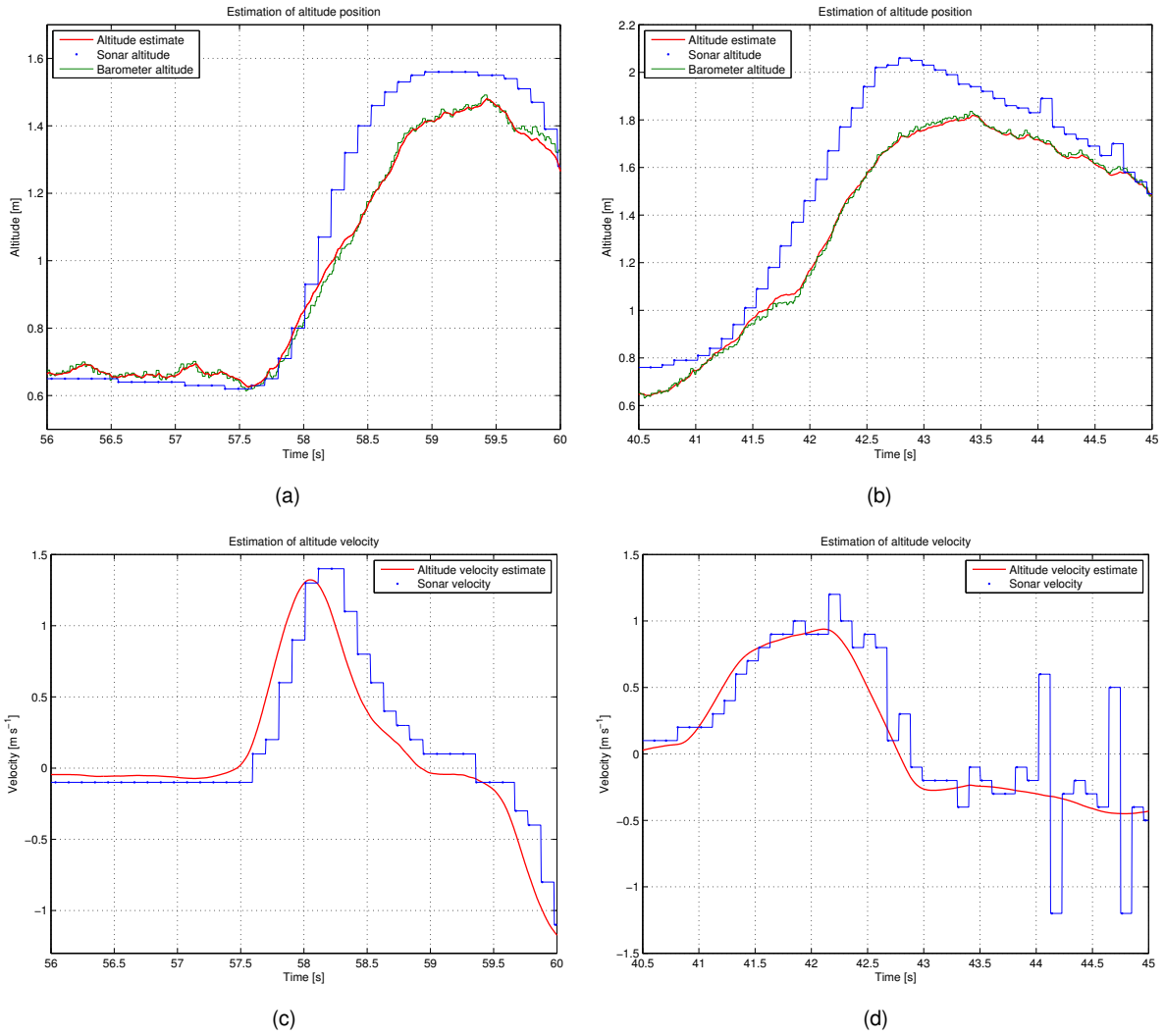


Figure 4.3: Two experiments are shown in this Figure. One experiment is shown on the left side, figures (a) and (c), where the hexacopter is held manually, with the motors turned off. On the right side, figures (b) and (d), another experiment is performed where the hexacopter is flying, being controlled by Remote Control. Figures (a) and (b) show the estimation of altitude position using accelerometer information corrected by barometer low-pass filtered measurements. Figures (c) and (d) show the estimation of altitude velocity using accelerometer information only. The sonar information is shown only as a guide.

but shifted on time. This is because the latter has a phase delay due to the differentiation of the sonar signal, and also the sonar may have a delay of up to 1 sampling period (0.1s). One can conclude that the estimated velocity from the accelerometer is very accurate in short periods of time, if the bias b_u is estimated correctly.

Figure 4.3(b) shows an experiment identical to the one in Figure 4.3(a), but with the hexacopter flying. The Kalman Filter shows the same behaviour. From this experiment, one can conclude that the barometer can be used to correct the accelerometer estimate of altitude if the sonar measurements are rejected (due to noise) for a large amount of time (at least 4 seconds, equivalent to 40 invalid sonar measurements).

Figure 4.3(d) shows the altitude velocity estimation during the same period as Figure 4.3(b) was captured. The integration of the accelerometer signal appears to be more accurate than the sonar estimates when the hexacopter is flying. Very small perturbations on the sonar measurements may lead

to very innacurate sonar velocity estimates, as seen in $t = 44$s and $t = 44.6$s in Figure 4.3(d). The impact of such errors will be addressed when discussing the experiments presented in Figures 4.5 and 4.6.

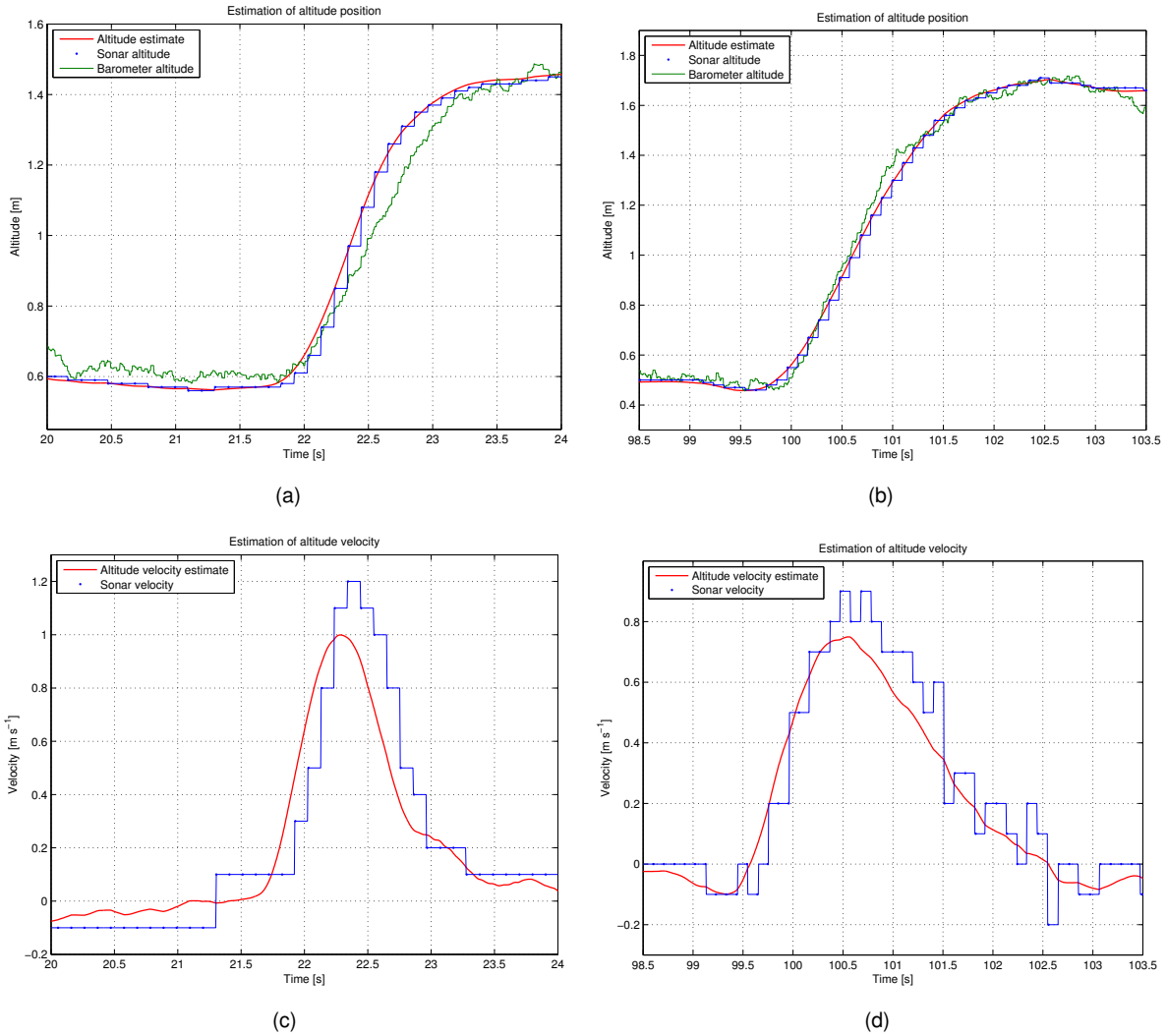
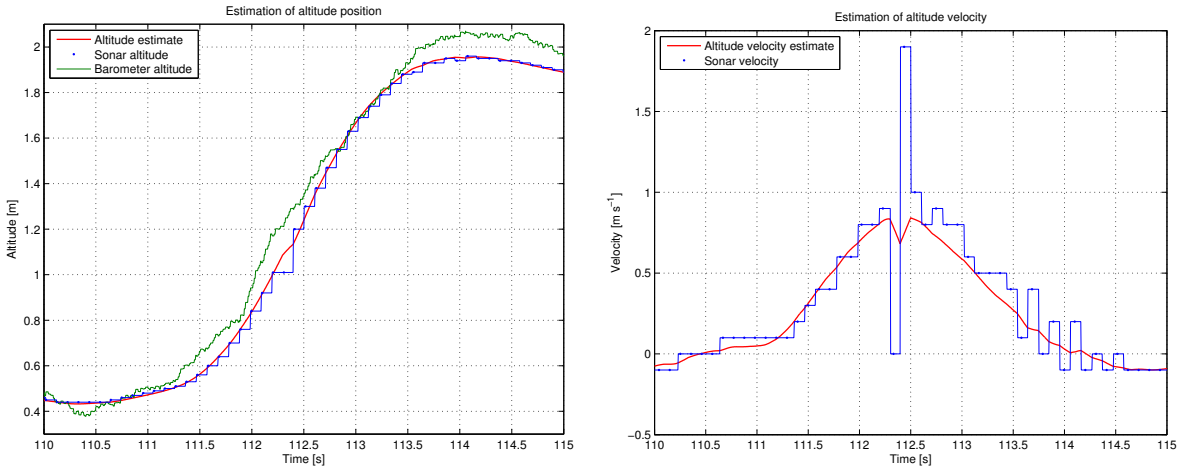


Figure 4.4: Two experiments are shown in this Figure. The first experiment is shown on the left side, figures (a) and (c), with the hexacopter being held manually, with the motors turned off. The other experiment is presented on the right side, figures (b) and (d), where the hexacopter is flying, being controlled with a Remote Control. Figures (a) and (b) show the estimation of altitude position. Figures (c) and (d) shows the estimation of altitude velocity. All of the results are achieved using the final EKF implementation.

The results using the final EKF are shown when holding the helicopter by hand, on Figure 4.4. One can see that the implemented Kalman Filter gives very good results. The position estimate follows the sonar measurement very well. It is noticeable that the estimate accounts for the rise of the hexacopter even before the sonar is able to detect this rise, in $t = 21.7$s, in Figure 4.4(a). On this Figure it is apparent that even when the sonar measurements have stabilized in one value ($t = 23.5$s), the estimate's value rises, which matches the sonar measurements that follow right after. This phenomenon of the estimate predicting the sonar measurements is due to the fact that the accelerometer can measure these signal variations before the sonar. Figure 4.4(b) shows the same behavior when the hexacopter is flying, being controlled by RC. Figures 4.4(c) and 4.4(d) show the velocity estimation with motors turned off and with

the hexacopter flying, respectively. The EKF velocity estimate is better than the sonar velocity estimates, especially when the hexacopter is flying. Figure 4.4(d) shows the sonar velocity going up and down from one estimate to the next, which does not correspond to the real motion, while the estimate is able to follow the curve matching the reality more accurately. The velocity estimate does not reach the values as high as the sonar velocity estimates do, but this small scaling factor will not have a big impact on the control loop of the system.



(a) (b)

Figure 4.5: Figure (a) shows the estimation of altitude position and figure (b) shows the estimation of altitude velocity, both using the final EKF implementation. During this experiment, the hexacopter is flying, being controlled with a Remote Control. Notice the small effects in both position and velocity estimates of a bad measurement of the sonar sensor in $t = 112.3$s.

On Figure 4.5, a bad measurement can be spotted at $t = 112.3$s. It is shown that it has little or no effect at all on the position estimate, in Figure 4.5(a). The effect of a bad measurement on sonar usually leads to a high error on its velocity estimate and may affect negatively the EKF velocity estimate. However, a bad sonar velocity estimate is always compensated on the next measurement, as one can see in Figure 4.5(b). The impact is concluded to be only temporary and to last 2 sonar sample periods, which is equivalent to $\Delta t = 0.2$s.

As explained in Section 2.6.2, one type of noise inherent to the sonar's measurements is spikes, which occur sporadically. Notice on Figure 4.6 that these noisy measurements, at $t = 161.8$s and $t = 162.7$s, are rejected by the validation gate and do not influence at all the EKF estimates.

Figure 4.7 show the results of an experiment where the hexacopter is handheld at the same absolute altitude at all times, but around t=39.1s it is put over a 65cm tall table. The sonar measurements that detect the table are rejected at first due to the sudden change of value in the measurements. After the sonar becomes invalid for a few measurements, the altitude estimate starts being corrected by the barometer, at t=39.25s. Meanwhile, the sonar continues giving the same readings and eventually, at t=39.6s, these readings are considered valid, being established a new ground level. The sonar measurements might seem odd shortly after t=39.1s. The reason that the measurements go from 1.08m to 0.51m rather than going directly to 0.43m, is that the sonar detects the edge of the table as the hexacopter approaches it. The behavior of the EKF in this scenario is as expected: when the table is firstly detected the measurements are considered outliers, but, after the sonar measurements persist, a

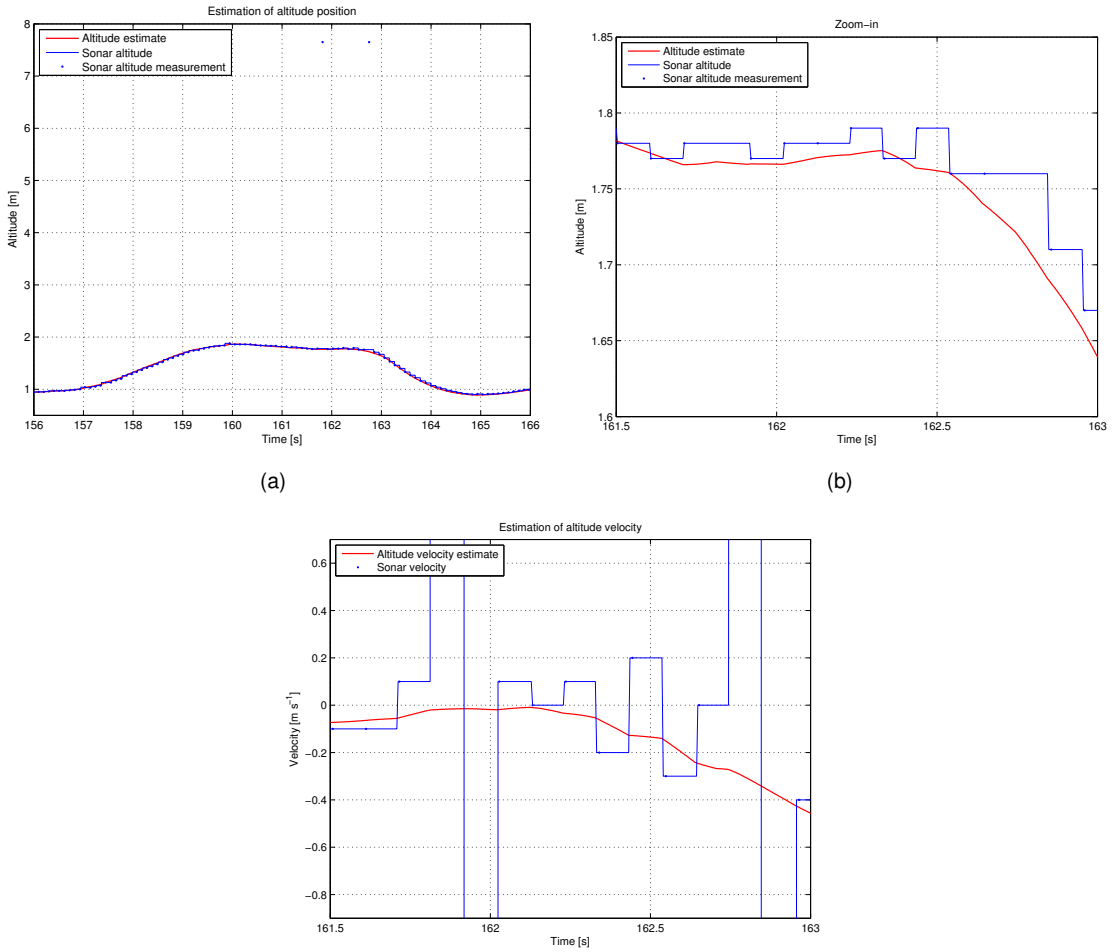
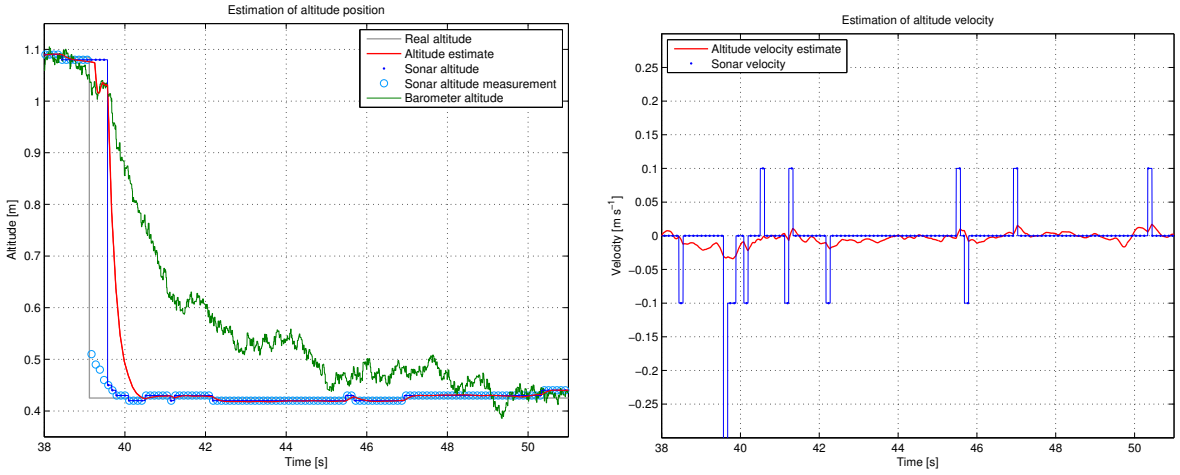


Figure 4.6: Figure (a) shows the estimation of altitude position with sonar measurement spikes at $t = 161.8$s and $t = 162.7$s. A zoom-in of figure (a) is displayed on figure (b), which shows that the position estimate is not influenced by sonar measurement spikes. Figure (c) shows the estimation of altitude velocity at the same time, confirming that the spikes have no effect on the velocity estimate.

new ground level is accepted.

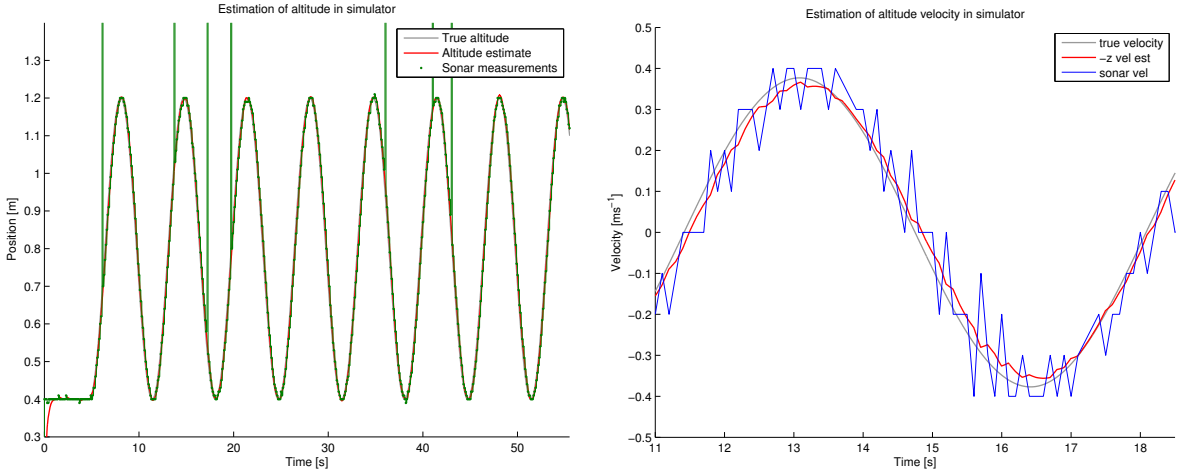
An experiment is performed in the simulator where the robot follows a sinusoidal altitude reference, and is shown in Figure 4.8. The simulator used is presented in Section 3.1.3.

Figure 4.9(a) shows the plot of the position estimate's error along the experiment in the simulator presented in Figure 4.8. The error is very low, with a Standard Deviation (SD) of σ=0.012241m and maximum error below 0.03m, excluding the initial phase where the algorithm is converging to the initial state ($t \in [0, 0.5]$s). The coefficient of determination R^2 of the altitude estimate in relation to the real estimate is 0.9957, which indicates a very good correspondence between the observed position and reality. Theoretically, the residuals of the EKF estimation should be white noise, which would mean that the observer model fits correctly the real system. Since the sensor measurements have delays, the state is observed with a delay, which means that estimates will always be delayed comparing to the real system, because sensor measurements cannot be predicted ahead of time. This delay will create a bias on the residuals that changes with state variations, which is the sinusoidal component that can be easily spotted on Figure 4.9(a). If the state of the system does not change its value ($t \in [1.5, 4.5]$s), then the sensor is measuring the correct value. In this case, the sinusoidal component does not exist, which



(a) (b)

Figure 4.7: In this experiment, the hexacopter is held manually at a altitude of 1.08m and then it is put over a 65cm tall table at $t = 39.1$s, always maintaining its absolute height to the ground floor. Figure (a) shows the estimation of altitude position, while figure (b) shows the estimation of altitude velocity. The sonar measurements are rejected as outliers until a new ground level is accepted, at $t = 39.6$s. Before the new ground level is accepted, the altitude starts being corrected with barometer measurements, at $t = 39.25$s and start being corrected only by the sonar again at $t = 39.6$s.

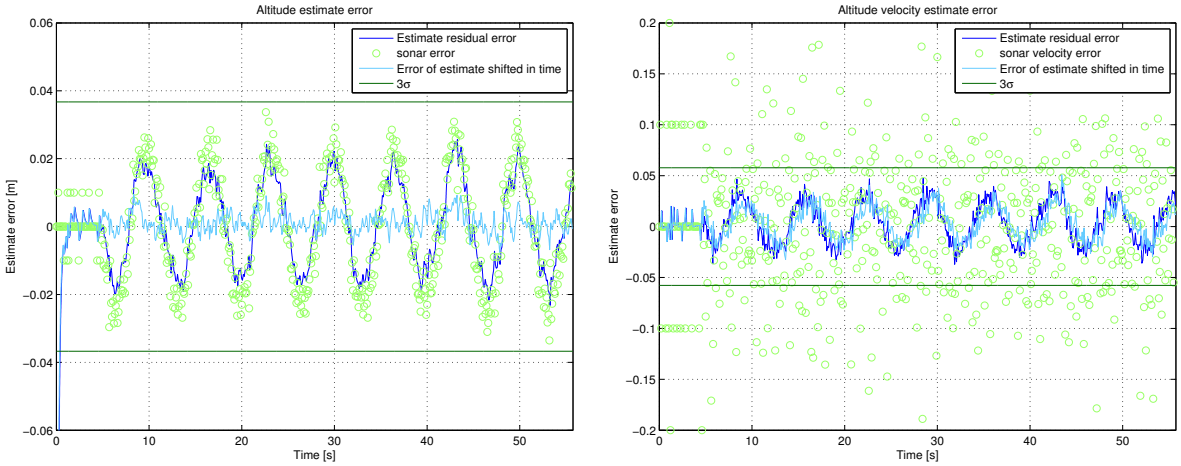


(a) (b)

Figure 4.8: In this simulation, the hexacopter is following a sinusoidal pattern. The position estimation is shown in Figure (a), whereas Figure (b) shows the velocity estimates.

means that the residuals are only white Gaussian noise. In order to prove that the sinusoidal component of the error is explained only by the delay in the estimation (which originates from delayed sensor information), the estimates were shifted by 0.068s (in order to eliminate the delay of the estimation). It can be observed on Figure 4.9(a) that the error of the shifted signal follows a normal distribution with zero mean and SD smaller than 1cm (σ=0.0028m), which means that the state is observed correctly, only delayed by 0.068s. The error of the shifted signal can also be seen on Figure 4.9(a). It is important to notice that this delay (0.068s) is smaller than one sonar sampling period (0.1s), which means that the EKF is significantly predicting the sonar measurements using accelerometer measurements.

Figure 4.9(b) presents the velocity estimate's error of the same experiment as Figure 4.9(a). The error of the estimate (maximum error below 0.05m.s^{-1} and σ=0.019258m.s^{-1}) is considerably lower than



(a) (b)

Figure 4.9: Figure (a) presents the error analysis of altitude estimation using the altitude EKF. One can observe a sinusoidal pattern that arises from the delay of the estimates comparing to the real state. The error of the altitude estimate shifted by 0.068s is also presented to evidence the normal distribution of the observed state if there were no delay on the estimate. Figure (b) shows the error analysis of altitude velocity estimation. As it happens for the position, the error becomes normally distributed if the delay is removed, as it is shown by shifting the velocity estimate 0.068s over time.

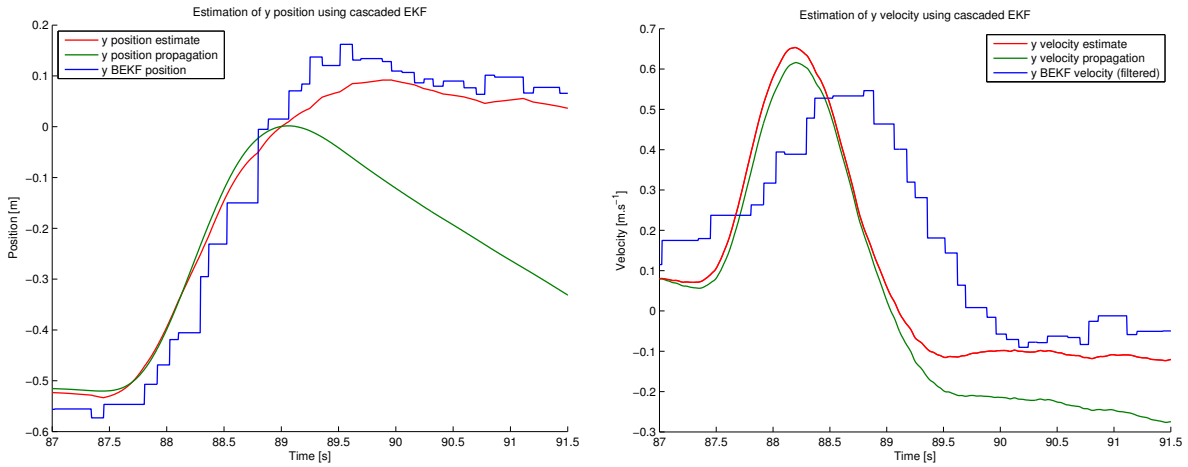
of the estimates received from the sonar (maximum error can go up to 70m.s^{-1} and σ=0.067175m.s^{-1}). The sinusoidal pattern observed on the estimate's error does not disappear by shifting the signal in time, as can be observed in Figure 4.9(b). This varying bias is explained by a small scaling factor between velocity estimate and true velocity; when the velocity is high, the estimation error grows. In simulation, this error was verified as being between 3-4$\%$ of the true velocity, thereby increasing in absolute value with higher velocities. The R^2 coefficient of fitting the altitude velocity estimate to the ground truth is 0.99301.

4.3.2 xy Estimate

The implementation described in Section 3.3 is adopted as a solution to the localization issue. The results of the cascaded EKF are discussed, where the output of the BEKF is shown as being laser-based estimates and the output of the PEKF is presented as the state estimates that are fed to the controllers.

Even though the BEKF position estimates have a small delay, a good estimation of position is accomplished, as one can see from Figure 4.10(a). Here, it can be spotted a 0.1s shift between the curves of the PEKF positon estimate and BEKF estimate right after t=87.5s, meaning that the PEKF position estimates are ahead of time comparing to the BEKF delayed estimates, as it was desired. The delay of the BEKF estimates compared to the state propagation is measured as 0.1s in this experiment, as we consider the state propagation to have virtually zero delay.

The velocity estimates computed from position estimates received from the BEKF give innacurate measurements in terms of scale, as seen in Figure 4.10(b); filtering the signal is the main cause. To minimize the impact, we saturate the correction that is applied in the update step of the PEKF; this makes the correction have a low effect when the estimate is too different from the state. On the other

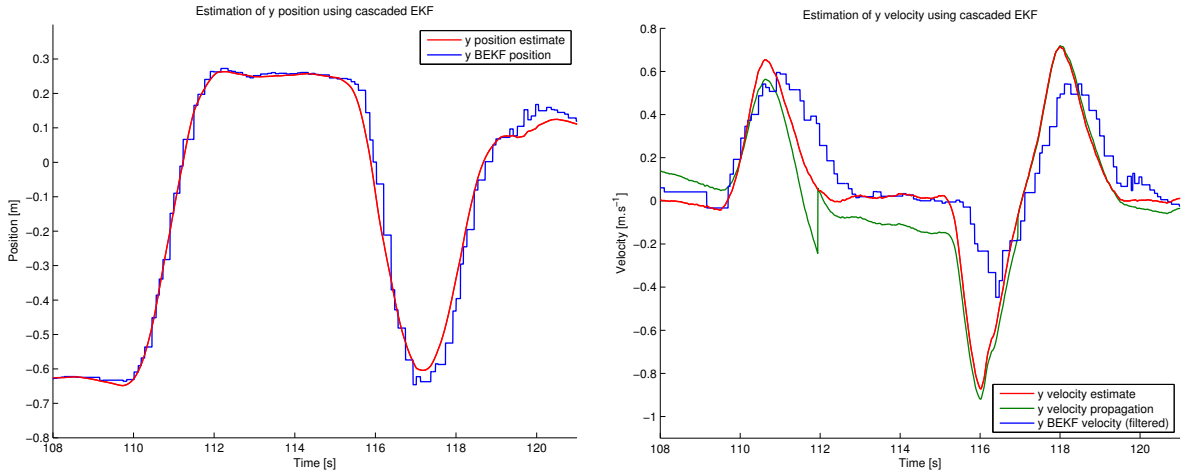


(a) (b)

Figure 4.10: Figure (a) and (b) show the estimation of position and velocity, respectively, along the y dimension. The BEKF estimates have a small delay varying between 0.1s and 0.15s. The state propagation is shown for the reader to have a notion of when the accelerometer detects changes of motion, but no conclusion should be drawn in terms of its absolute values. This experiment is done holding the hexacopter by hand.

hand, the velocity computed from BEKF estimates is very good at giving estimates when the velocity is low. The main role of this computed velocity estimates is to correct the bias accumulated in the state propagation, especially when the velocity is low.

The velocity state propagation shows discontinuities every 5s, on Figure 4.11(b), which exist on purpose. This signal is only shown to give an idea of the motion variations given by the accelerometer. The state propagation that is done in the EKF is constantly being corrected, while the signal plotted is what the state propagation would look like if it was not corrected. No conclusion can be made from its absolute value at each instant of time.

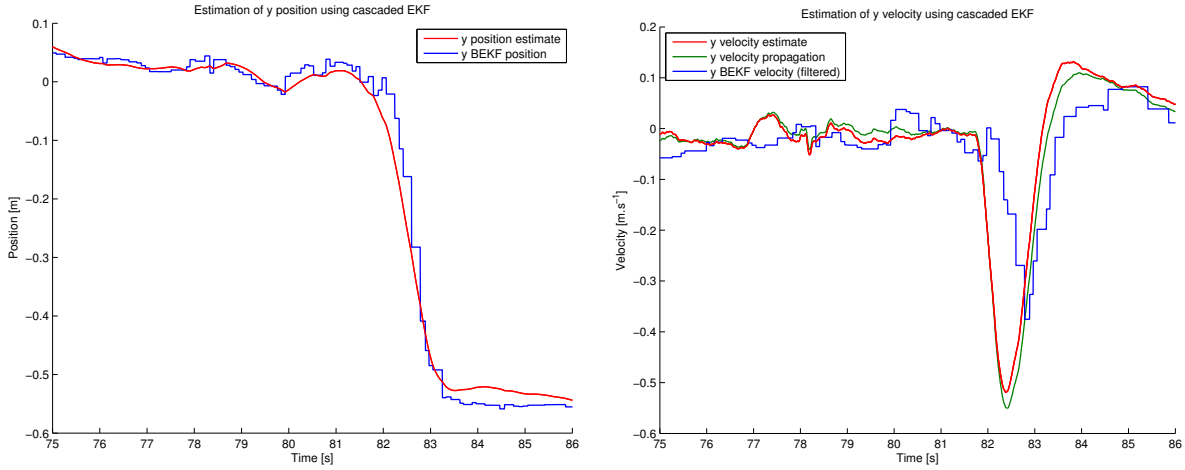


(a) (b)

Figure 4.11: Figure (a) and (b) show the estimation of position and velocity, respectively, along the y dimension. The velocity state propagation is always drifting, so the plot shows it being updated to the estimate's value every 5s (in $t = 112$s an $t = 117$s). This experiment is done holding the hexacopter by hand.

The Kalman Gain associated to the accelerometer bias was incorrectly set too big. Sometimes the effect of over-correcting the bias can be noticed, as happens in Figure 4.12(b). After the robot moves,

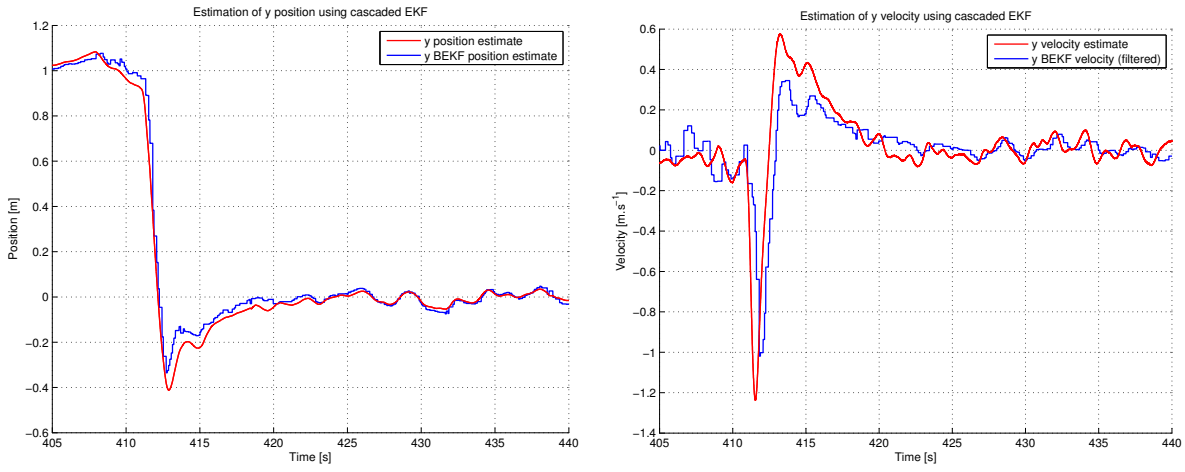
the integration of acceleration gives a positive velocity, which leads to a small oscillation of the position estimate after $t = 83.5$s. Then the position slowly converges to the correct value: -0.55m, as the bias is corrected and the velocity converges to the real value: 0m.s^{-1}. A similar situation occurs in Figure 4.11 at t=119s, where the accelerometer bias can be spotted, since the velocity propagation drifts to a negative velocity, leading to an oscillation in the position estimate that is then corrected.



(a) (b)

Figure 4.12: Figure (a) and (b) show the effect of over-correcting the accelerometer bias in the estimation of position and velocity, respectively. At $t = 83.5$s, the velocity should return to zero, but positive bias in the acceleration creates a positive bias in the velocity, leading to a position estimate that starts to grow at $t = 83.5$s starting when it should still be decreasing. A small oscillation in the position estimate can be identified after $t = 83.5$s. This experiment is done holding the hexacopter by hand.

In the experiment shown in Figure 4.13, the hexacopter is sent from position $y = 0$m to position $y = 1$m. It is not possible to draw conclusions regarding the performance of the estimator. Both the overshoot and the small oscillatory motion around the setpoint are observed with the naked eye; the overshoot has its origin on the controller response.

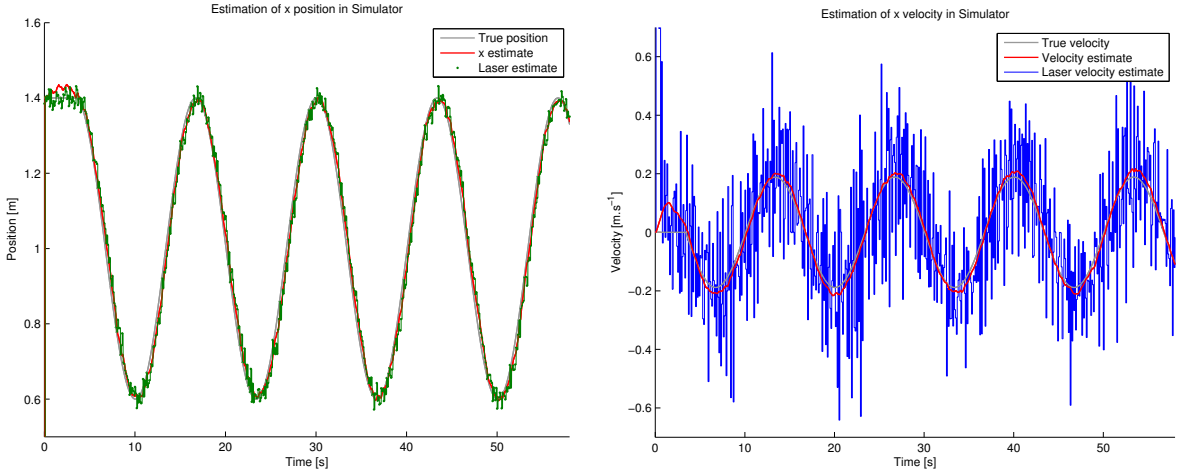


(a) (b)

Figure 4.13: Figure (a) and (b) show the estimation of position and velocity, respectively, in the y dimension, when the hexacopter is sent from position $y = 0$m to the setpoint $y = 1$m. Overshoot can be identified in response.

Figure 4.14 shows an experiment performed in the simulator, where the robot follows a sinusoidal

position reference along the x axis. The simulator used is presented in Section 3.1.3.



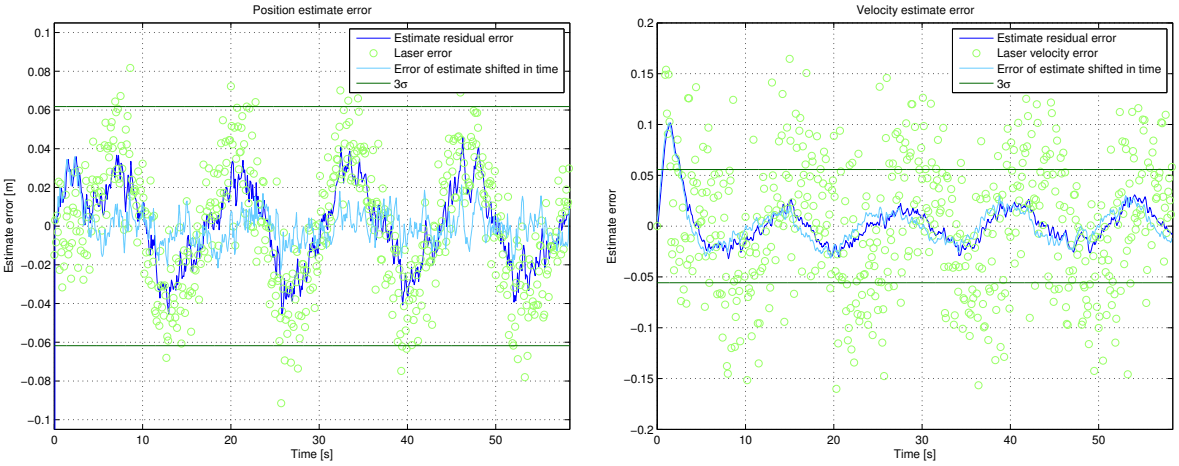
(a) (b)

Figure 4.14: In this simulation, the hexacopter is following a sinusoidal pattern along the x axis. The position estimation is shown in Figure (a), whereas Figure (b) shows the velocity estimates.

An error analysis is performed with the results collected from the experiment in the simulator presented in Figure 4.14. Figure 4.15(a) shows the residuals of the position estimates. The error is never bigger than 0.05m and has a SD of σ=0.020597. The sinusoidal pattern that the residuals exhibit is mitigated if the signal is shifted 0.14s, where the error becomes a normally distributed with variance σ^2=0.088^2. This shows that the estimate is correctly estimated, but is delayed 0.14s, which originates on the laser-based estimates having a delay of 0.2s.

The error associated to the velocity estimation during the experiment shown in Figure 4.14 is presented in Figure 4.15(b). The error is smaller than 0.04m.s^{-1}, with a SD of σ=0.018586. For the velocity estimate case, shifting it on time will not lead to a normally distributed error, because there is a scaling factor between the true velocity and the estimated one. Our aproach for correcting the velocity only uses laser-based velocity estimates from the past, $i.e.$, the velocity is corrected with a certain delay. This leads to small scaling errors that are created by propagating the state with the acceleration integration. Nevertheless, the error is very small in value, as intended, even with the noise model of the laser-based estimates increased in simulation. The laser velocity estimates have a residual with $\sigma = 0.063274$. The overall fit of the velocity estimates to the true velocity curve has coefficient of determination R$^2 = 0.97054$.

The experimental accuracy of the algorithm cannot be verified due to the lack of ground truth. Unlike the altitude case, we do not have a sensor of which we are certain of its accuracy and that measures the position (or velocity) of the robot directly. From the error analysis of the algorithm, we can conclude that the is estimator is able to provide very good estimates with noisy measurements, whose noise follows a gaussian distribution. In reality, the accuracy of the state estimation depends on the accuracy of the BEKF position estimates. Hence, the validity of the error analysis performed lies on the accuracy of the model designed in Section 3.1.3 to mimick the BEKF estimates. In motionless scenarios, we can verify that the BEKF provides accurate position estimates, using a measuring tape. The delay of the estimates is measured against the state propagation curve, which is used as a reference of zero delay. The delay of the estimate is close to zero both in position and velocity. For position, the delay is bigger because



(a) (b)

Figure 4.15: Figure (a) presents the error analysis of x position estimation using the altitude EKF. The delay of the estimates creates a bias that changes with the variation of the position of the robot; here this bias is observed as being sinusoidal. If the delay is removed, the error becomes normally distributed, as it is shown by shifting the position estimate 0.14s over time and observing that the associated error. Figure (b) presents error analysis of the x velocity estimation using the altitude EKF. Unlike in the position case, delay on the estimates is not the explanation for the sinusoidal behaviour of the residuals. Instead, it is a scaling factor that is creating errors that depend on the velocity value.

the estimates are corrected with a bigger weight by the BEKF estimates; on the velocity estimates, the delay is almost zero, since they are mostly the state propagation itself, hence we achieve a delay lower than our goal of 0.02s.

4.4 Conclusion

The original solution for estimating altitude position and velocity are not adequate for indoor settings. It becomes impossible to control the MAV autonomously without absolute localization. The solution developed in this project proves to be better than the original implementation at estimating position and velocity and also bettwer than the sonar sensor by itself. The position estimates are able to predict what the next sonar measurement will be, creating a smooth estimation. The solution developed meets the proposed goals in Section 3.2.2 in terms of sampling frequency, position accuracy, position and velocity estimation delay, and resistance to sensor failure. The accuracy in velocity measurements cannot be experimentally verified due to the lack of a direct source of measurement, but, if the simulation assumptions are correct, then the velocity accuracy also meets the defined goal.

Even though it is possible to use Optical Flow for navigation of MAVs, it only allows for very short flights, since the optical flow based pose estimation is affected by slow drift due to observing only the relative velocity of features [56]. It is a problem shared with visual odometry based implementations, where the motion is estimated by considering feature displacements between images [57]. The limitations of this sensor, discussed in Section 4.2.2, do not allow to reach the goals of this project, especially due to the impossibility of absolute localization.

Self-localization is achieved using the solution proposed. Experiments show that good results are obtained even with the BEKF position estimates arriving at a varying frequency. Since the walls of

our testbed do not touch the ceiling, the LRF often reads noise when the hexacopter is tilted, due to the lack of wall. This proves that the solution can withstand at least 2 consecutive measurements that only contain noise. Having people around the MAV does not impact the pose estimation, since the line extraction algorithm used, RANSAC, spontaneously does outlier rejection of small curved features. Even if incorrect lines are extracted, they are not likely to be matched to any feature in the map.

The solution implemented does not meet all the proposed goals in Section 3.3.2. The sampling frequency and velocity estimation delay requirements were met. The position estimation delay has been proven experimentally, as one can observe in Figure 4.10(a), but not in simulation. The worse result in simulation can come from the fact that the model for the BEKF position estimates considered in simulation is substantially more pessimistic than the real estimates; this can make the BEKF estimates have a bigger impact in state correction in simulation than they do in reality, which leads to transferring a bigger delay to the state estimation of position than what happens in reality. The position and velocity estimation accuracy cannot be proven due to the lack of a ground truth source. In a static situation, the position estimates have a high accuracy smaller than 2cm, measured rudimentary with a measuring tape; even when the robot is moved 5cm in between consecutive iterations of the BEKF, the convergence is quick enough to satisfy this accuracy. It is not possible to verify any accuracy value during flight, since there is no ground truth. In simulation, the goals set for accuracy are met with a big margin.

The Optical Flow sensor can be used to complement our proposed approach for x and y localization, not only for improving the velocity estimation, but also for the stabilization of the MAV in case of LRF outages or failure of the BEKF in detecting features on the environment. Implementing this fault tolerant approach would certainly improve the robustness of the proposed solution. The merge of the two approaches can be done, since the PEKF is designed to accept other position and/or velocity estimates as measurement inputs.

Chapter 5

Conclusions

This chapter presents the discussion of how satisfying the results of this project are in light of the main goals that were proposed. The main achievements are presented in Section 5.1 and the improvements that can be done regarding this work can be found in Section 5.2. The latter Section also poses the work developed as a foundation that can be expanded; here, new directions for future work are suggested.

5.1 Achievements

The work developed in this thesis has successfully met the proposed goals. The full study of our physical system has been addressed theoretically and the approaches used towards low computational self-localization have been presented with satisfactory results. The successful integration of the two software systems, Mobotware and PX4 autopilot, lead to a self-contained full control system that is able to autonomously execute a straight line path defined by any user. DTU's software was expanded to interact with the Pixhawk, which enabled to take advantage of features already existing in both systems.

The two criteria considered in the design of the localization algorithms were accuracy and speed. Having good error models of the sensors allowed us to create certain assumptions in the design of the localization algorithms. We assume that the process noise and measurement noise covariance matrices converge to steady-state values; we then assume larger values for the noise parameters in our model, to ensure that the state always converges, taking the risk of having a slower convergence. If the sensors do not follow the error model that correspond to the Kalman Filter design, the measurements are rejected, as happens with certain noisy measurements of the ultrasonic sensor. Using these assumptions allows us to decrease the complexity of the EKFs greatly, consequently speeding up the computations, since the computational load of aided navigation systems is typically dominated by the time update of the covariance matrix [45]. In this project, the linearization of the dynamic model is done considering a constant acceleration, which is a valid assumption when using a high sampling frequency.

The altitude localization algorithm is fault tolerant, as proposed, while the localization in the x and y dimensions is designed to be easily expandable into a fault tolerant approach, where adding an Optical Flow sensor is considered as viable and easy option. Both localization algorithms can withstand outlier

measurements from the sensors. On the laser range measurements, this is of particular importance, since it is expected that people walk around the robot in an indoor setting. This ability is verified experimentally, and the algorithm proves to maintain localization even with continuous noisy measurements on the laser. The localization in x and y dimensions can withstand laser range measurements that only contain noise in at least 2 consecutive iterations without diverging to incoherent results. The estimation algorithms perform online self-calibration of the barometer and accelerometer sensors. The altitude estimation EKF is able to detect ground level changes bigger than 20cm.

The cascaded approach of the localization in the x and y dimensions allows for an easy expansion, as already mentioned, but also for an easy change of self-localization algorithm. The BEKF can be easily changed to test the performance of other algorithms, using the same cascaded architecture.

The problem about the cascaded approach is that the estimates computed by one Kalman Filter cannot be modeled correctly as containing only white Gaussian noise. Since these estimates are used as measurement inputs in another KF, this raises the question whether having a cascaded KF is a viable option, since a KF should only receive measurements with white Gaussian noise. In this thesis, the estimates from one Kalman Filter are assumed as being the true value plus additive zero mean Gaussian noise and the results obtained are positive.

An approach to deal with delayed measurements in an EKF is successfully implemented in x and y velocity estimation. The result of this approach lead to estimation with virtually zero delay, while having very small impact on the accuracy of the estimate, as proved in simulation. This could not be proved experimentally due to the lack of ground truth for velocity.

All the proposed goals for the localization algorithms, stated in Sections 3.2.2 and 3.3.2, were met. Some could not be verified experimentally, as discussed in Section 4.4, but were verified in a simulation environment where the conditions were defined to be worse than reality, as explained in Section 3.1.3.

The semi autonomous navigation has been implemented successfully as a path execution approach, but should be used carefully. The system will have unexpected behavior if it is sent to an area that does not meet the requirements of the localization algorithm. The MAV should only be sent to areas in the known environment, since it will not be able to estimate its position without an adequate map. The limitations of the localization algorithm, presented in Section 2.7, are discussed in Section 3.3.5; these limitations should be taken into account when using the navigation solution. Our system can work accurately with a map that contains only 2 features, which is the minimum amount required, and that was the setup used in all the experiments.

The final system was achieved using only on-board devices to estimate, control and process information, as proposed initially. All the laser-based localization computation is done in a single board computer that is at the lower price range in the market. This represents a great achievement in this project.

The hexacopter is able to stay hovering in a confined circular space of 60cm radius, which is a small space considering its large radius: 44cm. The size of this bounded space should be different if another MAV is considered. The goal of navigating through doors (width of a regular door is close to 80cm) was never proposed along this project, since the hexacopter cannot be moved manually through a door without carefully rotating it into a certain angle.

The system is able to be dragged to another place and it will return by itself to the setpoint it was ordered to in the first place. Besides the ability to withstand pushes in any direction, it also supports rotations around any axis, without loosing stability. The behavior against constant forces actuating in the system, such as constant wind gusts, was not studied, since disturbances are considered to be transient in an indoor environment.

This system is designed for a navigation with a smooth behavior. Since the localization algorithms are currently dependent on having correct LRF measurements, the system should not have big attitude values (bigger than 0.35rad) for periods longer than 0.3s, since a big attitude may compromise the measurements. We did not try to perform aggressive maneuvers with the UAV; such is not advised.

5.2 Future Work

Even though positive experimental results were achieved in this project, there are many topics to be improved. Besides improving already implemented subjects, this project can be viewed as a self-localizing autonomous platform that can be used in a future application or for future research.

The localization in the x and y dimensions has been simplified in order to get lower computational demand. Nevertheless, one improvement that should be emphasized is the need to receive the error covariance associated to the pose estimates computed in the BEKF. These covariance values should be used in the PEKF ideally to compute adequate Kalman Gains. Simpler solutions can be to reject pose estimates that do not prove to be informative enough. This may contribute to the BEKF estimates not deteriorating the PEKF position and velocity estimates, but can also be used as a warning for the Pixhawk to enter fail-safe mode, in the case where the BEKF is not able to localize itself for several consecutive iterations.

The position controllers should be better tuned and possibly modified, in order to achieve a smoother response. This was not studied due to lack of time. The current response shows some overshoot and the hovering area is small, but still bigger than desired. Additionally, the vertical position controller shows difficulties tracking altitude position: the response is slow (order of seconds) when the position error is small (smaller than 10cm).

A command to enable/disable the Offboard mode via the computer is implemented. Once this mode is enabled via the computer, it has to be disabled in the same way. Even though this approach is correct, it is less safe than manually doing it using the RC, thus removing this feature should be considered. This concern arises from the fact that if there is a problem while the Offboard mode is enabled, then the user can disable this mode faster using a switch on the RC than by writing a command in the computer.

The robustness of the estimation can be increased by adding other algorithms that can provide estimates to the PEKF, which would work in parallel with the BEKF. The algorithms that can be added may use the same sensor information, such as computing motion estimates from consecutive laser range readings [16], [5]. The algorithms can also be based on new additional sensors, such as the already discussed Optical Flow sensor, PX4FLOW [54], or camera for feature-based motion estimation [6], [58], [7], [8]. SLAM algorithms are not recommended for this system, unless the on-board computer

is upgraded. It should be considered algorithms that can complement the localization algorithm already implemented, ideally compensating for its limitations. The PX4FLOW sensor has an advantage against the LRF when the indoor environment has a big open space, which does not have obstacles that the laser sensor can detect; in this case, the optical flow sensor can provide estimates until the MAV is able to return to an area with obstacles, where the robot can perform self-localization again.

Safety is a topic that should be addressed in more depth in the future. A fault tolerant approach that can withstand sensor failure for at least a small period of time, in the x and y estimates, should be developed, as it was developed for the z estimates using the barometer as an emergency sensor. Such approach can be easily added to the current solution by appending an Optical Flow sensor. The Pixhawk's EKF is implemented in such a way that adding the latter sensor is simple. As seen on Section 4.2.2, the PX4FLOW sensor can provide accurate velocity estimates, which contributes to a better estimate of position in case failure in extracting position estimates from the laser sensor. This sensor was not added in the first place because our goal was to use a laser-based only approach.

Another safety aspect to be improved is the fact that the localization algorithm is not redundant, *i.e.*, if the laser scanner fails, we are unable to have a position estimate. The position will only be propagated for 1 second after the BeagleBone stops sending laser-based position estimates to the Pixhawk. After that, the system was designed so that the Pixhawk denies being able to estimate x and y position and velocity, consequently entering fail-safe mode. If the hexacopter is hovering in the same position at the moment the laser failure happens, there will not be any problem, as the accelerometer will be able to propagate the position well enough in this case. But if the hexacopter is moving at a considerable velocity, unexpected behavior might occur during that 1 second. Another safety issue is the laser scanner being blocked by something or someone. This can lead to incorrect laser-based position estimates, leading to unexpected behavior. It is safe to rely on the laser being able to visualize partially two walls, since all the experiments were performed in this condition. So it is safe to say that if the system is fed with a map of the environment that has information over the whole 360 degrees, there should not be any problem of blocking several angular areas of the sight of the LRF.

This project serves as a basis that can be expanded into a more robust navigation solution. Now that the system is able to localize and execute a defined path autonomously, the navigation solution can be improved to avoid obstacles, in order to increase safeness. Trajectory planning can be added, leaving the role of the user even more futile. Implementations of path planning with obstacle avoidance are already available [59], [60], [26], [61], [62]. Other approaches can be followed using our existing case, where a starting map is given by the user, which is then modified and expanded online, as the robot flies, to match the real environment more accurately.

The work developed here can also be used as a testing platform. Due to the cascaded approach adopted for the localization in the x and y dimensions, the BEKF localization algorithm can easily be substituted by another approach. This allows to fulfill a performance study of several approaches. Furthermore, instead of replacing algorithms, other sensors or estimators can be added in parallel to the BEKF, *i.e.*, feeding their information to the PEKF and, possibly, improving the accuracy of the state estimates.

Bibliography

[1] S. Carrascosa and E. Escorsa. Hovering over the drone patent landscape, 2014.

[2] E. Altug, J. Ostrowski, and C. Taylor. Quadrotor control using dual camera visual feedback. In *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, volume 3, pages 4294–4299 vol.3, Sept 2003. doi: 10.1109/ROBOT.2003.1242264.

[3] S. Lupashin, A. Schollig, M. Sherback, and R. D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1642–1648, May 2010. doi: 10.1109/ROBOT.2010.5509452.

[4] G. P. Tournier, M. Valenti, J. P. How, and E. Feron. Estimation and control of a quadrotor vehicle using monocular vision and moir patterns. In *In AIAA Guidance, Navigation and Control Conference*, pages 2006–6711. AIAA, 2006.

[5] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Stereo Vision and Laser Odometry for Autonomous Helicopters in GPS-denied Indoor Environments. In *Proceedings of SPIE*, page 7332. SPIE, 2010. URL `http://dspace.mit.edu/handle/1721.1/52660`.

[6] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart. Onboard imu and monocular vision based control for mavs in unknown in- and outdoor environments. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3056–3063, May 2011. doi: 10.1109/ICRA.2011.5980343.

[7] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *Computer Vision (ICCV), 2013 IEEE International Conference on*, pages 1449–1456, Dec 2013. doi: 10.1109/ICCV.2013.183.

[8] K. Schauwecker, N. Ke, S. Scherer, and A. Zell. Markerless visual control of a quad-rotor micro aerial vehicle by means of on-board stereo processing. In P. Levi, O. Zweigle, K. Häußermann, and B. Eckstein, editors, *Autonomous Mobile Systems 2012*, Informatik aktuell, pages 11–20. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-32216-7. doi: 10.1007/978-3-642-32217-4_2. URL `http://dx.doi.org/10.1007/978-3-642-32217-4_2`.

[9] R. Brockers, M. Hummenberger, S. Weiss, and L. Matthies. Towards autonomous navigation of miniature uav. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 645–651, June 2014. doi: 10.1109/CVPRW.2014.98.

[10] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. The pixhawk open-source computer vision framework for mavs. In *ICRA'11*, 2011.

[11] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *ICRA'11*, pages 2992–2997, 2011.

[12] R. Siegwart and I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004. ISBN 026219502X.

[13] I. Cox and J. Kruskal. On the congruence of noisy images to line segment models. In *Computer Vision., Second International Conference on*, pages 252–258, Dec 1988. doi: 10.1109/CCV.1988.589996.

[14] P. Jensfelt. Approaches to mobile robot localization in indoor environments. Technical report, PhD thesis, Signal, Sensors and Systems (S3), Royal Institute of Technology, SE-100 44, 2001.

[15] R. Y. Brogaard. Control of multi rotor helicopter, 2012.

[16] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 935–938, Jun 1994. doi: 10.1109/CVPR.1994.323928.

[17] M. Bryson and S. Sukkarieh. Vehicle model aided inertial navigation for a uav using low-cost sensors, 2006.

[18] L. Armesto, S. Chroust, M. Vincze, and J. Tornero. Multi-rate fusion with vision and inertial sensors. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 1, pages 193–199 Vol.1, April 2004. doi: 10.1109/ROBOT.2004.1307150.

[19] P. Corke. An inertial and visual sensing system for a small autonomous helicopter. *Journal of Robotic Systems*, 21(2):43–51, 2004. ISSN 1097-4563. doi: 10.1002/rob.10127. URL `http://dx.doi.org/10.1002/rob.10127`.

[20] H. Rehbinder and B. Ghosh. Pose estimation using line-based dynamic vision and inertial sensors. *Automatic Control, IEEE Transactions on*, 48(2):186–199, Feb 2003. ISSN 0018-9286. doi: 10.1109/TAC.2002.808464.

[21] T. Cheviron, T. Hamel, R. Mahony, and G. Baldwin. Robust nonlinear fusion of inertial and visual data for position, velocity and attitude estimation of uav. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 2010–2016, April 2007. doi: 10.1109/ROBOT.2007.363617.

[22] S. Weiss, M. W. Achtelik, S. Lynen, M. C. Achtelik, L. Kneip, M. Chli, and R. Siegwart. Monocular vision for long-term micro aerial vehicle state estimation: A compendium. *Journal of Field Robotics*, 30(5):803–831, 2013. ISSN 1556-4967. doi: 10.1002/rob.21466. URL `http://dx.doi.org/10.1002/rob.21466`.

[23] A. Kosaka and A. Kak. Fast vision-guided mobile robot navigation using model-based reasoning and prediction of uncertainties. In *Intelligent Robots and Systems, 1992., Proceedings of the 1992 IEEE/RSJ International Conference on*, volume 3, pages 2177–2186, Jul 1992. doi: 10.1109/IROS.1992.602076.

[24] K. E. Wenzel, P. Rosset, and A. Zell. Low-cost visual tracking of a landing place and hovering flight control with a microcontroller. In *Proceedings of the UAV'09, the 2nd International Symposium on Unmanned Aerial Vehicles*, Reno, Nevada, USA, June 8-10 2009. URL http://www.ra.cs.uni-tuebingen.de/publikationen/2009/wenzel2009_uav_wiitracking.pdf.

[25] M. Blösch, S. Weiss, D. Scaramuzza, and R. Siegwart. Vision based mav navigation in unknown and unstructured environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 21–28, May 2010. doi: 10.1109/ROBOT.2010.5509920.

[26] S. Grzonka, G. Grisetti, and W. Burgard. A fully autonomous indoor quadrotor. *Robotics, IEEE Transactions on*, 28(1):90–100, Feb 2012. ISSN 1552-3098. doi: 10.1109/TRO.2011.2162999.

[27] S. Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 20–25, May 2011. doi: 10.1109/ICRA.2011.5980357.

[28] S. Weiss, R. Brockers, and L. Matthies. 4dof drift free navigation using inertial cues and optical flow. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 4180–4186, Nov 2013. doi: 10.1109/IROS.2013.6696955.

[29] R. Brockers, S. Susca, D. Zhu, and L. Matthies. Fully self-contained vision-aided navigation and landing of a micro air vehicle independent from external sensor inputs. volume 8387, pages 83870Q–83870Q–10, 2012. doi: 10.1117/12.919278. URL http://dx.doi.org/10.1117/12.919278.

[30] G. Chowdhary, D. M. Sobers, C. Pravitra, C. Christmann, A. Wu, H. Hashimoto, C. Ong, R. Kalghatgi, and E. N. Johnson. Self-contained autonomous indoor flight with ranging sensor navigation. *Journal of Guidance, Control, and Dynamics*, 35(6):1843–1854, 2015/10/04 2012. doi: 10.2514/1.55410. URL http://dx.doi.org/10.2514/1.55410.

[31] I. J. Cox. Blanche-an experiment in guidance and navigation of an autonomous robot vehicle. *Robotics and Automation, IEEE Transactions on*, 7(2):193–204, Apr 1991. ISSN 1042-296X. doi: 10.1109/70.75902.

[32] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29:5–28, 1998.

[33] J.-S. Gutmann and C. Schlegel. Amos: comparison of scan matching approaches for self-localization in indoor environments. In *Advanced Mobile Robot, 1996., Proceedings of the First Euromicro Workshop on*, pages 61–67, Oct 1996. doi: 10.1109/EURBOT.1996.551882.

[34] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN 0262201623.

[35] D. C. K. Yuen and B. A. Macdonald. A comparison between extended kalman filtering and sequential monte carlo techniques for simultaneous localisation and map-building. In *In Proceedings of the 2002 Australasian Conference on Robotics and Automation*, 2002.

[36] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart. A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 1929–1934, Aug 2005. doi: 10.1109/IROS.2005.1545234.

[37] G. Borges and M.-J. Aldon. Line extraction in 2d range images for mobile robotics. *Journal of Intelligent and Robotic Systems*, 40(3):267–297, 2004. ISSN 0921-0296. doi: 10.1023/B:JINT.0000038945.55712.65. URL http://dx.doi.org/10.1023/B%3AJINT.0000038945.55712.65.

[38] A. Harati and R. Siegwart. A new approach to segmentation of 2d range scans into linear regions. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2083–2088, Oct 2007. doi: 10.1109/IROS.2007.4399499.

[39] R. Vázquez-Martín, P. Núñez, A. Bandera, and F. Sandoval. Curvature-based environment description for robot navigation using laser range sensors. *Sensors (Basel, Switzerland)*, 9(8):5894–5918, 2009. doi: 10.3390/s90805894. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3315112/.

[40] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat. Comparing icp variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013. ISSN 0929-5593. doi: 10.1007/s10514-013-9327-2. URL http://dx.doi.org/10.1007/s10514-013-9327-2.

[41] J. G. Leishman. *Principles of helicopter aerodynamics*. Cambridge aerospace series. Cambridge University Press, Cambridge, New York, 2000. ISBN 0-521-66060-2. URL http://opac.inria.fr/record=b1096782.

[42] R. M. Murray, S. S. Sastry, and L. Zexiang. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1994. ISBN 0849379814.

[43] S. Bouabdallah. Design and control of quadrotors with application to autonomous flying, 2007.

[44] A. Beck. Robot hardware abstraction layer, 2008.

[45] J. A. Farrell. *Aided Navigation - GPS with High Rate Sensors*. McGraw-Hill Education, 2008.

[46] A. M. Sabatini and V. Genovese. A stochastic approach to noise modeling for barometric altimeters. *Sensors (ISSN 1424-8220)*, November 2013.

[47] V. Bistrovs and A. Kluga. Adaptive extended kalman filter for aided inertial navigation system. *Elektronika Ir Elektrotechnika (ISSN 1392 ? 1215)*, 2012.

[48] J. González-Jiménez, A. Stentz, and A. Ollero. A mobile robot iconic position estimator using a radial laser scanner. *Journal of Intelligent Robotic Systems*, 13:161–179, 1995.

[49] R. Needham. *Calibration of Strapdown System Accelerometer Dynamic Errors*. Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, 1994. URL `https://books.google.pt/books?id=rnaeNwAACAAJ`.

[50] E. W. Nettleton and H. F. Durrant Whyte. Delayed and asequent data in decentralized sensing networks. volume 4571, pages 1–9. SPIE, 2001. doi: 10.1117/12.444148. URL `http://dx.doi.org/10.1117/12.444148`.

[51] D. Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006. ISBN 0471708585.

[52] T. Larsen, N. Andersen, O. Ravn, and N. Poulsen. Incorporation of time delayed measurements in a discrete-time kalman filter. In *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*, volume 4, pages 3972–3977 vol.4, Dec 1998. doi: 10.1109/CDC.1998.761918.

[53] N. Gageik, M. Strohmeier, and S. Montenegro. An autonomous uav with an optical flow sensor for positioning and navigation. *International Journal of Advanced Robotic Systems*, 2013. doi: 10.5772/56813.

[54] D. Honegger, L. Meier, P. Tanskanen, and M. Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *ICRA*, pages 1736–1741. IEEE, 2013. ISBN 978-1-4673-5641-1. URL `http://dblp.uni-trier.de/db/conf/icra/icra2013.html#HoneggerMTP13`.

[55] F. Kendoul, I. Fantoni, and K. Nonami. Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles. *Robotics and Autonomous Systems*, 57(6-7):591–602, 2009. URL `http://dblp.uni-trier.de/db/journals/ras/ras57.html#KendoulFN09`.

[56] J.-C. Zufferey and D. Floreano. Fly-inspired visual steering of an ultralight indoor aircraft. *Robotics, IEEE Transactions on*, 22(1):137–146, Feb 2006. ISSN 1552-3098. doi: 10.1109/TRO.2005.858857.

[57] S. Fowers, D.-J. Lee, B. Tippetts, K. Lillywhite, A. Dennis, and J. Archibald. Vision aided stabilization and the development of a quad-rotor micro uav. In *Computational Intelligence in Robotics and Automation, 2007. CIRA 2007. International Symposium on*, pages 143–148, June 2007. doi: 10.1109/CIRA.2007.382886.

[58] D. L. Johansen, D. L. Johansen, D. Clark, N. Taylor, D. Timothy, W. Mclain, B. Y. University, A. R. Parkinson, and D. L. Johansen. Abstract video stabilization and target localization using feature tracking with small uav video, 2006.

[59] J. B. Saunders, O. Call, A. Curtis, A. W. Beard, and T. W. Mclain. Static and dynamic obstacle avoidance in miniature air vehicles," in aiaa infotech at aerospace, 2005, paper no. In *in Proceedings of the Infotech@Aerospace Conference*, pages 2005–6950, 2005.

[60] S. Hrabar. 3d path planning and stereo-based obstacle avoidance for rotorcraft uavs. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 807–814, Sept 2008. doi: 10.1109/IROS.2008.4650775.

[61] J. Amin, J. Boškovic, and R. Mehra. *A Fast and Efficient Approach to Path Planning for Unmanned Vehicles*. American Institute of Aeronautics and Astronautics, 2015/10/04 2006. doi: doi:10.2514/6.2006-6103. URL `http://dx.doi.org/10.2514/6.2006-6103`.

[62] L. Heng, L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. Autonomous obstacle avoidance and maneuvering on a vision-guided mav using on-board processing. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2472–2477, May 2011. doi: 10.1109/ICRA.2011.5980095.

[63] Pixhawk. Connecting a companion computer, 2015. URL `https://pixhawk.org/dev/companion_link`.

Appendix A

Mobotware

Mobotware is DTU's real-time control software. It is used in this project and modified to the project's needs. The user should have read Section 2.5.2 before reading this Appendix, in order to have a basic understanding Mobotware. In this appendix, the details of the programs that compose Mobotware are presented, directed at a user that intends to have a deeper understanding of the software, possibly to use the implementation developed in this project.

The procedure to run mobotware on the BeagleBone from a ground-based computer is presented in Appendix A.5.

A.1 Robot Hardware Daemon

Robot Hardware Daemon (RHD) is a real-time synchronized database that contains all the relevant variables that are shared between programs, such as the position of the robot. As explained in Section 2.5.2, its running frequency is defined by the developer, and defines the frequency of the database synchronization and update, as well as the update of new information received/sent from/to the Pixhawk.

The database is divided into two tables: Write and Read tables. The variables in each table are defined in the initialization stage. Figure A.1 shows the interaction of MRC with the RHD database tables. An RHD plugin can write values from variables on the Read table and they can read variables from the Write table. Other applications, such as MRC, can write on the Write table but only read from the Read table. This ensures that the flow of information is uni-directional. As one can apprehend, the names of the tables are client-oriented, where the client in this case is the MRC.

A.2 Mobile Robot Control

Mobile Robot Control (MRC) is a low-level application, running in soft real-time, that is mainly used as a bridge for communication between user, RHD and control/estimation programs.

There were several approaches taken to deal with MRC, depending on the needs of the project at a certain time. The final approach uses a Telnet client to control MRC. All the needed functionalities

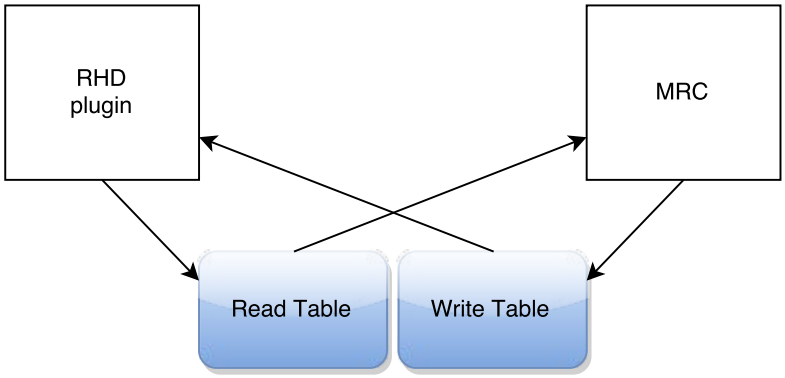


Figure A.1: The communication flow is uni-directional, as shown by the arrows. The client, in this case MRC, writes to the RHD on the Write table and and reads from the Read table. The RHD plugin does the analogous, on the other direction. The RHD Database is created and initialized by an RHD plugin, in our case this plugin is the MavlinkComm.

are achieved by a series of commands that can be found in Appendix A.3.1 and are explained in Section 2.5.2. In this implementation, a *plan*[1] is used in order to do two tasks at the same time: 1) enable MRC to update a few RHD variables autonomously and continuously; 2) allow MRC to be accessible to the user remotely, from a Telnet client. By using the Telnet client, the user is able to communicate with Mobotware, by using MRC to change variables in the RHD database. By changing RHD variable values, the user can start several actions related to controlling the robot. These actions will be explained in Section A.3.

MRC requires a configuration file located at *mobotware-3.648/mrc/trunk/mrc/hexacopter/calib/robot.conf*. After sending the plan through the Telnet client, MRC starts a periodic call to run the localization algorithm - *aulocalize* - on the laser scanner server ULMS. MRC is responsible for getting information to and from the ULMS server regarding the robot's pose.

A.3 Telnet Client

As mentioned in Section A.2, we use a Telnet client to enter a *plan* in MRC and also to enable the user to change values of RHD variables. The latter corresponds to sending commands to the Pixhawk, which allows to control the position of the hexacopter. By sending these commands, we can override the Remote Control (RC) and control the robot's pose from the ground-based computer.

The commands that have to be written when we start the Telnet client are found in Appendix A.3.1. These will declare local variables and run MRC's *plan*. After sending these initialization commands, MRC is fully operational and the user is now able to enter commands to control the robot's position. The robot's position can only be controlled from the computer if the Pixhawk is set to Offboard Mode. This mode can be enabled in two ways: 1) by a physical switch on the Remote Control, or 2) through the telnet client with the command 1, followed by command 7, listed in Table A.1. It is advised to change the mode using the RC, because it is faster to turn the mode on and off.

[1] A *plan* is equivalent to a script, but can be defined in run-time and will run in background, leaving MRC responsive to other jobs.

To order the robot to go to a specific setpoint in the Pixhawk coordinate frame (PF), the setpoint should first be set using commands 2-5, listed in Table A.1. For safety issues, x, y and z setpoints have to be less than 1.5m away from the robot's current position along each axis, otherwise the setpoint will be rejected. This ensures that the hexacopter does not attempt to fly away due to a user mistake after the user enters the setpoint. In order to start sending setpoints to the Pixhawk, the variable *sendsetpoint* should be set to 1, using command 6 listed in A.1. Notice that the PF is a North-East-Down (NED) frame, hence the value given to the z setpoint should be negative. If we set the value of the z setpoint to be positive, the vehicle will land and stay on the ground, still with Offboard mode enabled.

When attempting to land using Offboard mode, it is advised to firstly send a z setpoint of -250mm (equivalent to hovering at a small distance (25cm) to the ground) and only then send a positive setpoint, such as 500mm. This procedure provides a slow effective landing. Giving a big positive z setpoint, such as 500mm (50cm underground), ensures that the hexacopter will stay on the ground with the propellers at minimum thrust. After the robot is standing still on the ground, the main mode switch in the RC should be assigned to Manual and the throttle RC stick should be set to the minimum position (all the way down). Only then should Offboard mode be disabled. This procedure is the correct method to land the hexacopter autonomously.

In order for the written commands to take effect, *offboardcommand* has to be set to 1, by using the command 7 listed in Table A.1. The variable *offboardcommand* is used as a confirmation to send a set of orders: every time the user wishes to send an order to the Pixhawk, he should set the variable *offboardcommand* to 1. This ensures that even though a mistake might have been made while writing a command, it will have no effect until this confirmation step is executed, giving the user a chance to change a currently unconfirmed order.

It is advised to disable the Offboard mode only when the main mode switch on the RC is set to POSCTL mode. This ensures that when disabling the Offboard mode, the robot will be in POSCTL where it will remain hovering in the same position. The only exception to this advice is in the case of landing the vehicle in Offboard mode, mentioned on the previous paragraphs.

#	Command	Argument Range	Units	Argument Description
1	setvar "offboardmode"	[0..1]	-	0: set mode off 1: set mode on
2	setvar "setpointx"	$]x^{WF}$-1500 , x^{WF}+1500[	mm	Update x position setpoint
3	setvar "setpointy"	$]y^{WF}$-1500 , y^{WF}+1500[	mm	Update y position setpoint
4	setvar "setpointz"	$]z^{WF}$-1500 , z^{WF}+1500[	mm	Update z position setpoint
5	setvar "setpointyaw"	[-2$\pi\times$ 10^3 , 2$\pi\times$ 10^3]	rad$\times$ 10^3	Update yaw angle setpoint
6	setvar "sendsetpoint"	[0..1]	-	0: do not send any setpoint 1: send x, y, z and yaw setpoints
7	setvar "offboardcommand"	[1]	-	1: send commands 1 to 6

Table A.1: Telnet commands available to the user, related with the control of the hexacopter's position. These orders will be sent to the Pixhawk. The command should be written by the user in the Telnet client terminal in the following way: Command [white space] Argument .

A.3.1 Telnet Client messages to MRC

The following commands should be introduced to the MRC by writing them in the Telnet Client. They create and initialize variables in the MRC and also contain the procedure to create and execute the *plan* explained in the previous Section.

```
# Create local variables
offboardmode = 0
sendsetpoint = 0
setpointx = 0
setpointy = 0
setpointz = 0
setpointyaw = 0
offboardcommand = 0

# Create MRC plan
beginplan A

plan set "$gyro1gain" 1
plan oldxoff = 0
plan oldyoff = 0
plan oldthetaoff = 0
plan true = 1
plan laser "push t=0.11 cmd='localize silent=true'"

plan label "send"
plan offboardcommand = 0
plan setoutput "offboardmode" offboardmode
plan setoutput "sendsetpoint" sendsetpoint
plan setoutput "setpoint[0]" setpointx
plan setoutput "setpoint[1]" setpointy
plan setoutput "setpoint[2]" setpointz
plan setoutput "setpoint[3]" setpointyaw
plan goto "start"

plan label "start"
plan wait 0.001

plan xoff = $l0 * 1000
```

```
plan yoff = $l1 * 1000
plan thetaoff = $l2 * 1000
plan if(offboardcommand == true) "send"
plan setoutput "xoff" xoff
plan setoutput "yoff" yoff
plan setoutput "thetaoff" thetaoff
plan goto "start"

endplan A

# Run MRC plan
# Note: send this command separately, after sending previous commands
runplan A
```

A.4 MavlinkComm

Establishing a MAVLINK connection is dependent on the hardware used in the connection itself. Three approaches were taken:

- USB: is a fast way to test the program, but unadvised during flight due to possible data corruption and unexpected behavior [63].
- FTDI: can be used to connect the serial port on the Pixhawk's end to the USB port in the BeagleBone's end. A cable was built using a 3.3V FTDI converter cable.
- UART: can be used directly, since both BeagleBone and Pixhawk have 3.3V logic and BeagleBone has pins that can be defined in order to have several UART connections. UART1 port (*ttyO1*) was enabled.

The final approach taken was a UART-UART connection, which is preferred over the FTDI (UART-USB) for enabling a faster transmission. The selected port should be defined in *rhdconfig.mavlinkcomm.xml* configuration file.

MAVLINK can operate at different baudrates (9600, 57600, 115200, 230400, 460800 or 921600), which needs to be defined both at Pixhawk's end and at MavlinkComm. Initially, the highest baudrate was chosen, but the final choice was 57600, which ensured that all the information was sent at the correct frequencies and reduced the number of *blank messages*[2] received at the BeagleBone's side. If more information is sent or topics are sent at higher frequencies, it is recommended to use a higher baudrate. The selected communication baudrate should be defined in *rhdconfig.mavlinkcomm.xml* configuration file.

[2]MavlinkComm often receives packages that do not contain any MAVLINK messages. These packages are detected, read and, after decoded, it is revealed that they do not contain any information. These packets do not correspond to corrupted packages, since all the information that should be received is being correctly received at approximately the right time.

A.5 Running mobotware from a ground-based computer

The order to startup Mobotware is given from a ground station computer. The computer has to be able to access the BeagleBone and have user rights to establish an *ssh*[3] connection. The computer can access the BeableBone through its mini-USB port (IP address: 192.168.7.2), Ethernet through the existing switch (IP address: 192.168.1.3), Ethernet using the DTU's network (IP address: 10.59.8.136, which may change) or through a direct Ethernet connection, where DHCP is turned off and a static IP address is configured (in our case, it is configured as 192.168.1.3).

In order to simplify Mobotware's startup, several scripts were created. Each mobotware application has its own script: RHD, RHDTEST, ULMS and MRC. These scripts start each application with its individual requirements and they are located in the BeagleBone in */mobotware-3.648/run_mobotware/name_of_app*, where *name_of_app* should be replaced by the name of each application.

The main script that is in charge of running every other script is in the ground computer and is in charge of creating an *ssh* session for each mobotware app, where the corresponding script will run. Several versions were made depending if real-time plot of results were intended and on which programs to run. The simplest version is presented in the Section that follows.

A.5.1 Mobotware startup script

This script should be run from a ground-based computer that can remotely access the BeagleBone on the hexacopter. The IP address should be selected according to the connection established with the BeagleBone.

This script uses a custom bash function called newtab, which is not relevant to this project, whose only purpose is to open a new tab in the OSX terminal app.

Notice that when sending a command via ssh, an ssh session needs to be created with –t argument in order to force resources allocation (such as tty), thus enabling Mobotware apps to function correctly.

```
#!/bin/bash

# This script is using the newtab function defined in bash profile

    source ~/.bash_profile
    newtab -G ssh -t root@192.168.1.3 'bash ./mobotware-3.648/
run_mobotware/rhd/runRHD.sh'
    sleep 1
    newtab -G ssh -t root@192.168.1.3 'bash ./mobotware-3.648/
run_mobotware/rhd/runRHDTEST.sh'
```

[3] Secure SHell is an encrypted network protocol. It allows to establish a secure channel to a machine over an insecure network and access it remotely.

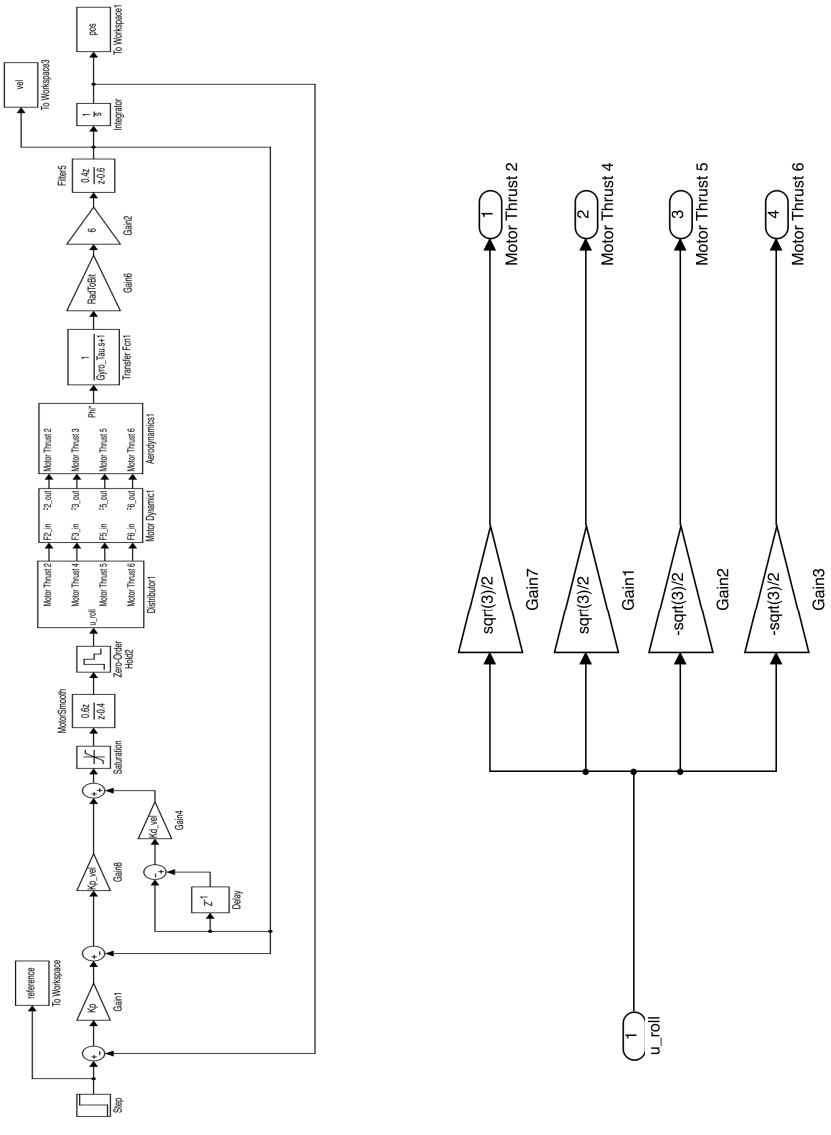
```
    sleep 1
    newtab -G ssh -t root@192.168.1.3 'bash ./mobotware-3.648/
run_mobotware/ulmsserver/runULMS.sh'
    sleep 3
    newtab -G ssh -t root@192.168.1.3 'bash ./mobotware-3.648/
run_mobotware/mrc/runMRC.sh'
    sleep 1
    newtab -G ssh -t root@192.168.1.3 'telnet localhost 31001'

echo "Scripts are running on BeagleBone through SSH."
```

Appendix B

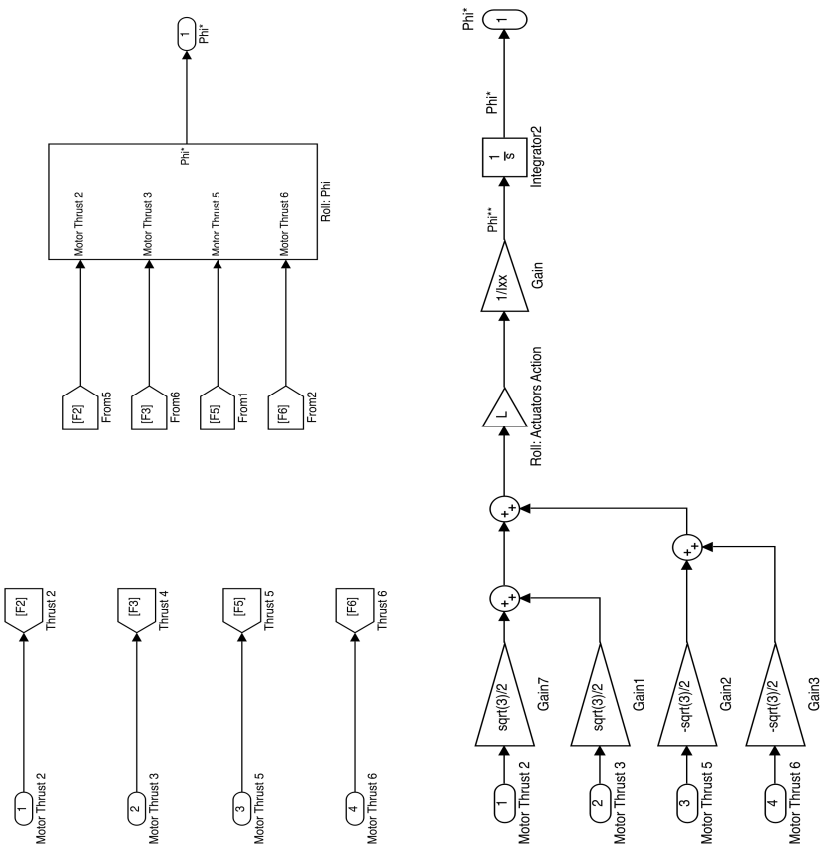
Simulink Controller Models

B.1 Roll Controller



(a) (b)

Figure B.1: Figure (a) shows the roll cascaded controller, while Figure (b) shows the Motor Distributor block.



(a) (b)

Figure B.2: Figure (a) shows the Aerodynamics blocks, while Figure (b) shows a block inside the Aerodynamics block.

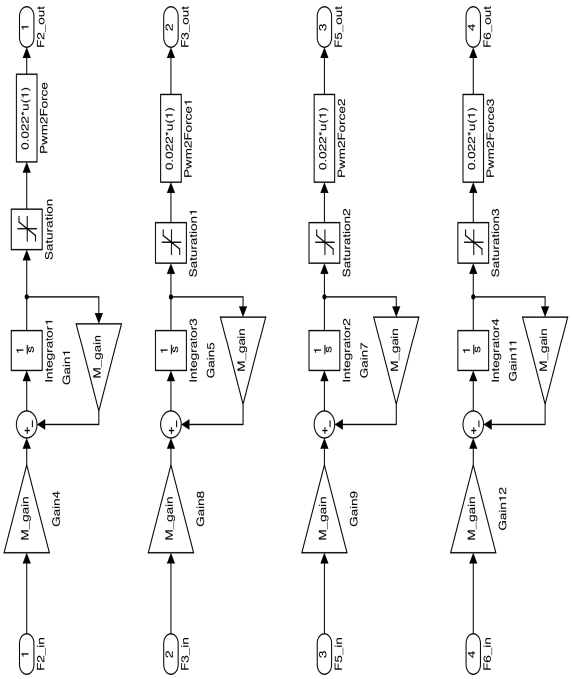
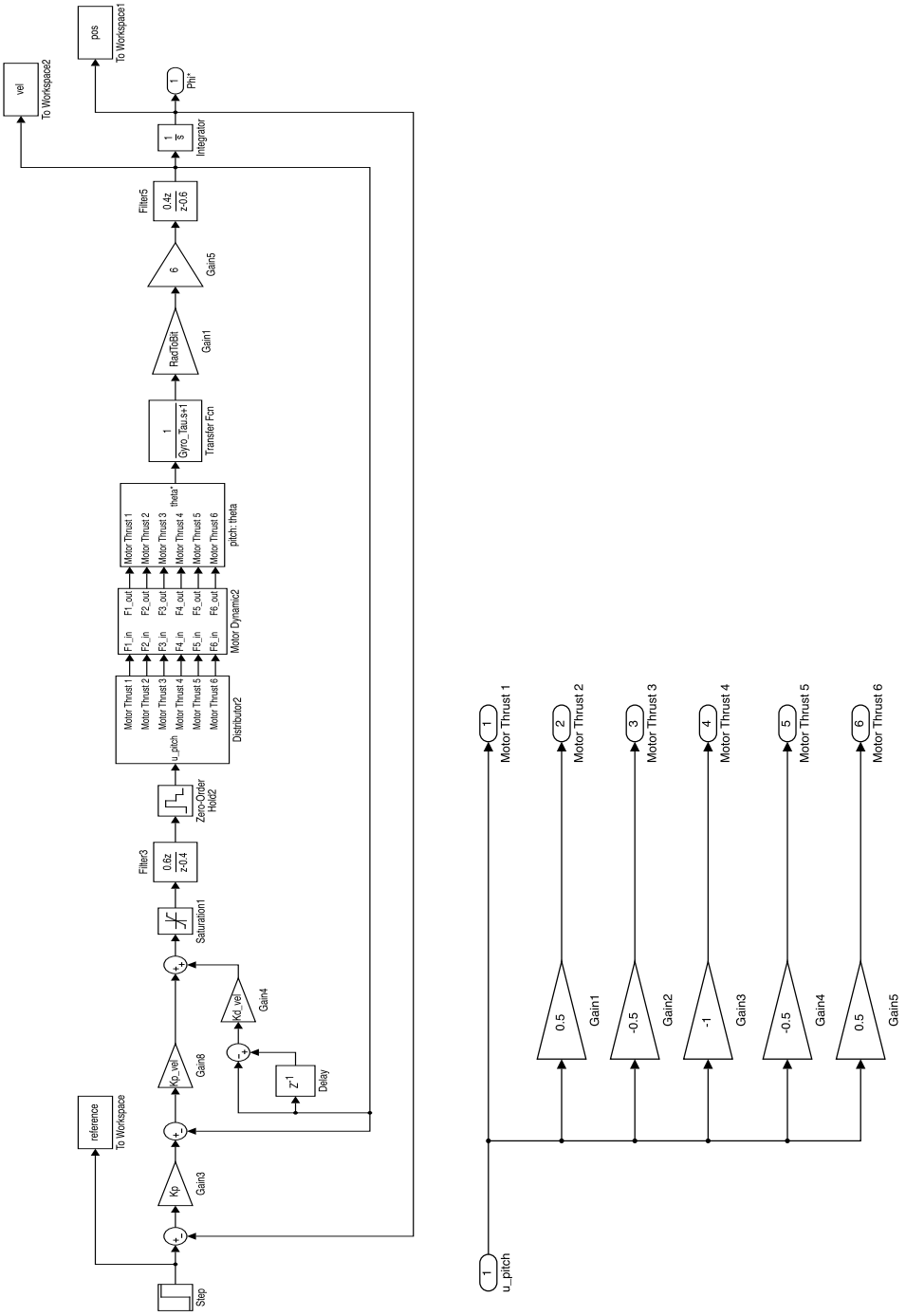


Figure B.3: Motor Dynamics block.

B.2 Pitch Controller



(a) (b)

Figure B.4: Figure (a) shows the Pitch cascaded controller; Figure (b) shows the Motor Distributor block.

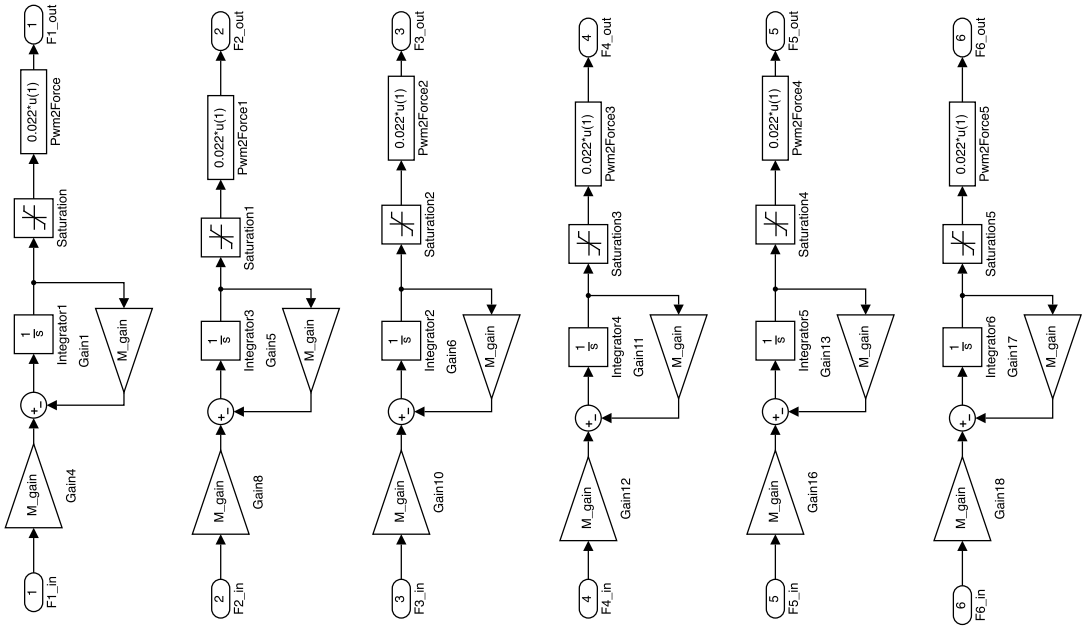


Figure B.5: Motor Dynamics block.

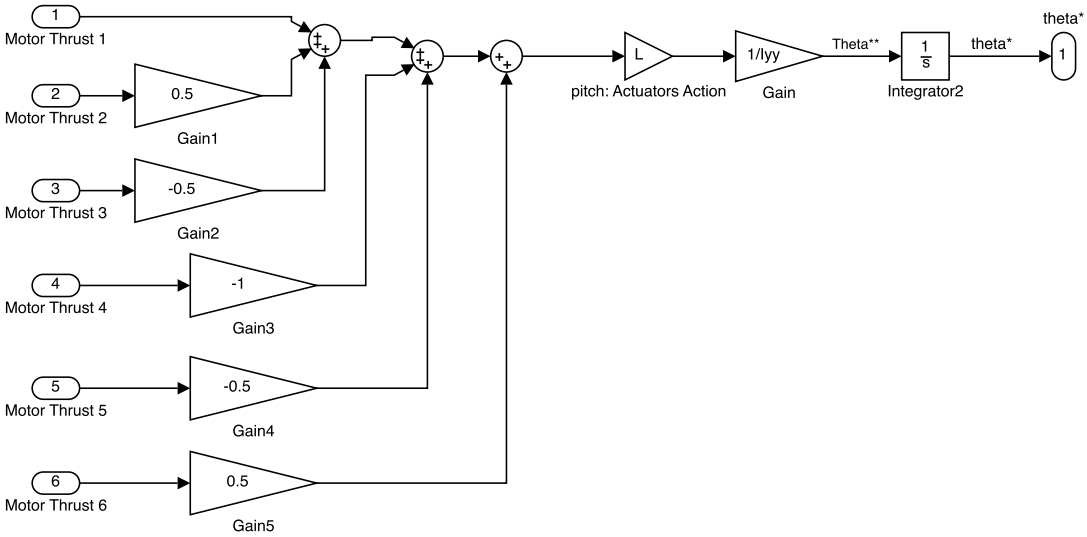


Figure B.6: Aerodynamics block.

Appendix C

PX4 Control Loop

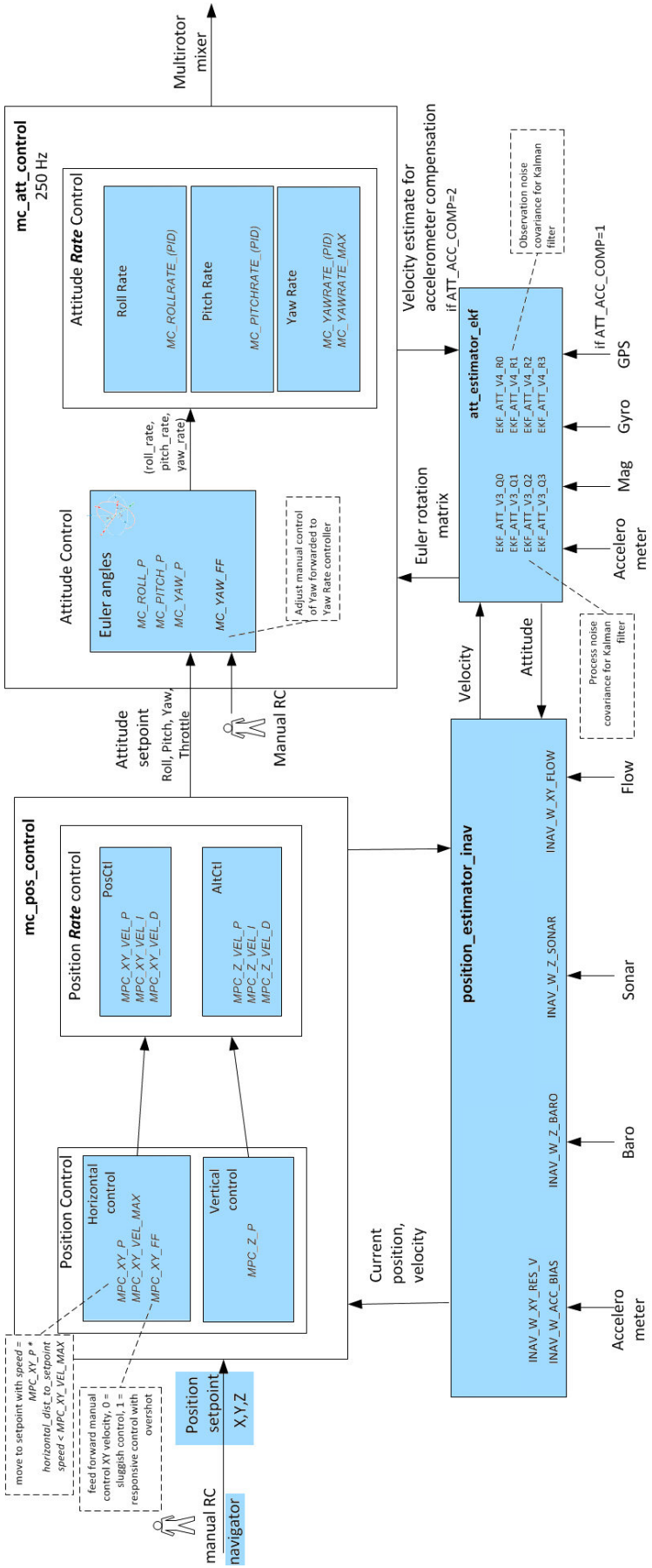


Figure C.1: PX4 control loop modules, where the fields highlighted in blue are the ones changed in this project. (Image based on source[2])

[2] https://pixhawk.org/dev/multirotor/start