POLITECNICO DI MILANO
Corso di Laurea Magistrale in Computer Engeneering
Dipartimento di Elettronica, Informazione e Bioingegneria

AN APPLICATION OF THE EXTENDED KALMAN FILTER TO THE ATTITUDE CONTROL OF A QUADROTOR

Advisor: Prof. Matteo MATTEUCCI
Co-Advisor: Dott. Andrea ROMANONI
Co-Advisor: Prof. Marco LOVERA

Master thesis by:
Leonardo ASCORTI, ID 745919

Academic year 2012-2013

Contents

List of Figures

List of Tables

Acknowledgements

I am very grateful to my advisor, prof. Matteo Matteucci, who supported me through all the stages of the development of my work, not only with great competence, but also with remarkable courtesy and patience. I would also like to acknowledge my two co-advisors, dott. Andrea Romanoni and expecially prof. Marco Lovera, his knowledge and experience in aerospace engeneering was fundamental for this thesis.

A special thank goes to dott. Marco Bergamasco, who built the quadrotor that inspired this research during his PhD course.

Abstract

The focus of this thesis is the application of the extended Kalman filter to the attitude control system of a four-propellers unmanned aerial vehicle usually known as quadrotor.

The Kalman filter is a mathematical tool well suited for an algorithmic implementation that estimates the state of a dynamic system influenced by random noise given a set of measurements which are also corrupted by random noise. If the system and measurement eqautions are linear functions of the state variables and the noises are both normally distributed, the filter can be proven to be an optimal estimator. In practice, the linearity conditions are often not satisfyed, but some rielaborations of the filter algorithm have been used for more than fourty years in many nonlinear applications with good results. The extended Kalman filter (EKF), used in this thesis, is one of the first and best known nonlinear filter versions.

A quadrotor is a helicopter lifted and propelled by four rotors. Small sized quadrotors are often used as UAVs (unamanned aerial vehicles) in research and amateur projects, because of the simple symmetric structure and relatively easy control law with respect to traditional helicopters. In this thesis, the extended Kalman filter is applied to estimate the state of the quadrotor from the noisy measurements of on board low-cost MEMS sensors. The estimated state is intended to be used by a control algorithm (not discussed in this work) to maintain the desired attitude during various maneouvers.

The EKF is implemented in Simulink in both continuous and discrete time, as an extension of a pre-existing model, which simulates the dynamics and control of a quadrotor. The performance and stability of the system is then analyzed with many test cases, from simple hovering to complex trajectories, both with open and closed loop control. The model is also tested for robustness in case of errors in the measurements of physical parameters and incorrect sensor calibration.

Estratto in lingua italiana

Lo scopo di questa tesi è l'applicazione del filtro esteso di Kalman (abbreviato EKF) al sistema di controllo dell'assetto di un elicottero quadrirotore.

Il filtro di Kalman è uno strumento matematico, facilmente applicabile in forma di algoritmo, per stimare lo stato di un sistema dinamico perturbato da rumore sulla base di un insieme di misure anch'esse corrotte da rumore. Il filtro in pratica è in grado di compensare i due tipi di errore per ottenere una stima migliore di quella che potrebbe essere ottenuta conoscendo solo il valore delle misure o il modello del sistema dinamico. Nel caso in cui sia il sistema dinamico sia le equazioni che descrivono le misure sono funzioni lineari delle variabili di stato ed entrambi i rumori sono generati secondo distribuzioni gaussiane, è possibile dimostrare che il filtro di Kalman è uno stimatore ottimo. I sistemi reali difficilmente soddisfano queste condizioni, in particolare per quanto riguarda la linearità delle equazioni, tuttavia la ricerca in questo campo ha sviluppato varianti del filtro originale che forniscono risultati soddisfacenti (sebbene non ottimi in senso stretto) anche quando applicati a sistemi non lineari. In particolare, in questa tesi è utilizzato il filtro di Kalman esteso, uno dei primi algoritmi non lineari che negli anni è stato applicato a con successo in diversi scenari, tra cui la guida automatica di veicoli ha una particolare rilevanza.

I quadrirotori sono un tipo di velivolo senza pilota (UAV) a decollo e atterraggio verticale (VTOL) che ha suscitato notevole interesse tra i ricercatori negli ultimi anni, perché le leggi che ne governano il volo sono relativamente semplici rispetto a quelle degli elicotteri propriamente detti e anche la loro realizzazione fisica è considerevolmente meno complessa, dato che essi non hanno parti meccaniche come pale inclinabili o flap, ma si manovrano esclusivamente variando la velocitá dei quattro rotori e presentano comunque una buona agilitá di movimento. Tra le varie aree di ricerca che coninvolgono questi velivoli, ha particolare importanza lo sviluppo di algoritmi e leggi di controllo, sia per l'assetto sia per la navigazione.

In questa tesi, il filtro esteso di Kalman è impiegato per stimare lo stato del velivolo in base alle misure fornite dai sensori di tipo MEMS a basso costo presenti a bordo. Lo stato cos stimato è poi utilizzato dal sistema di controllo (il cui funzionamento non è discusso in questa sede) per mantenere l'assetto richiesto nel corso delle manovre di volo. Il modello completo del quadrirotore, dei sensori e del filtro di Kalman è implementato in Simulink e i risultati di diverse simulazioni con differenti

condizioni sono analizzati per studiarne le prestazioni, la stabilità e la robustezza agli errori.

Nel Capitolo 1 si ricava il modello matematico del sistema dinamico, sulla base di leggi fisiche e aerodinamiche. Il modello prevede uno stato del sistema composto da dieci variabili:

- le velocità lineari lungo i tre assi cartesiani nel sistema di riferimento del velivolo;
- le velocità angolari attorno ai tre assi cartesiani nel sistema di riferimento del velivolo;
- l'orientamento del velivolo rispetto ad un sistema di riferimento esterno, espresso in angoli di Eulero (rollio, beccheggio e imbardata);
- la quota rispetto al terreno.

Nello stesso capitolo si ricava anche il modello dei sensori, sulla base delle loro caratteristiche generiche. Questo modello comprende delle costanti che devono essere definite in sede di calibrazione. Da ultimo, i modelli del sistema e dei sensori sono linarizzati (calcolandone la matrice jacobiana), come richiesto dall'algoritmo EKF.

Il Capitolo 2 è dedicato alla descrizione del modello Simulink utilizzato per le simulazioni. Il modello è composto da una parte preesistente che rappresenta la dinamica del quadrirotore, l'algoritmo di controllo e il modello dei motori e da una parte realizzata nel corso di questa tesi, che modella i sensori MEMS e il filtro di Kalman. Per ragioni tecniche, il modello dei motori è stato poi escluso dal sistema utilizzato per le simulazioni. Il filtro di Kalman è implementato sia in versione a tempo continuo che a tempo discreto. La versione a tempo continuo, anche se non è implementabile su un velivolo reale, è utilizzata come termine di paragone per verificare le prestazioni del filtro discreto.

Nel Capitolo 3 sono presentati e analizzati i risultati di diverse simulazioni del modello. La prima serie di simulazioni riguarda le stime del filtro nel caso di traiettorie di volo seguite in anello aperto, cioè con il sistema di controllo che agisce sullo stato reale e non sull'uscita del filtro. In questo caso è possibile valutare le prestazioni confrontando la traiettoria stimata con quella effettivamente seguita, che è la migliore possibile data la legge di controllo. Si osserva inoltre il fatto che le prestazioni del filtro aumentano all'aumentare dell'inclinazione del campo magnetico e si fornisce una spiegazione del fenomeno. Il seguente gruppo di simulazioni contiene traiettorie in anello chiuso, cioè con il controllore operante sull'uscita del filtro. Si nota come la versione tempo continuo genera instabilità che possono essere risolte applicando un filtro passabasso, mentre la versione a tempo discreto è stabile a patto che il ritardo introdotto dal filtro anti-aliasing operante sui sensori sia sufficientemente piccolo. L'ultimo gruppo di simulazioni rappresenta il comportamento del sistema in caso di errori nell'impostazione dei numerosi parametri da cui dipende il filtro di Kalman. Si osserva che il filtro a tempo discreto è più robusto rispetto

a questi errori, ma che in ogni caso piccoli errori nell'impostazione dei parametri fisici, come la massa del velivolo, conducono a grossi errori nella stima, mentre la tolleranza è maggiore nel caso di errori nella calibrazione dei sensori.

Il Capitolo 4 trae le conclusioni del lavoro, sottolineando i buoni risultati in caso di corretta impostazione dei parametri, ma anche la scarsa robustezza nel caso di errori nella stima degli stessi. Sono presentati anche alcuni possibili sviluppi futuri, dalla correzione delle imprecisioni nel modello fino all'implementazione di un sistema di navigazione che estenda il semplice controllo dell'assetto con informazioni sulla posizione.

Per concludere, nell'Appendice A è fornita un'introduzione matematica ai filtri bayesiani (di cui il filtro di Kalman fa parte) ed è presentata la dimostrazione matematica rigorosa dell'ottimalit del filtro di Kalman sia nella versione continua, sia in quella discreta. L'Appendice B contiene invece tutto il codice Matlab utilizzato per le simulazioni, in modo che gli esperimenti presentati siano facilmente riproducibili.

Introduction

Goals

This thesis focuses on an application of the extended Kalman filter to the attitude control of a type of unmanned aerial vehicle (UAV) called quadrotor.

The filter is used to obtain an improved extimation of the attitude and speed of the aircraft by integrating the data from the sensors with the predictions of the aerodynamic model. Since the model is strongly nonlinear, the extended version of the Kalman filter has been used.

The performance and the robustness of the filter have been tested in a simulated environment modelled with Simulink

The Kalman filter

The Kalman filter is a very important discovery in the field of statistical estimation theory. It can be informally described as an optimal mathematical tool to estimate the state of a linear dynamic system perturbed by white noise by using measurements which are also linear with respect to the state and corrupted by white noise. Moreover, it is well suited for computer implementation, because it uses a finite representation of the estimation problem and a discrete step algorithm (but a continuous time version is also possibile) [13]. The main limitation of the filter to be optimal is the necessary condition that the dynamic system has to be linear. Since the first years after the Kalman's results, researchers have worked to adapt the filter algorithm to nonlinear problems, which are more common in practical applications. Two very popular nonlinear algorithms are the extended Kalman filter (EKF) [30], which is the subject of this thesis, and the much more recent unscented Kalman filter (UKF) [17, 38]. These algorithms have been experimentally proven to work well in many pratical situations, but they are not optimal and even their stability can be guaranteed only when the system meets some particular conditions that are hard to verify in practice [22, 5, 29, 34].

For all these reasons, the Kalman filter and its extended version have been successfully applied to many pratical problems in different fields, e.g., guidance and control of vehicles like ships, aircrafts and spaceships, signal processing, econometrics, etc.

Quadrotors and their applications

A quadrotor is a helicopter lifted and propelled by four rotors. The rotors of a quadcopter are usually fixed-pitch, so the aircraft is controlled by changing the relative speed of the four rotors and thus increasing or decreasing their thrust and torque. Some quadrotors have variable pitch blades, but even in this case the pitch can be changed only as a group property of the blades of each rotor and not depending on the position[1].

Small sized quadrotors are often used as UAVs (unamanned aerial vehicles) in research and amateur projects, because of the simple symmetric structure and relatively easy control law. The main advantages of quadrotors with respect to conventional helicopters are:

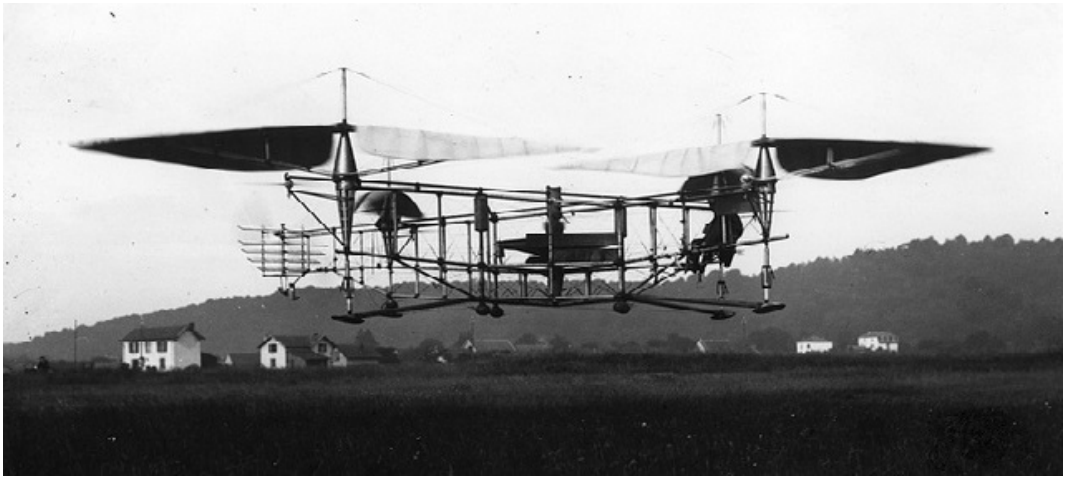
- quadrotors do not require mechanical linkages to change the pitch of the blades and they do not even use mechanically driven control surfaces like other aircraft types (flaps, rudders...), so they are much easier to design and build, expecially in small size;
- the four rotors are smaller than the single rotor of a conventional helicopter of comparable size, allowing them to posses less kinetic energy and thus cause less damage in the case of an accidental crash;
- the symmetric structure and the fact that the movements of a quadrotor depend only on the rotation rates of the propellers make them quite maneuvrable with relatively simple control systems.

The first quadrotors were designed in the 20s as manned vehicles. The prototype known as Oehmichen N°2, created by the French engineer tienne Oehmichen, was probably the first reliable VTOL (vertical take-off and landing) aircraft to be able to carry a person for at least one kilometer[2]. Another early experimental quadrotor was designed by George de Bothezat for the US army, but it had too many reliability and control problems, so the project was soon canceled [12]. This two prototypes are shown in Figure 1.

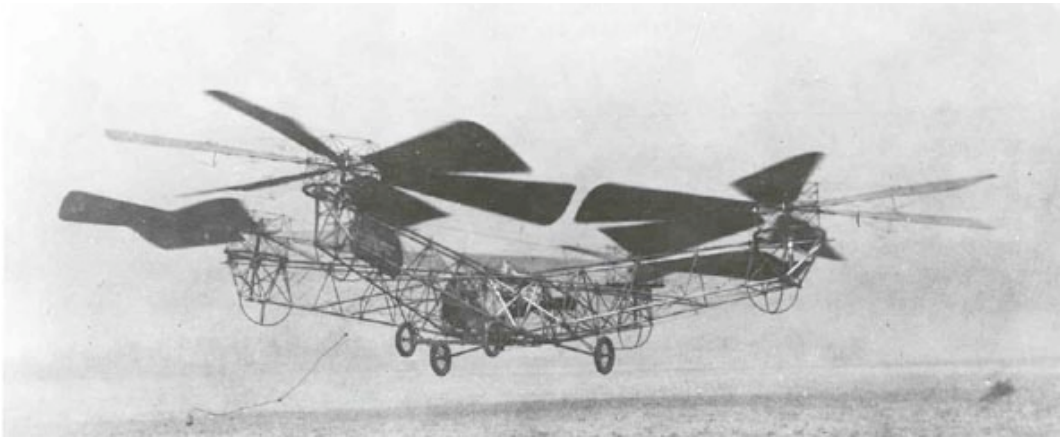
Nowadays, quadrotors are mainly employed as small scale UAVs for research and entertainment purposes. In the last few years some models were successfully introduced to the mainstream market, like the Parrot AR.Drone (Figure 2) which can be controlled from an iPhone and includes some predefined gaming apps. More relevant applications are surveillance and air photography and even the scouting of buildings, thanks to their indoor flying capability. As stated above, quadrotors are also often chosen by scholars and researchers as the reference platform for research in the fields of robotics, autonomous vehicles and flight control algorithms.

[1] In traditional helicopters, the pitch of the main rotor blades can be adjusted either globally, using the so called *collective* control to change the altitude or depending on the blade position, using the *cyclic* control, to change the direction of flight.

[2] The first kilometer-long flight took place in 1924 and lasted 7 minutes and 40 second. Oehmichen was awarded a price for this accomplishment.



(a) Ohemichen



(b) De Bothezat

Figure 1: First prototypes of manned quadrotors.

Figure 2: The Parrot AR.Drone, a commercial smartphone-controlled quadrotor.

State of the art

Most of the research interest about vertical take off and landing UAVs is focused on the quadrotros. Some researchers developed their own platforms, such as the tiny Mesicopter [21], the X4-flier [14] or the one by Castillo [8], while other worked on commercially available models. The majority of the papers regards the development and evaluation of control algorithms. Many different control tecniques that have been proposed include:

- linear contollers (PID, PD) [4, 36];
- Lyapunov theory [8, 31];
- adaptive techniques [3, 27];
- visual feedback [26, 2];
- fuzzy algorithms [9];
- neural networks [11].

Other researches focus on the employs of this aerial vehicles, from multi-agent patrolling and surveillance [16, 10] to more fancy ones, like structure building [23] and river mapping [32].

The application of the extended Kalman filter to the quadrotors, which is the topic of this thesis, is also described in some papers. An attitude determination algorithm is proposed and tested by Liu and Zhou [24], while Abeywardena and Munasinghe [1] analyze the performance of another algorithm using a Matlab simulation. The EKF is used to solve the state estimation problem in the works by Kis [20], Soumelidis [33] and Hoffmann [15], which are however more focused on the control system. Other researchers used the filter on visual informations to perform navigation tasks [25, 35].

Structure of the thesis

The thesis is structured as follows:

Chapter 1 explains the basics of the extended Kalman filter and derives the dynamic and sensor model of the quadrotor in a form that is suitable for the application of the filter.

Chapter 2 describes the Simulink model used to test the implementation of the EKF. The model extends a pre-existing one created by Tommaso Bresciani [6].

Chapter 3 analyzes the performance and the robustness of the Simulink EKF model by presenting data from various simulated test cases.

Appendix A provides the formal mathematical derivation of both the discrete and continuous time versions of the Kalman filter.

Appendix B contains all the Matlab code used by the Simulink model described Chapter 2.

Chapter 1

Attitude estimate with the extended Kalman filter

The first section of this chapter provides a very concise introduction to the Kalman filter equations and implementation. A much more complete description of the filter theory with mathematical proofs can be found in Appendix A. Section 1.2 derives the dynamic equations that describe the quadrotor attitude and movements. In Section 1.5 the on board sensors are described. Finally, in Section 1.6 the dynamic system and the sensor model are put in an appropriate form for the EKF to be applied and their linearizations are computed as required by the filter.

1.1 Introduction to Kalman filtering

The original formulation of the Kalman filter was developed by Rudolf Emil Kálmán at the beginning of the '60s [19, 18]. The filter algorithm assumes a discrete time linear dynamic system that can be represented as a difference equation using the state space model:

$$\boldsymbol{x}_k = \boldsymbol{F}_k \boldsymbol{x}_{k-1} + \boldsymbol{B}_k \boldsymbol{u}_{k-1} + \boldsymbol{w}_k, \tag{1.1}$$

where $\boldsymbol{F}_k$ is the state matrix, $\boldsymbol{x}_k$ is the vector of the state variables, $\boldsymbol{B}_k$ is the input matrix, $\boldsymbol{u}_k$ is the control vector and $\boldsymbol{w}_k$ is a zero-mean, gaussian distributed process noise. If the system is time invariant, the matrices $\boldsymbol{F}$ and $\boldsymbol{B}$ do not depend on the time k. The filter also assumes the availability of a set of measurements that are also linear functions of the state variables:

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{v}_k, \tag{1.2}$$

where $\boldsymbol{H}_k$ is the measurement matrix and $\boldsymbol{v}_k$ is also a zero-mean, gaussian distributed measurement noise uncorrelated with $\boldsymbol{w}_k$

The Kalman filter is a recursive algorithm that exploits the knowledge of the expected covariance of the state variables to correct the a priori estimation of both the state of the system and the covariance itself. The estimate obtained by the

Kalman filter is statistically optimal. The algorithm can be divided in two main steps (in the following equations, a $(-)$ after a variable indicates its estimated value before the measurement update, while a $(+)$ indicates the estimate after the update):

Prediction step: at this step, the state at time k is predicted according to the system model:

$$\hat{\boldsymbol{x}}_k(-) = \boldsymbol{F}_{k-1}\hat{\boldsymbol{x}}_{k-1}(+) + \boldsymbol{B}_{k-1}\boldsymbol{u}_{k-1}. \tag{1.3}$$

The covariance at time k is also predicted according to the equation:

$$\boldsymbol{P}_k(-) = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1}(+)\boldsymbol{F}_{k-1}^{\boldsymbol{T}} + \boldsymbol{Q}_{k-1}, \tag{1.4}$$

where $\boldsymbol{Q}_k$ is the covariance of the process noise $\boldsymbol{w}_k$ in Equation (1.1).

Update step: at this step, the a priori estimate is updated according to the measurements:

$$\hat{\boldsymbol{x}}_k(+) = \hat{\boldsymbol{x}}_k(-) + \overline{\boldsymbol{K}}_k[\boldsymbol{z}_k - \boldsymbol{H}_k\hat{\boldsymbol{x}}_k(-)] \tag{1.5}$$

and the covariance estimate is also updated:

$$\boldsymbol{P}_k(+) = [\boldsymbol{I} - \overline{\boldsymbol{K}}_k\boldsymbol{H}_k]\boldsymbol{P}_k(-). \tag{1.6}$$

The matrix $\overline{\boldsymbol{K}}_k$ in Equations (1.5) and (1.6) is the optimal Kalman gain at time k, which is computed as:

$$\overline{\boldsymbol{K}}_k = \boldsymbol{P}_k(-)\boldsymbol{H}_k^T[\boldsymbol{H}_k\boldsymbol{P}_k(-)\boldsymbol{H}_k^T + \boldsymbol{R}_k]^{-1}, \tag{1.7}$$

where $\boldsymbol{R}_k$ is the covariance of the measurement noise $\boldsymbol{v}_k$ in Equation 1.2.

In practical applications, the error covariance matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ and the measurement matrix $\boldsymbol{H}$ are often time invariant.

1.1.1 Kalman filter for continuous time systems

A version of the Kalman filter known as the *Kalman-Bucy filter* can be applied to continuous time dynamic system. In this case, the system model is a differential equation:

$$\dot{\boldsymbol{x}} = \boldsymbol{F}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) + \boldsymbol{w}(t) \tag{1.8}$$

and the measurement model is also a continuous linear function of the state:

$$\boldsymbol{z}(t) = \boldsymbol{H}(t)\boldsymbol{x}(t) + \boldsymbol{v}(t). \tag{1.9}$$

The main diffrerence with respect to the discrete time version is that in the continuous domain the update and the prediction steps are coupled and cannot be distinguished, so the filter consists of only two equations:

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{F}(x)\hat{\boldsymbol{x}}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) + \overline{\boldsymbol{K}}(t)[\boldsymbol{z}(t) - \boldsymbol{H}(t)\hat{\boldsymbol{x}}(t)], \tag{1.10}$$

$$\dot{\boldsymbol{P}}(t) = \boldsymbol{F}(t)\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{F}^{\boldsymbol{T}}(t) - \overline{\boldsymbol{K}}(t)\boldsymbol{R}(t)\overline{\boldsymbol{K}}^{\boldsymbol{T}}(t) + \boldsymbol{Q}(t) \tag{1.11}$$

and the Kalman gain is:

$$\overline{\boldsymbol{K}}(t) = \boldsymbol{P}(t)\boldsymbol{H}^{\boldsymbol{T}}(t)\boldsymbol{R}^{-1}. \tag{1.12}$$

The equation (1.11) that expresses the covariance update is known as a Riccati equation[1].

1.1.2 The extended Kalman filter

One of the main limitations of the Kalman filter is the fact that it requires both the dynamic system and the measurement functions to be linear with respect to the state variables. In many pratical applications, these requirements are not satisfied. The extended Kalman filter is a modified version of the standard Kalman filter that can be applied when the system and/or the measurement models are nonlinear.

Equation (1.13) and (1.14) are respectively the nonlinear differential (for the continuous case) and difference (for the discrete case) equations that describe the system model:

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) + \boldsymbol{w}(t), \tag{1.13}$$

$$\boldsymbol{x}_k = \Phi_{k-1}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_{k-1}) + \boldsymbol{w}_{k-1}. \tag{1.14}$$

The nonlinear measurement model equation for the continuous and discrete time version are respectively:

$$\boldsymbol{z}(t) = h(\boldsymbol{x}(t)) + \boldsymbol{v}(t), \tag{1.15}$$

$$\boldsymbol{z}_k = h_k(\boldsymbol{x}_k) + \boldsymbol{v}_k. \tag{1.16}$$

The nonlinear equations are used directly for the state prediction and to compute the measurement residual, so Equation (1.3), (1.5) and (1.10) become respectively:

$$\hat{\boldsymbol{x}}_k(-) = \Phi_{k-1}(\hat{\boldsymbol{x}}_{k-1}(+), \boldsymbol{u}_{k-1}), \tag{1.17}$$

$$\hat{\boldsymbol{x}}_k(+) = \hat{\boldsymbol{x}}_k(-) + \overline{\boldsymbol{K}}_k[\boldsymbol{z}_k - h_k(\hat{\boldsymbol{x}}_k(-))], \tag{1.18}$$

$$\dot{\hat{\boldsymbol{x}}}(t) = f(\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t)) + \overline{\boldsymbol{K}}(t)[\boldsymbol{z}(t) - h(\hat{\boldsymbol{x}}(t))]. \tag{1.19}$$

The part concerning the covariance update is more complex. If the state transition and measurement equations are nonlinear, the probability distribution of the state vector becomes non-Gaussian, so it cannot be fully described by the mean and the covariance matrix alone. The EKF linearizes the state transition and measurement models around the current state estimate and it uses the Jacobians to update the covariance estimate with the same equations used by the standard Kalman filter. So in the discrete model we redefine $\boldsymbol{F}_k$ in Equation (1.4) and $\boldsymbol{H}_k$ in Equation (1.6)

[1] A Riccati equation is any first order ordinary differential equation (in scalar or matrix form) that is quadratic in the unknown function. It is named after the Italian matematician Jacopo Francesco Riccati (1676-1754).

and (1.7) as:

$$\boldsymbol{F}_{k-1} = \begin{bmatrix} \frac{\partial \Phi_1}{\partial x_1} & \cdots & \frac{\partial \Phi_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \Phi_m}{\partial x_1} & \cdots & \frac{\partial \Phi_m}{\partial x_n} \end{bmatrix}_{\hat{\boldsymbol{x}}_{k-1}(+), \boldsymbol{u}_{k-1}} ,$$

$$\boldsymbol{H}_k = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix}_{\hat{\boldsymbol{x}}_k(-)} .$$

In the continuous model, we redefine $\boldsymbol{F}(t)$ and $\boldsymbol{H}(t)$ in Equation (1.11) and (1.12) as:

$$\boldsymbol{F}(t) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}_{\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t)} ;$$

$$\boldsymbol{H}(t) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \cdots & \frac{\partial h_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1} & \cdots & \frac{\partial h_m}{\partial x_n} \end{bmatrix}_{\hat{\boldsymbol{x}}(t)} .$$

Since the Jacobians have to be evaluated at the current estimate in real time, the EKF is more complex than the standard Kalman filter from the computational point of view.

1.2 Quadrotor dynamics

In order to provide the state-transition model to the extended Kalman filter, we need a physical model of the quadrotor dynamics and kinematics that is both accurate and simple enough to be mathematically analyzed and simulated on a computer system. The proposed model is shown in Figure 1.2: the quadrotor is represented as a symmetric cross-like structure with a spherical central mass and four propellers, the front and the back ones rotate clockwise and the other two rotate counterclockwise. The mass of the structure that connects the propellers to the central mass is assumed negligible. The model is completed with the definition of two different Cartesian reference frames: the *earth frame* and the *body frame*, which are fully described in the following subsection.

1.2.1 Earth frame and body frames

As stated before, our dynamic model involves two different reference frames: the earth frame is centered on a given point at ground level with the z_f axis pointing upwards, y_f pointing towards the magnetic north and x_f pointing to the east, while

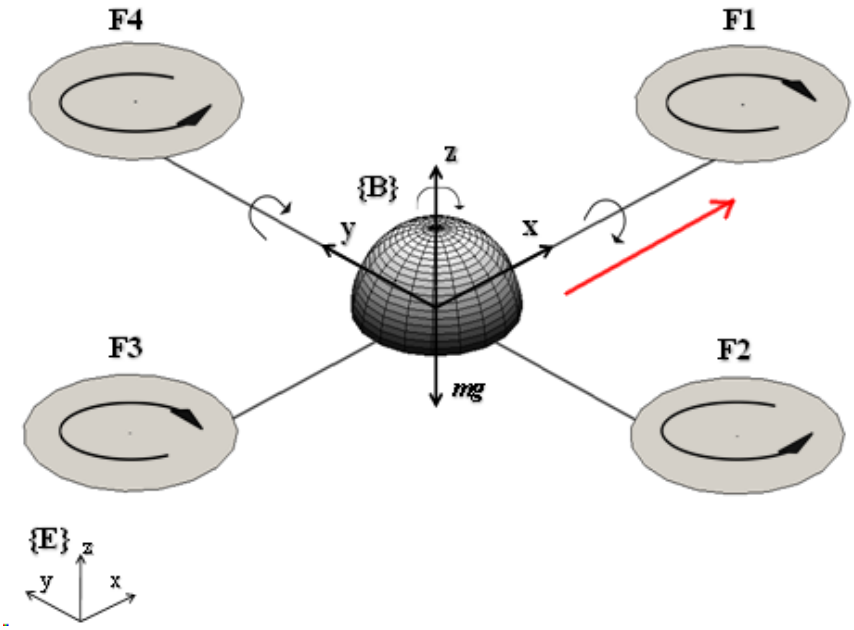


Figure 1.1: Model of the quadrotor with two Cartesian reference frames. The red arrow indicates the main direction of motion.

the body frame is centered at the mass center of the aircraft, with the x_b axis directed towards the main direction of motion (which is usually arbitrary because of the symmetric structure of the quadrotor), the y_b axis pointing to the left and the z_b axis pointing upwards with respect to the aircraft body. Both the frames follow the right hand convention.

Once we have defined the two reference frames, we need a set of transformations between them, so that any vector given in the earth frame can be expressed in the body frame and vice-versa. These transformations are required because our goal is to control the attitude of the object with respect to the enviroment, but the measurements provided by the on-board sensors are referred to the body frame. While the translations are quite straightforward and actually they are not even really important for our purposes (an active control of the position is not developed in this thesis), the three dimensional rotations are much more complex than their two-dimensional counterpart, so a complete explanation is needed.

1.2.2 Euler angles

The Euler angles are one way to describe the orientation of a rigid body with respect to a fixed reference frame. They were introduced by Leonhard Euler in the 18th century, but are still widely used. The Euler angles can be explaind using the rotations around the moving axes, known as *intrinsec* rotations, that are simple to imagine: if we at first rotate the frame around the x axis, then the y axis will end up

pointing in a different direction than the initial one. A second rotation around the y axis will rotate the frame around this new direction. In general, each new rotation is referred to the direction that its axis has assumed after all the previous ones.

Given two cartesian reference frames with the same origin, the first frame can be rotated to match the orientation of the second frame by applying three rotations around the moving axes in a given order. The sequence of the three rotations uniquely defines the orientation of the second reference frame with respect to the first one. To avoid ambiguity, the chosen sequence of rotation axes must follow a common convention. We will use the sequence $z - y - x$, which is conventional in aerospace engineering. The resulting angles ψ, θ and φ are called respectively *yaw*, *roll* and *pitch*[2].

Rotation matrix

The rotations around the three axes of a Cartesian frame are analitically described by the following matrices:

$$\boldsymbol{R}(x,\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{bmatrix},$$

$$\boldsymbol{R}(y,\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix},$$

$$\boldsymbol{R}(z,\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

It is important to notice that we have described the Euler angles using the rotations around the moving axis, because they are easier to understand by intuition, but the above matrices perform the rotations around the axes of the fixed reference frame (called *extrinsic* rotations). However, it is possibile to prove that any sequence of intrinsic rotations is equivalent to the same sequence of extrinsec rotations performed in reverse order, in our case $x - y - z$ is the opposite of $z - y - x$ (that's also the reason why the Euler angles are usually listed as roll, pitch and yaw and not the other way around as we did in the previous paragraph). So, the full rotation matrix is computed in the following way[3]:

[2] The angles defined according to this convention are more properly called Tait-Benn angles, because the original Euler angles perform the first and the last rotations around the same axis.

[3] Remember that the matrix product is not commutative and the first rotation is associated to the leftmost matrix.

$$\boldsymbol{R}(\varphi,\theta,\psi) = \boldsymbol{R}(z,\psi)\boldsymbol{R}(y,\theta)\boldsymbol{R}(x,\varphi) = \\ = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\varphi - \sin\psi\cos\varphi & \cos\psi\sin\theta\cos\varphi + \sin\psi\sin\varphi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\varphi + \cos\psi\cos\varphi & \sin\psi\sin\theta\cos\varphi - \cos\psi\sin\varphi \\ -\sin\theta & \cos\theta\sin\varphi & \cos\theta\cos\varphi \end{bmatrix}. \tag{1.20}$$

The matrix $\boldsymbol{R}(\varphi,\theta,\psi)$ and its inverse (which corresponds to its transpose because it is an orthogonal matrix) can be used to switch between the body frame and the earth frame: if $\boldsymbol{x}$ is a vector in the body frame, $\boldsymbol{Rx}$ expresses the same vector in the earth frame, on the contrary, a vector $\boldsymbol{y}$ defined in the earth frame becomes $\boldsymbol{R^T y}$ in the body frame.

Geometric definition of the Euler angles

In the previous paragraphs we have defined the Euler angles in the Cartesian space by an operative definition, decribing how to move one frame to make it conincident with the second one, and in an analitic way, explaining how to obtain the body frame by transforming the coordinates of the earth frame. However, the angles φ, θ and ψ can also be described by a geometric definition, as shown in Figure 1.2.2. First of all, we define the *line of nodes* as the interception between the earth frame $x_f y_f$ plane and the body frame $y_b z_b$ plane[4]. Then, according to our previous convention, the Euler angles are defined as follows:

- φ is the angle between y_b and the line of nodes;
- θ is the angle between x_b and its projection on the $y_f y_f$ plane;
- ψ is the angle between y_f and the line of nodes.

1.2.3 Angular velocities

Now we can use the Euler angles to express the angular velocity, which is defined as:

$$\boldsymbol{\omega} = \frac{d\theta}{dt}\boldsymbol{u} \tag{1.21}$$

where θ is the rotation angle and $\boldsymbol{u}$ is the versor oriented in the direction of the rotation axis. If $\boldsymbol{\omega}$ is the angular velocity of the quadrotor, it can be decomposed along three directions: the line of nodes, the z_f axis and the x_b axis:

$$\boldsymbol{\omega} = \boldsymbol{\omega}_n + \boldsymbol{\omega}_{z_f} + \boldsymbol{\omega}_{x_b}. \tag{1.22}$$

[4] According to this definition, the line of nodes is undefined when $\theta = \pm 90°$. In this case, two of the three rotation axes become coincident and the system loses one degree of freedom. This is a well-known problem of the Euler angles called the *gimbal lock*.

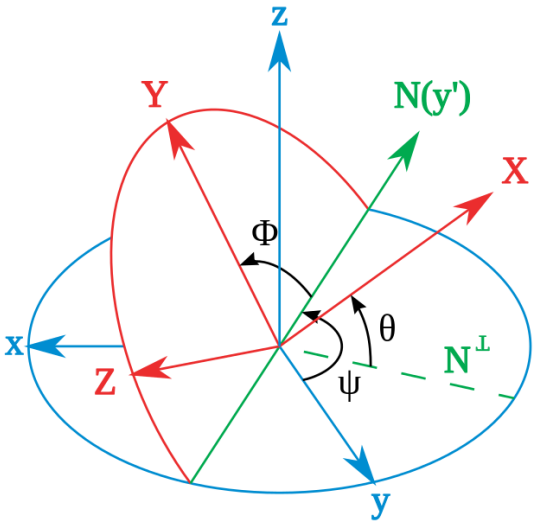


Figure 1.2: Visual representation of the Euler angles

These three directions are not an orthogonal basis, but it is convenient to use them because they are the rotation axes of the Euler angles that we have just defined, so we can write:

$$\boldsymbol{\omega} = \dot{\varphi}\boldsymbol{x_b} + \dot{\theta}\boldsymbol{n} + \dot{\psi}\boldsymbol{z_f}. \tag{1.23}$$

Finally, we can find the angular velocity with respect to the body frame by transforming the versors $\boldsymbol{z_f}$ and $\boldsymbol{n}$ with the matrix $\boldsymbol{R^T}$:

$$\boldsymbol{R^T z_f} = \boldsymbol{R^T} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin\theta \\ \cos\theta\sin\varphi \\ \cos\theta\cos\varphi \end{bmatrix}, \tag{1.24}$$

$$\boldsymbol{R^T n} = \boldsymbol{R^T} \begin{bmatrix} 0 \\ -\sin\psi \\ \cos\psi \end{bmatrix} = \begin{bmatrix} 0 \\ \cos\varphi \\ -\sin\varphi \end{bmatrix}, \tag{1.25}$$

$$\boldsymbol{\omega} = (\dot{\varphi} - \dot{\psi}\sin\theta)\boldsymbol{x_b} + (\dot{\theta}\cos\varphi + \dot{\psi}\cos\theta\sin\varphi)\boldsymbol{y_b} + (-\dot{\theta}\sin\varphi + \dot{\psi}\cos\theta\cos\phi)\boldsymbol{z_b}. \tag{1.26}$$

The latter equation can be written in matrix form:

$$\boldsymbol{\omega_B} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \boldsymbol{R_R}(\varphi,\theta,\psi) \begin{bmatrix} \dot{\varphi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$$

with:

$$\boldsymbol{R_R}(\varphi,\theta,\psi) = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\varphi & \sin\varphi\cos\theta \\ 0 & -\sin\varphi & \cos\varphi\cos\theta \end{bmatrix}. \tag{1.27}$$

In practice, since the sensors on the quadrotor directly measure the values of p, q and r, the inverse of $\boldsymbol{R_R}$ is much more useful. First of all, we note that $\mathbf{det}(\boldsymbol{R_R}) = \cos\theta$, so the matrix becomes singular for $\theta = \pm 90°$ (a consequence of the gimbal lock). For $\theta \neq \pm 90°$ the matrix is invertible and its inverse is:

$$\boldsymbol{R_R}(\varphi, \theta, \psi)^{-1} = \begin{bmatrix} 1 & \sin\varphi\tan\theta & \cos\varphi\tan\theta \\ 0 & \cos\varphi & -\sin\varphi \\ 0 & \frac{\sin\varphi}{\cos\theta} & \frac{\cos\varphi}{\cos\theta} \end{bmatrix}. \tag{1.28}$$

1.3 Rigid body dynamics

The dynamic equations for the quadrotor are usually formulated in the body frame rather than in the earth frame, because of some reasons:

- the inertia matrix is time-invariant;
- body symmetry can be exploited to simplify the equations;
- sensor measurements and driving forces are naturally expressed in the body frame.

1.3.1 Linear motion

The well known Newton's second law states that:

$$m\frac{d\boldsymbol{v}}{dt} = \boldsymbol{F} \tag{1.29}$$

where $\boldsymbol{F}$ is the total force acting on the body center of mass.

Since Newton's laws are valid only for inertial systems, we cannot directly apply Equation (1.29) to the body frame. If we have a moving vector inside a reference frame which, in turn, rotates with respect to a fixed frame, we can apply the equation of Coriolis to relate the derivative of the vector in the body frame $\left(\frac{d\boldsymbol{v}}{dt}\right)_b$ to its derivative in the inertial frame $\left(\frac{d\boldsymbol{v}}{dt}\right)_i$:

$$\left(\frac{d\boldsymbol{v}}{dt}\right)_i = \left(\frac{d\boldsymbol{v}}{dt}\right)_b + \boldsymbol{\omega_{b/i}} \times \boldsymbol{v_b}, \tag{1.30}$$

where $\boldsymbol{\omega_{b/i}}$ is the angular velocity of the moving frame with respect to the fixed one. Combining Equations (1.29) and (1.30) we can write:

$$m\left(\dot{\boldsymbol{v}}_{\boldsymbol{b}} + \boldsymbol{\omega_{b/i}} \times \boldsymbol{v_b}\right) = \boldsymbol{F},$$

and then:

$$\dot{\boldsymbol{v}}_{\boldsymbol{b}} = -\boldsymbol{\omega_{b/i}} \times \boldsymbol{v_b} + \frac{\boldsymbol{F}}{m}. \tag{1.31}$$

1.3.2 Angular motion

Newton's second law for the angular acceleration has the following form:

$$\frac{d\boldsymbol{h}}{dt} = \boldsymbol{\tau}, \tag{1.32}$$

where $\boldsymbol{h}$ is the angular momentum of the rigid body and $\boldsymbol{\tau}$ is the applied torque. We can obtain its expression in the body frame using the same Equation (1.30) that we used for the linear acceleration:

$$\dot{\boldsymbol{h_b}} = -\boldsymbol{\omega} \times \boldsymbol{h_b} + \boldsymbol{\tau} \tag{1.33}$$

and then, since the angular momentum is the angular velocity multiplied by the time-invariant inertia matrix $\boldsymbol{J}$:

$$\boldsymbol{J}\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega} \times \boldsymbol{J}\boldsymbol{\omega} + \boldsymbol{\tau}, \tag{1.34}$$

with $\boldsymbol{\omega} = [p, q, r]^T$.

The moment of inertia tensor

The moment of inertia tensor for a rigid body is a positive semidefinite symmetric matrix:

$$\boldsymbol{J} = \begin{bmatrix} J_{XX} & J_{XY} & J_{XZ} \\ J_{XY} & J_{YY} & J_{YZ} \\ J_{XZ} & J_{YZ} & J_{ZZ} \end{bmatrix}. \tag{1.35}$$

As stated before, the quadrotor can be modeled as a central sphere of mass M and radius R connected with four point masses m at distance l from the center (representing the motors). According to this model, the quadrotor is symmetrical about the three axes, so $\boldsymbol{J}$ becomes a diagonal matrix with:

$$\begin{aligned} \boldsymbol{J_{XY}} &= \boldsymbol{J_{XZ}} = \boldsymbol{J_{YZ}} = 0, \\ \boldsymbol{J_{XX}} &= \boldsymbol{J_{YY}} = \frac{2MR^2}{5} + 2ml^2, \\ \boldsymbol{J_{ZZ}} &= \frac{2MR^2}{5} + 4ml^2. \end{aligned}$$

1.3.3 Altitude

An attitude control system does not perform navigation tasks, so the estimation of the linear position with respect to the earth frame is not needed. However the vertical component z must be taken into account, because the system should avoid unwanted and potentially dangerous altitude variations while manouvering. The derivative of z is the vertical component of the velocity vector with respect to the body frame, which can be computed from the body frame velocity values using the matrix (1.28):

$$\dot{z} = -\sin\theta u + \cos\theta \sin\varphi v + \cos\theta \cos\varphi w. \tag{1.36}$$

1.4 Forces and controls

The only variable that we can directly control to affect the motion of the quadrotor is the speed of each propeller, so we have to relate their values to the torques τ_x, τ_y and τ_z, which are the control variables described later in Equation (1.61). For most flight conditions of practical interest, it can be proven that each motor generates a force that is proportional to the square of its angular speed and perpendicular to the rotor plane: $F_i = b\Omega_i^2$, where b is a constant. Thus the total force acting on the quadrotor is always directed upwards with respect to the body frame and its modulus is:

$$F = F_r + F_l + F_f + F_b = b(\Omega_r^2 + \Omega_l^2 + \Omega_f^2 + \Omega_b^2). \tag{1.37}$$

This force, combined with the proper roll and pitch angles, can be used to move the quadrotor in any direction.

The rolling torque is the result of the difference between the forces of the left and the right motors and the pitching torque is produced in the same way by the difference between the forces of the front and the back motors:

$$\tau_\varphi = l(F_l - F_r) = bl(\Omega_l^2 - \Omega_r^2), \tag{1.38}$$

$$\tau_\theta = l(F_f - F_b) = bl(\Omega_f^2 - \Omega_b^2). \tag{1.39}$$

In these two equations, l is the distance between the rotors and the mass centre of the quadcopter. The yawing torque is generated by a different principle: according to Newton's third law, each rotor produces a yawing torque on the body of the quadrotor in the opposite direction of the blades rotation, proportional to the square of the angular speed: $\tau_i = d\Omega_i^2$. The total yawing torque is then the sum of the contributions of the single rotors[5]:

$$\tau_\psi = \tau_r + \tau_l - \tau_f - \tau_b = d(\Omega_r^2 + \Omega_l^2 - \Omega_f^2 - \Omega_b^2). \tag{1.40}$$

To be more accurate, the rotors, like every rotating object, are subject to the gyroscopic effect. This implies that a torque perpendicular to the rotating plane generates a precession movement perpendicular to both the rotating plane and the torque. According to this principle, a pitch torque produces a roll and vice versa, so the equations (1.38) and (1.39) become:

$$\tau_\varphi = bl(\Omega_l^2 - \Omega_r^2) + J_m q(\Omega_f + \Omega_b - \Omega_l - \Omega_r), \tag{1.41}$$

$$\tau_\theta = bl(\Omega_f^2 - \Omega_b^2) + J_m p(\Omega_l + \Omega_r - \Omega_f - \Omega_b) \tag{1.42}$$

where J_m is the inertia of the motor. In these equations, the term due to the gyroscopic effect is proportional to the difference between the speed of the motors rotating clockwise and the ones rotating counter clockwise, because they generate

[5]The following equation is true if we assume that the left and right rotors rotate clockwise and the front and back ones rotate counterclockwise, otherwise the signs are reversed.

two opposite torques that cancel out. This term is different from zero only when the quadrotor is both yawing and rolling or pitching at the same time and even in that case it is very small with respect to the other term of the equation, so the gyroscopic effect is often negected for pratical applications.

Now we can choose the control variables to make the dynamic equations linear with respect to them. The most natural choice is:

$$\begin{aligned} U_1 &= b(\Omega_f^2 + \Omega_b^2 + \Omega_l^2 + \Omega_r^2), \\ U_2 &= bl(\Omega_l^2 - \Omega_r^2), \\ U_3 &= bl(\Omega_f^2 - \Omega_b^2), \\ U_4 &= bl(\Omega_r^2 + \Omega_l^2 - \Omega_f^2 - \Omega_b^2). \end{aligned}$$

If we write the previous equations in matrix form we obtain:

$$\boldsymbol{U} = \boldsymbol{A}\boldsymbol{\Omega}_{\boldsymbol{sq}} = \begin{bmatrix} b & b & b & b \\ 0 & -bl & 0 & bl \\ -bl & 0 & bl & 0 \\ -bl & bl & -bl & bl \end{bmatrix} \begin{bmatrix} \Omega_b^2 \\ \Omega_r^2 \\ \Omega_f^2 \\ \Omega_l^2 \end{bmatrix}. \tag{1.43}$$

Since matrix $\boldsymbol{A}$ is invertible, each control variable U_i can assume any value independently from the others. Given the desired values of the control variables, the angular speeds required to obtain them can be computed as:

$$\begin{cases} \Omega_b = \sqrt{\frac{1}{4b}U_1 - \frac{1}{2bl}U_3 - \frac{1}{4d}U_4} \\ \Omega_r = \sqrt{\frac{1}{4b}U_1 - \frac{1}{2bl}U_2 + \frac{1}{4d}U_4} \\ \Omega_f = \sqrt{\frac{1}{4b}U_1 + \frac{1}{2bl}U_3 - \frac{1}{4d}U_4} \\ \Omega_l = \sqrt{\frac{1}{4b}U_1 + \frac{1}{2bl}U_2 + \frac{1}{4d}U_4} \end{cases}. \tag{1.44}$$

A problem arises if the quantities under the square roots are negative. In that case we would have imaginary values of Ω, which obviously have no physical meaning, thus the desired control is not achievable[6].

1.5 On board sensors

Every aircraft has a set of sensors that provide the information needed by the attitude and the navigation control systems. This set of sensors is usually called an IMU (inertial measurement unit). The IMU of a quadrotor contains the following sensors:

[6] Actually, the rotor speed values of a real quadrotor are limited to some interval, so even besides the imaginary case there is a big set of controls that are theoretically possibile, but cannot be achieved in pratice. Some dedicated control system has to be implemented to handle this problem.

- an accelerometer;
- a gyroscope;
- a magnetometer;
- a barometer[7].

A typical three-axis MEMS sensor is calibrated according to this linear model:

$$\begin{bmatrix} y_x \\ y_y \\ y_z \end{bmatrix} = \begin{bmatrix} 1 & M_{xy} & M_{xz} \\ M_{yx} & 1 & M_{yz} \\ M_{zx} & M_{zy} & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{S_x} & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 \\ 0 & 0 & \frac{1}{S_z} \end{bmatrix} \begin{bmatrix} x_x \\ x_y \\ x_z \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}. \tag{1.45}$$

The variables used in the above equation are:

- x_i: input value along the i-th axis;
- y_i: sensor output for the i-th axis;
- b_i: sensor bias on the i-th axis;
- v_i: gaussian distributed random error on the i-th axis;
- S_i: scale factor of the ith axis;
- M_{ij}: sensitivity of the ith axis output to jth axis input.

In particular, the coefficients M_{ij} and the scale factors S_i reflect the fact that the axes of these low cost sensors are not perfectly aligned and their sensitivity differs from the nominal one. All the parameters vary with the temperature, but are assumed to be constant during the flight time of the quadrotor so a dynamic calibration is not required.

1.5.1 Gyroscope

The gyroscope measures the angular rates around the three axes. Its sensor model is a simple application of Equation (1.45):

$$\boldsymbol{y_g} = \boldsymbol{M_g S_g} \omega + \boldsymbol{b_g} + \boldsymbol{v_g} \tag{1.46}$$

where $\boldsymbol{\omega} = [p, q, r]^T$.

[7] The barometer is used only to control the altitude and it is not a typical part of an IMU, so it may be considered as an independet sensor.

1.5.2 Accelerometer

The accelerometer measures a quantity called *proper acceleration* along the three axes. The proper acceleration is the sum of the linear acceleration (the derivative of the linear velocity) and a constant pseudo-acceleration directed upwards with respect to the body frame that has the same magnitude as the gravity vector $\boldsymbol{g}$[8]. If $\boldsymbol{a}$ is the proper acceleration vector in the body frame, from Equation (1.45) we have:

$$\boldsymbol{y_a} = \boldsymbol{M_a S_a a} + \boldsymbol{b_a} + \boldsymbol{v_a}. \tag{1.47}$$

Following the definition of proper acceleration, the components of the vector $\boldsymbol{a}$ can be written as:

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} g \sin\theta \\ -g \cos\theta \sin\varphi \\ -g \cos\theta \cos\varphi \end{bmatrix}, \tag{1.48}$$

where $\dot{u}$, $\dot{v}$ and $\dot{w}$ are the time derivatives of the linear velocity vector. Substituting in Equation (1.47) we have:

$$\boldsymbol{y_a} = \boldsymbol{M_a S_a} \left(\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} + \begin{bmatrix} g \sin\theta \\ -g \cos\theta \sin\varphi \\ -g \cos\theta \cos\varphi \end{bmatrix} \right) + \boldsymbol{b_a} + \boldsymbol{v_a}. \tag{1.49}$$

1.5.3 Magnetometer

A magnetometer is a sensor that measures the intensity and the direction of a magnetic field. If the earth magnetic field vector is known, the sensor output can be used to estimate the attitude of the quadrotor. According to Equation (1.45), in absence of magnetic distortions, the output of the magnetometer is:

$$\boldsymbol{y_m} = \boldsymbol{M_m S_m m_b} + \boldsymbol{b_m} + \boldsymbol{v_m}, \tag{1.50}$$

where $\boldsymbol{m_b}$ is the magnetic field vector with respect to the body frame, S and M are the matrices of scale and misalignment factors, b_s is the sensor bias and $\boldsymbol{v_k}$ is a random gaussian white noise.

Hard and soft iron distortions

Hard and soft iron distortions are caused by magnetic fields and metallic objects surrounding the sensor. If the sources of the distortions are parts of the quadrotor or its payload (motors, battery...), the distortions can be measured and corrected statically [28].

[8] The formal definition of proper acceleration is much more complex and involves relativity theory.

Hard iron distortions The hard iron distortions are caused by objects that produce a magnetic field. These distortions introduce a fixed bias in the measurements, so Equation (1.50) needs to be corrected by adding the bias vector h_h:

$$\boldsymbol{y_m} = \boldsymbol{M_m S_m m_b} + \boldsymbol{b_s} + \boldsymbol{h_h} + \boldsymbol{v_m}. \tag{1.51}$$

Soft iron distortions The soft iron distortions are caused by ferromagnetic materials surrounding the sensor. These distortions are more complex to model than the hard iron ones, because they depend upon the direction of the magnetic field with respect to the sensor. To be more precise, let the sensor freely rotate in the space and let S be the set of all the possible values that the measured magnetic field can assume. In absence of distortions, S is the surface of a sphere, because the measured vector can assume any direction, but its module does not change. The soft iron distortions apply a linear transformation to the measurement space, changing S into an ellipsoid. A linear transformation in a tridimensional space is defined by a 3-by-3 matrix, so the sensor model becomes:

$$\boldsymbol{y_m} = \boldsymbol{H_s M_m S_m m_b} + \boldsymbol{b_s} + \boldsymbol{h_h} + \boldsymbol{v_m}. \tag{1.52}$$

However, we can sum the two bias vector and consider them as a single bias $\boldsymbol{b_m}$ and multiply the matrices $\boldsymbol{H_s}$ and $\boldsymbol{M_m}$ to obtain a single transformation matrix $\boldsymbol{G}$. This way, the espression for the sensor model becomes very similar to the standard Equation (1.45), except for the diagonal elements of $\boldsymbol{G}$, which here can be different from 1.

Since we want to use the sensor to determine the attitude of the quadrotor, it is useful to express the magnetic filed vector in body frame m_b as a function of the euler angles φ, θ and ψ by using the inverse of the matrix $\boldsymbol{R}(\varphi, \theta\psi)$ defined in Equation (1.20): if $\boldsymbol{m_f}$ is the magnetic field vector measured in the fixed frame we have:

$$\boldsymbol{m_b} = \boldsymbol{R}^T(\varphi, \theta, \psi)\boldsymbol{m_f},$$

so Equation (1.52) becomes:

$$\boldsymbol{y_m} = \boldsymbol{G R^T}(\varphi, \theta, \psi)\boldsymbol{m_f} + \boldsymbol{b_m} + \boldsymbol{v_m}. \tag{1.53}$$

1.5.4 Barometer

The barometer is different from the other sensors that we have described, because it measures a scalar value, so the model in Equation (1.45) cannot be applied. We want to use the barometer as an altimeter, so the instrument needs to know the current value of the pressure at the ground level (or another reference altitude). A static calibration is not possibile, because the ground pressure changes with weather conditions. In practice, the pressure is measured before the take off and assumed to be constant during the flight time. The pressure P and the altitude z are related by this formula:

$$P = P_0 e^{-\frac{g}{RT}(z - z_0)}, \tag{1.54}$$

where P_0 is the pressure at the known altitude z_0 (usually the ground), g is the gravitational acceleration, R is the dry air gas constant and T is the temperature at the altitude z[9]. Since the quadrotor is expected to fly no higher that a few tenths of meters, we can assume T to be a constant, equal to the value T_0 measured on the ground.

The sensor model used for the barometer is:

$$y_b = k_b x_b + b_b + v_b. \tag{1.55}$$

Combining the latter with Equation (1.54) and imposing $z_0 = 0$ we obtain the output of the sensor as a function of the altitude z:

$$y_a = k_a P_0 e^{-\frac{g}{RT}z} + b_a + v_a. \tag{1.56}$$

1.6 State space representation for the EKF

In the previous sections we have provided a matemathical description of the quadrotor dynamics and controls and the operating principles of its sensors. Now we want to rearrange these equations in a form that is suitable for the application of the continuous time EKF. At first, we have to identify the system variables, which are the variables that appear in the dynamic equations:

- u, v, w, the linear velocity components along the three axes of the body frame;
- p, q, r, the angular velocitiy components around the three axes of the body frame;
- φ, θ and ψ, the Euler angles roll, pitch and yaw (in this order);
- z, the altitude of the quadrotor.

Therefore, we have to express both the state-transition model and the sensor model according to this set of 10 variables.

1.6.1 State-transition model

The general state-transition model for the continuous time EKF is described by the equation (1.8).

[9] A more accurate version of the equation (1.54) should take into account the relative humidity. Note also that the assumption that P_0 is constant is always true only for short periods of time, because weather changes cause significant long-term pressure variations.

Linear velocity

The cross product in Equation (1.31) can be expressed in matrix form as:

$$\boldsymbol{\omega} \times \boldsymbol{v_b} = [\boldsymbol{\omega}]_\times \boldsymbol{v_b} = \begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \tag{1.57}$$

where $[\cdot]_\times$ is the skew operator. We have to take into account also the gravitational force, which generates a fixed acceleration of modulus g always directed downwards with respect to the earth frame. The equation finally becomes:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} g \sin\theta \\ -g \cos\theta \sin\phi \\ -g \cos\theta \sin\phi \end{bmatrix} + \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} \tag{1.58}$$

$$= \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv \end{bmatrix} + \begin{bmatrix} g \sin\theta \\ -g \cos\theta \sin\phi \\ -g \cos\theta \sin\phi \end{bmatrix} + \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}. \tag{1.59}$$

Angular velocity

We can rewrite the equation (1.34) to explicit the derivatives of the angular velocity components:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} J_{XX}^{-1} & 0 & 0 \\ 0 & J_{YY}^{-1} & 0 \\ 0 & 0 & J_{ZZ}^{-1} \end{bmatrix} \left(\begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix} \begin{bmatrix} J_{XX} & 0 & 0 \\ 0 & J_{YY} & 0 \\ 0 & 0 & J_{ZZ} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} \tau_\varphi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \right) \tag{1.60}$$

$$= \begin{bmatrix} \frac{J_{YY} - J_{ZZ}}{J_{XX}} qr \\ \frac{J_{ZZ} - J_{XX}}{J_{YY}} pr \\ \frac{J_{XX} - J_{YY}}{J_{ZZ}} pq \end{bmatrix} + \begin{bmatrix} J_{XX}^{-1} \tau_\varphi \\ J_{YY}^{-1} \tau_\theta \\ J_{ZZ}^{-1} \tau_\psi \end{bmatrix}. \tag{1.61}$$

Euler angles

We do not have a dynamic equation that involves the derivatives of the Euler angles, but we have used them to define the angular velocity with respect to the body frame, so we can use the matrix $\boldsymbol{R_R}(\varphi, \theta, \psi)^{-1}$ in Equation (1.28) to express the derivatives of φ, θ and ψ as functions of the angular velocity components p, q and r:

$$\begin{cases} \dot{\varphi} = p + \sin\phi \tan\theta q + \cos\phi \tan\theta r \\ \dot{\theta} = \cos\varphi q - \sin\varphi r \\ \dot{\psi} = \frac{\sin\varphi}{\cos\theta} q + \frac{\cos\varphi}{\cos\theta} r \end{cases}. \tag{1.62}$$

Complete model

We can now write the complete state-transition model by combining Equations (1.59), (1.61), (1.62) and (1.36):

$$\begin{cases} \dot{u} = rv - qw + g\sin\theta \\ \dot{v} = pw - ru - g\cos\theta\sin\varphi \\ \dot{w} = qu - pv - g\cos\theta\cos\varphi + \frac{U_1}{m} \\ \dot{p} = \frac{I_{YY}-I_{ZZ}}{I_{XX}}qr + \frac{1}{I_{XX}}U_2 - \frac{J_m}{I_{XX}}q\Omega_R \\ \dot{q} = \frac{I_{ZZ}-I_{XX}}{I_{YY}}pr + \frac{1}{I_{YY}}U_3 + \frac{J_m}{I_{YY}}p\Omega_R \\ \dot{r} = \frac{I_{XX}-I_{yy}}{I_{ZZ}}pq + \frac{1}{I_{ZZ}}U_4 \\ \dot{\varphi} = p + \sin\varphi\tan\theta q + \cos\varphi\tan\theta r \\ \dot{\theta} = \cos\varphi q - \sin\varphi r \\ \dot{\psi} = \frac{\sin\varphi}{\cos\theta}q + \frac{\cos\varphi}{\cos\theta}r \\ \dot{z} = -\sin\theta u + \cos\theta\sin\varphi v + \cos\theta\cos\phi w \end{cases} \tag{1.63}$$

where $\Omega_R = \Omega_l + \Omega_r - \Omega_f - \Omega_b$.

1.6.2 Measurement model

The general measurement model for the EKF is described by Equation (1.9). We have to express all the sensor equations as functions of the state variables.

Accelerometer

While the equations for the gyroscope, the magnetometer and the barometer defined in Section 1.5 are already functions of the state variables, the equation of the accelerometer (1.49) is a function of the derivative of the linear velocity. However, we can substitute $\dot{u}$, $\dot{v}$ and $\dot{w}$ for their expressions in the dynamic equation (1.31):

$$\boldsymbol{y_a} = \boldsymbol{M_a}\boldsymbol{S_a}\left(\begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv + \frac{U_1}{m} \end{bmatrix} + \boldsymbol{b_a} + \boldsymbol{v_a}\right), \tag{1.64}$$

which is a function of the state variables. It is important to note that this equation contains also the control variable U_1. In the standard EKF measurement model, the control variables are not involved, but the filter should still work without modifications as long as the value of the variable is known without uncertainty. In any case, this particular situation must be taken into account during the implementation.

Complete model

The complete measurement model can be obtained by combining Equations (1.64), (1.50), (1.46) and (1.56):

$$\begin{cases} \boldsymbol{y_a} = \boldsymbol{M_a S_a} \begin{bmatrix} rv - qw \\ pw - ru \\ qu - pv + \frac{U_1}{m} \end{bmatrix} + \boldsymbol{b_a} + \boldsymbol{v_a} \\ \boldsymbol{y_g} = \boldsymbol{M_g S_g \omega} + \boldsymbol{b_g} + \boldsymbol{v_g} \\ \boldsymbol{y_m} = \boldsymbol{G R^T}(\varphi, \theta, \psi)\boldsymbol{m_f} + \boldsymbol{b_m} + \boldsymbol{v_m} \\ y_b = k_b P_0 e^{-\frac{g}{RT}z} + b_b + v_b \end{cases} . \tag{1.65}$$

1.6.3 Linearizations

As we said in Subsection 1.1.2, the extended Kalman filter requires both the state-transition and the measurement models to be linearized using the Jacobians.

Jacobian of the state-transition model

If we ignore the gyroscope effect, which is almost always negligible, the Jacobian of the state-transition model is a 10-by-10 sparse matrix[10]. Its nonzero elements are:

$$\begin{aligned} \boldsymbol{J}_{1,2} &= r; \\ \boldsymbol{J}_{1,3} &= -q; \\ \boldsymbol{J}_{1,5} &= -w; \\ \boldsymbol{J}_{1,6} &= v; \\ \boldsymbol{J}_{1,8} &= g\cos\theta; \\ \boldsymbol{J}_{2,1} &= -r; \\ \boldsymbol{J}_{2,3} &= p; \\ \boldsymbol{J}_{2,4} &= w; \\ \boldsymbol{J}_{2,6} &= -u; \\ \boldsymbol{J}_{2,7} &= -g\cos\theta\cos\varphi; \\ \boldsymbol{J}_{2,8} &= g\sin\theta\sin\varphi; \\ \boldsymbol{J}_{3,1} &= q; \\ \boldsymbol{J}_{3,2} &= -p; \\ \boldsymbol{J}_{3,4} &= -v; \\ \boldsymbol{J}_{3,5} &= u; \\ \boldsymbol{J}_{3,7} &= g\cos\theta\sin\varphi; \end{aligned}$$

[10] A matrix is sparse if more than an half of its elements are zeros. In this case, 60 of the 100 elements are zeros.

$$
\begin{aligned}
\boldsymbol{J}_{3,8} &= g\sin\theta\cos\varphi;\\
\boldsymbol{J}_{4,5} &= \frac{I_{YY}-I_{ZZ}}{I_{XX}}r;\\
\boldsymbol{J}_{4,6} &= \frac{I_{YY}-I_{ZZ}}{I_{XX}}q;\\
\boldsymbol{J}_{5,4} &= \frac{I_{ZZ}-I_{XX}}{I_{YY}}r;\\
\boldsymbol{J}_{5,6} &= \frac{I_{ZZ}-I_{XX}}{I_{YY}}p;\\
\boldsymbol{J}_{6,4} &= \frac{I_{XX}-I_{YY}}{I_{ZZ}}q;\\
\boldsymbol{J}_{6,5} &= \frac{I_{XX}-I_{YY}}{I_{ZZ}}p;\\
\boldsymbol{J}_{7,4} &= 1;\\
\boldsymbol{J}_{7,5} &= \sin\varphi\tan\theta;\\
\boldsymbol{J}_{7,6} &= \cos\varphi\tan\theta;\\
\boldsymbol{J}_{7,7} &= \cos\varphi\tan\theta q-\sin\varphi\cos\theta r;\\
\boldsymbol{J}_{7,8} &= \frac{\sin\varphi q+\cos\varphi r}{\cos^2\theta};\\
\boldsymbol{J}_{8,5} &= \cos\varphi;\\
\boldsymbol{J}_{8,6} &= -\sin\varphi;\\
\boldsymbol{J}_{8,7} &= -\cos\varphi r-\sin\varphi q;\\
\boldsymbol{J}_{9,5} &= \frac{\sin\varphi}{\cos\theta};\\
\boldsymbol{J}_{9,6} &= \frac{\cos\varphi}{\cos\theta};\\
\boldsymbol{J}_{9,7} &= \frac{\cos\varphi q-\sin\varphi r}{\cos\theta};\\
\boldsymbol{J}_{9,8} &= \frac{\tan\theta}{\cos\theta}(\sin\varphi r+\cos\varphi q);\\
\boldsymbol{J}_{10,1} &= -\sin\theta;\\
\boldsymbol{J}_{10,2} &= \cos\theta\sin\varphi;\\
\boldsymbol{J}_{10,3} &= \cos\theta\cos\varphi;\\
\boldsymbol{J}_{10,7} &= \cos\theta\cos\varphi v-\cos\theta\sin\varphi w;\\
\boldsymbol{J}_{10,8} &= -\cos\theta u-\sin\theta\sin\varphi v-\sin\theta\cos\varphi w.
\end{aligned}
$$

Jacobian of the measurement model

The jacobian of the sensor model is also a sparse matrix of the form:

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J_1} & \boldsymbol{J_2} & \boldsymbol{0_{3\times 4}} \\ \boldsymbol{0_{3\times 3}} & \boldsymbol{M_g S_g} & \boldsymbol{0_{3\times 4}} \\ \boldsymbol{0_{3\times 3}} & \boldsymbol{0_{3\times 3}} & \boldsymbol{J_m} \quad 0 \\ \boldsymbol{0_{1\times 9}} & J_a & \end{bmatrix}. \tag{1.66}$$

The submatrices that compose the above matrix are:

- $\boldsymbol{J_1}$: the derivatives of the accelerometer equation with respect to the linear velocity variables u, v, w;
- $\boldsymbol{J_2}$: the derivatives of the accelerometer equation with respect to the angular velocity variables p, q and r;
- $\boldsymbol{J_3}$: the derivatives of the magnetometer equation with respect to the euler angle variables φ, θ and ψ;
- J_a (scalar): the derivative of the altimeter equation with respect to the altitude variable.

Their expressions are:

$$\boldsymbol{J_1} = \boldsymbol{M_a S_a} \begin{bmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{bmatrix}; \tag{1.67}$$

$$\boldsymbol{J_2} = \boldsymbol{M_a S_a} \begin{bmatrix} 0 & -w & v \\ w & 0 & -u \\ -v & u & 0 \end{bmatrix}; \tag{1.68}$$

$$\boldsymbol{J_3} = \begin{bmatrix} \boldsymbol{G}\frac{\partial \boldsymbol{R^T}(\varphi_0,\theta_0,\psi_0)}{\partial \varphi}\boldsymbol{m_f} & \boldsymbol{G}\frac{\partial \boldsymbol{R^T}(\varphi_0,\theta_0,\psi_0)}{\partial \theta}\boldsymbol{m_f} & \boldsymbol{G}\frac{\partial \boldsymbol{R^T}(\varphi_0,\theta_0,\psi_0)}{\partial \psi}\boldsymbol{m_f} \end{bmatrix}; \tag{1.69}$$

$$J_a = -\frac{k_a P_0 g}{RT} e^{-\frac{g}{RT}z}. \tag{1.70}$$

Chapter 2

Simulation model

This chapter contains the description of the Simulink model developed to test the performance of the attitude control system based on the extended Kalman filter. Our work actually extends a pre-existing model developed by Tommaso Bresciani for his master thesis.

The first section of this chapter describes the pre-existing model, Section 2.2.1 regards the model of the on board sensors (which was missing in the model by Bresciani) and finally Section 2.3 describes the model of the EKF both in continuous and discrete time.

The simulink models of the system and its blocks are shown here, while the Matlab code used to implement some blocks is listed in Appendix B.

2.1 Pre-existing model

The higher-level structure of the pre-existing model is shown in Figure 2.1. The main signals that connect the blocks are:

`pos-vel-acc`: this signal groups together the information about the linear and angular position, velocity and acceleration of the aircraft, so it is actually a mux of 8 vector signals, each one of which has three components.

`omega`: this signal is a vector of four components that represent the angular rates of the propellers.

`Vcontrol`: this signal is a vector of four components that represent the current voltages generated by the control to power the engines.

`target`: this signal is generated by the user and contains the information about the desired behaviour of the quadrotor. It has four components: roll, pitch, yaw and altitude.

Here is a brief description of the blocks:

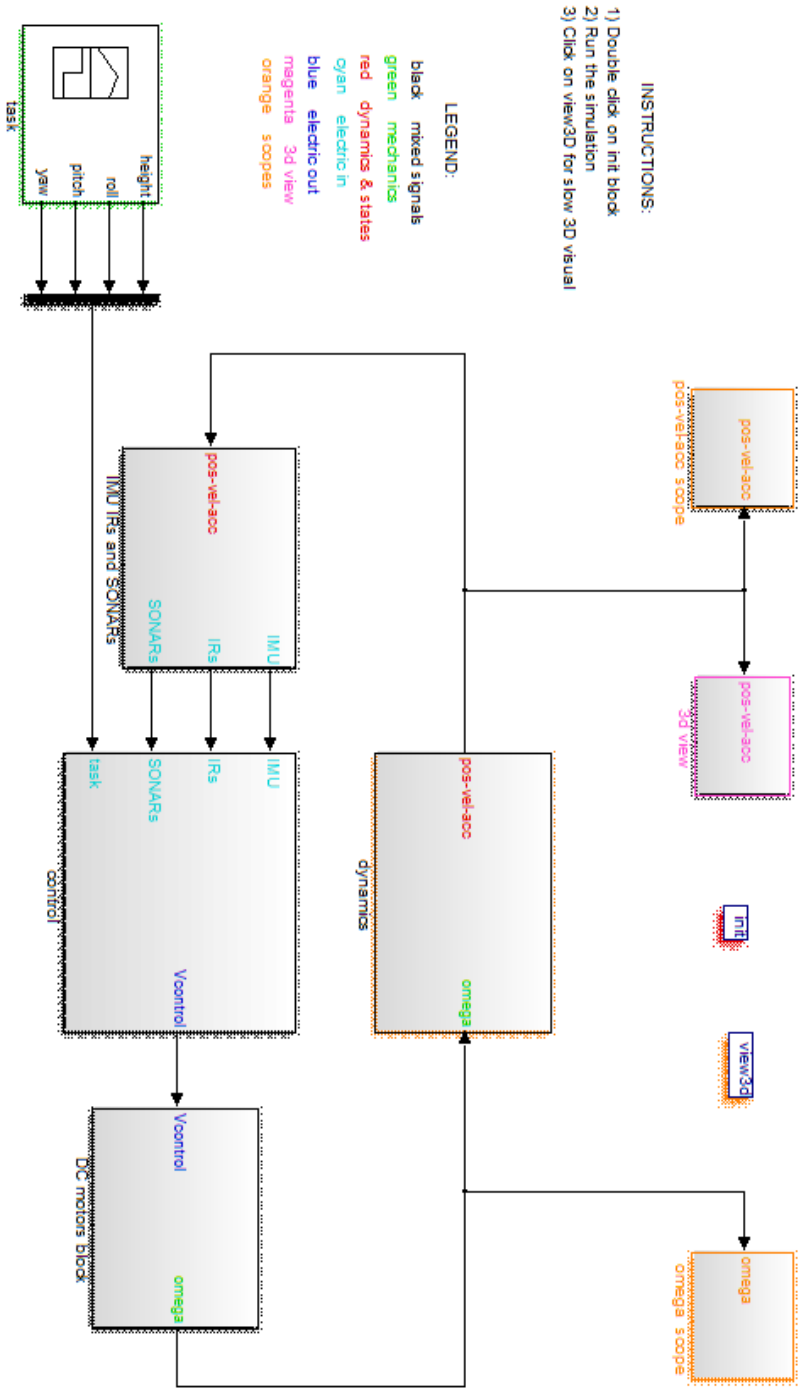


Figure 2.1: Pre-existing Simulink model.

`dynamics`: this block models the dynamics of the quadrotor as a physical system, taking the angular rates of the propellers as input (signal `omega`) and outputs the position, velocity and acceleration `pos-vel-acc`. The block is analyzed in detail in Subsection 2.1.1.

`IMU, IRs and SONARs`: this block contains the model of the sensors. In practice, the block is just a stub, because the behaviour of the sensors is not actually modeled, but parts of the input signal `pos-vel-acc` are perturbed with noise and directly passed to the output. Moreover, the infrared sensors (IRs) and the sonars are not even used by the control system.

`control`: this block implements the control algorithm. It takes the `target` signal and the output of the sensors as input and returns the voltages to be passed to the motors to obtain the desired behaviour (signal `Vcontrol`). The block is analyzed in Subsection 2.1.2.

`DC motors block`: this block models the engines that power the four rotors. It takes the voltages `Vcontrol` as input and outputs the angular speeds of the rotors (`omega`). This block is analyzed in Subsection 2.1.3.

task: this is a *signal builder*, a type of block that allows to define a signal with a grahical interface. In this case, the signal builder is used to specify the target values of the roll, pitch, yaw and altitude as functions of time.

`3D scope`: the code of this block draws a tridimensional view of the quadrotor that is updated in real time. The resulting animation is very useful for an intuitive understanding of the aircraft motion. The code that draws the quadrotor is listed in Listing B.15.

`pos-vel-acc scope`: this block groups many scopes that plot the components of the signal `pos-vel-acc` versus time. It can be used to monitor the whole dynamic system.

`omega scope`: this block is a scope that plots the angular velocities of the rotors versus time.

Before starting any simulation, the initialization script `init.m` must be executed. It sets all the global variables and constants used by the model blocks. The code of the script is listed in Listing B.1.

2.1.1 Dynamics block

The Simulink model of the `dynamics` block is shown in Figure 2.2. The main block is `dynamic_system`, which contains the matlab code that implements the dynamic equations (1.63) (Listing B.4). The integrator integrates the equations starting from the initial conditions specified as parameters in the initialization script. The `routing` block selects the right components of the state and its integral to generate the signal `pos-vel-acc`.

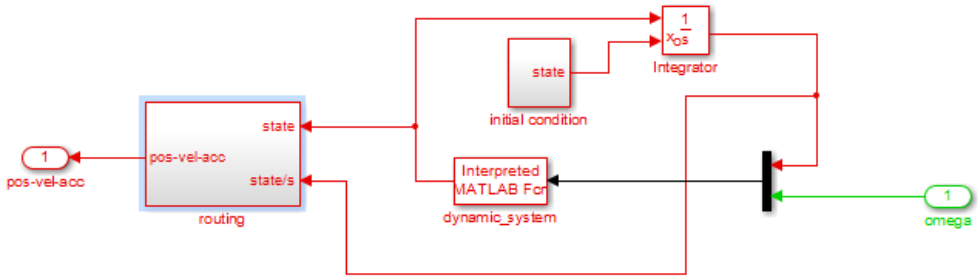


Figure 2.2: Simulink model of the `dynamics` block.

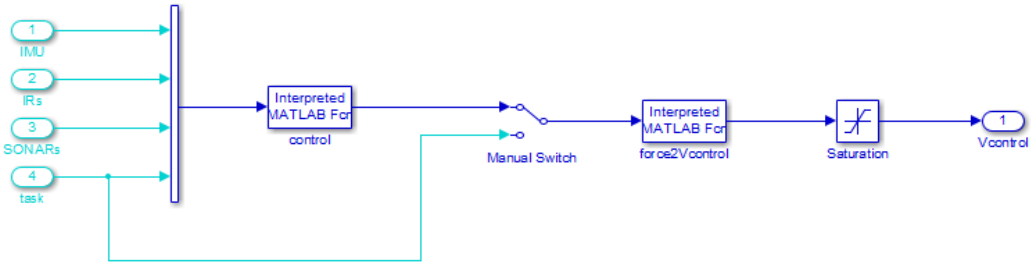


Figure 2.3: Simulink model of the `control` block.

2.1.2 Control block

The Simulink model of the `control` block is shown in Figure 2.3. The block called `control` contains the Matlab code that implements the logic of the proportional-derivative (PD) controller (Listing B.2. The block `force2Vcontrol` (code in Listing B.3) converts the output of the controller to the corresponding voltage values to be passed to the motor block. A saturator is used to avoid out-of-range voltage values. The manual switch inserted between the `control` block and the `force2Vcontrol` block can be used to send the `task` signal directily to the motors, excluding the control action.

2.1.3 DC motors block

This block is composed by four sub-blocks of the same type, one for each motor. The Simulink model of the sub-blocks is shown in Figure 2.4. The signal `Tload`, which represents the torque load of the rotor, is set to a constant. One problem of this motor block is that the output values of the rotor angular rates do not match exactly the ones expected by the controller. In other words, if the input signal `Vcontrol` is set to a value that should correspond to the hovering condition (the thrust exactly

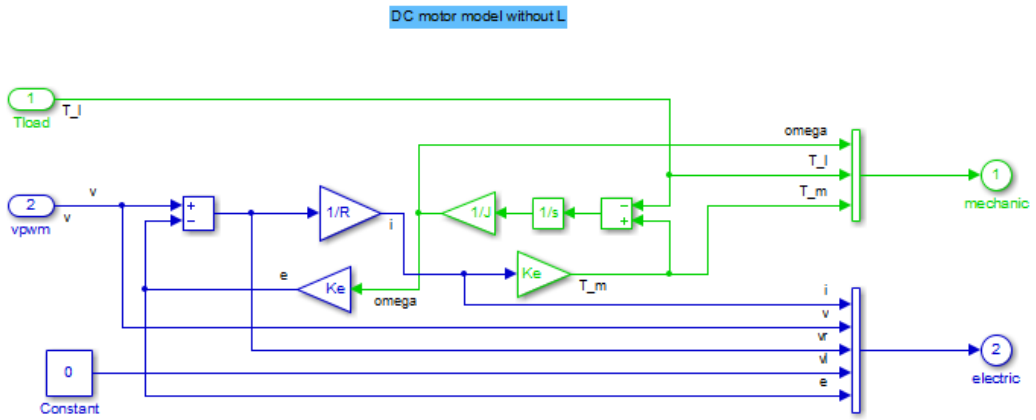


Figure 2.4: Simulink model of the `DC motors` block.

compensate the gravity), the actual `omega` output produced by the block is smaller than the correct one, so the simulated quadrotor falls down.

2.2 IMU model

Before extending the model with the implementations of the sensors and the the EKF, I applied some modifications to the pre-existing parts:

- the blocks that execute interpreted Matlab code have been replaced by compiled code blocks to improve their performance;
- the signal `pos-vel-acc` has been split into four simpler signals;
- the `control` block has been modified to directly output the control signal (not transformed to the voltage control required by the motors) and the angular rates of the propellers;
- a switch has been added to exclude the `DC motors` block.

This last two modification require a further explanation: since the state-transition model used for the EKF in eq. (1.63) contains the control variables U_i, the block that implements the filter must know its value from the controller. As we said in Subsection 2.1.3, the motor block outputs a value of `omega` that is biased with respct to the one expected by the controller. In the pre-existing model, the contoller itself can compensate for this bias because it knows the actual position and velocity of the quadrotor, but the Kalman filter cannot operate correctly if its predictions are based on control variables that do not correspond to the real ones, because it is designed to correct the random errors in the model, not a fixed bias.

Figure 2.5 shows the updated model, with the new IMU block and the ones related to the discrete and continuous time versions of the EKF. The upper part of the model is not visible, but it is substantially unchanged.

2.2.1 IMU blocks library

The IMU of the quadrotor is modeled according to the equations presented in Section 1.5. Some blocks inside the IMU model have been reused multiple times, so I put them in a dedicated library.

Three-axis MEMS sensor

The block that models a generic three-axis MEMS sensor according to eq. (1.45) is the most relevant of the ones included in the library. It has been used to model all the sensors of the IMU excepts the barometer, which measures a scalar quantity. This sensor block does not transform the measured physical quantity into a voltage scale like a real sensor, the output instead has the same dimensions as the input, so an implicit reconversion from the voltage to the corresponding physical quantity is assumed. The block is composed by three subsystems of the same type, one for each axis. The subsystem is shown in Figure 2.6: the signal from the main axis is summed with the interferences from the other axes (misalignment factors), then the result is multiplied by a global scale factor and a bias is added to the result. The signal is then perturbed with Gaussian noise, optionally clipped (if `isClipped` is true) and delayed. The main parameters of this block are:

- the bias on each axis;
- the variance of the Gaussian error on each axis;
- the upper and lower saturation limit (maximum and minimum values that can be output by the sensor);
- the output delay.

The advanced parameters (listed in a different tab of the mask associated to the Simulink block) are:

- the gains of the three axes (corresponding to the values S_i in eq. (1.45));
- the misalignment factors of each axis on each other (corresponding to the values M_{ij} in eq. (1.45);
- the seeds used by the random generator that produces the noise on each axis.

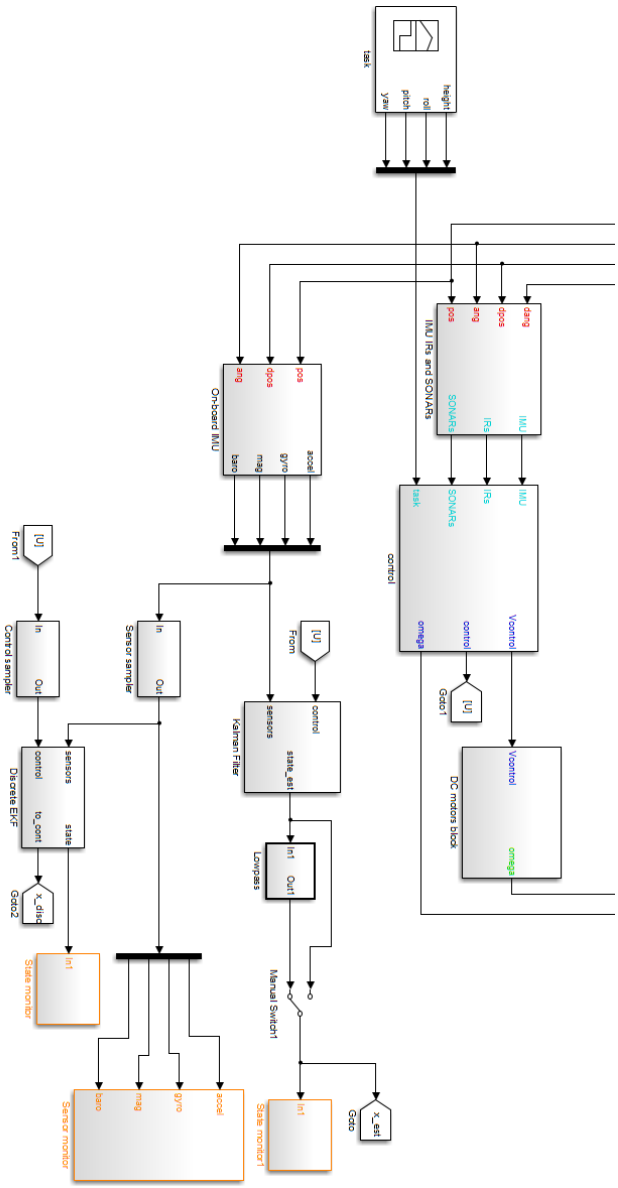


Figure 2.5: Partial view of the updated Simulink model.

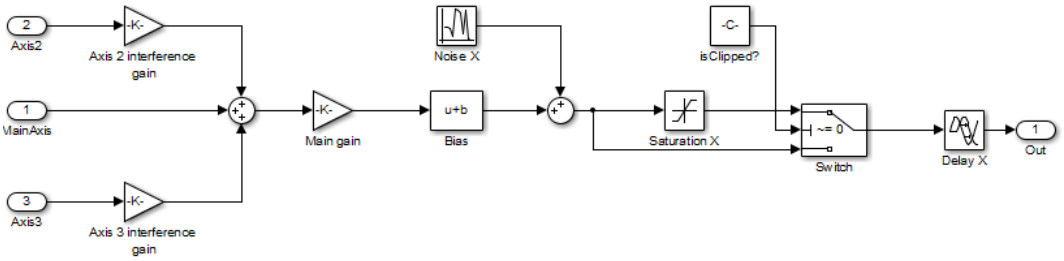


Figure 2.6: Simulink model of a generic MEMS sensor.

Other blocks in the library

The other blocks contained in the library are:

Scalar sensor: this block is used to model the barometer, it is very similar to the one described in the previous paragraph for the three-axis sensor, but the input and output signals are scalars and obviously there are no misalignment factors;

Earth to body frame: this block converts a vector expressed in the earth frame to its equivalent in the body frame, given the Euler angles. The block applies the matrix defined in eq. (1.20), the Matlab code is in Listing B.5;

Euler to pqr conversion: this block takes the instantaneous orientation of the quadrotor in Euler angles as input and computes its angular velocity with respect to the body frame. The conversion is performed by applying the matrix defined in eq. (1.27), the Matlab code is in Listing B.6.

2.2.2 Structure of the IMU subsystem

The complete Simulink model of the quadrotor on board IMU is shown in Figure 2.7. The subsystems are:

`Real IMU`: this subsystem contains the sensor blocks. The input is composed by the quantities that the sensors need for their measurements (all expressed in body frame): the linear velocity, the angular rates, the gravity vector, the magnetic filed vector and the atmosferic pressure. The acceleration measured by the accelerometer is not needed as an input, because it is computed internally as the time derivative of the velocity plus the gravity (following the definition of proper acceleration given in Subsection 1.5.2);

`Body frame velocity and angular rates`: this subsystem computes the linear and angular velocity in the body frame given the velocity in the earth frame and the Euler orientation in Euler angles. The subsystem operates the conversion using the library blocks described in the previous subsection.

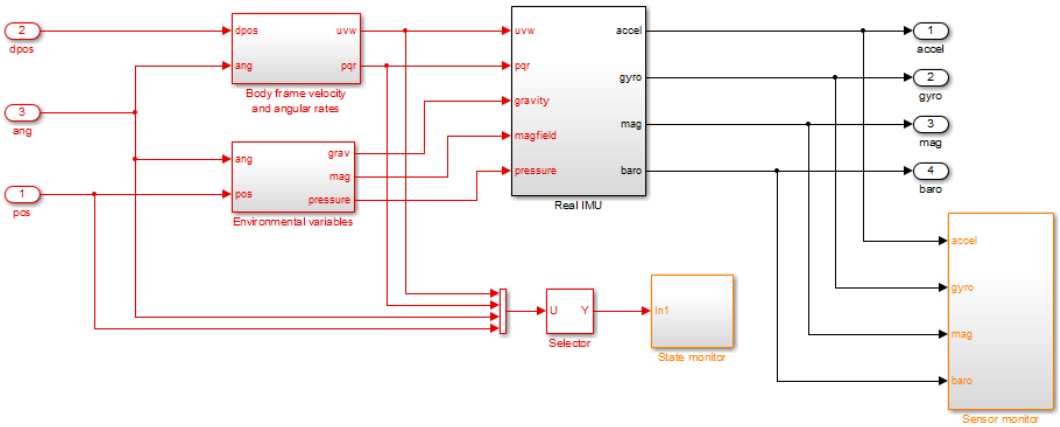


Figure 2.7: Simulink model of the IMU block.

`Environmental variables:` this subsystem generates the values of the atmosferic pressure, the magnetic field and the gravitational acceleration with respect to the body frame:

- the atmosferic pressure is computed according to eq. (1.54), as a function of the altitude z (code in Listing B.7);
- the magnetic field vector is converted in the body frame and perturbated with the hard and soft iron distortions:
- the gravitational acceleration vector (assumed to be constant) is just converted in body frame.

The IMU block involves a lot of parameters, notably:

- the bias, scaling, error covariances and misalignment factors of all the sensors;
- the local values of the magnetic field, the ground pressure and the temperature;
- the hard and soft iron distortions (which depend on the physical structure of the quadrotor).

If all these parameters are properly set (in the initialization file `init.m`, the simulation of the sensors should be accurate enough for our purposes[1]. The two orange blocks in Figure 2.7 are sets of scopes that are useful to check the behaviour of the system: `sensor monitor` monitors the output of the sensors, `state monitor` monitors the state variables of the Kalman filter (if everythig works as expected, the output of the filter should be very similar to the state shown by these scopes).

[1] A realistic MEMS sensor simulation would require a much more complex model that takes into account other properties of the electro-mechanical system, like the impulse response and the nonlinearities in the operating range.

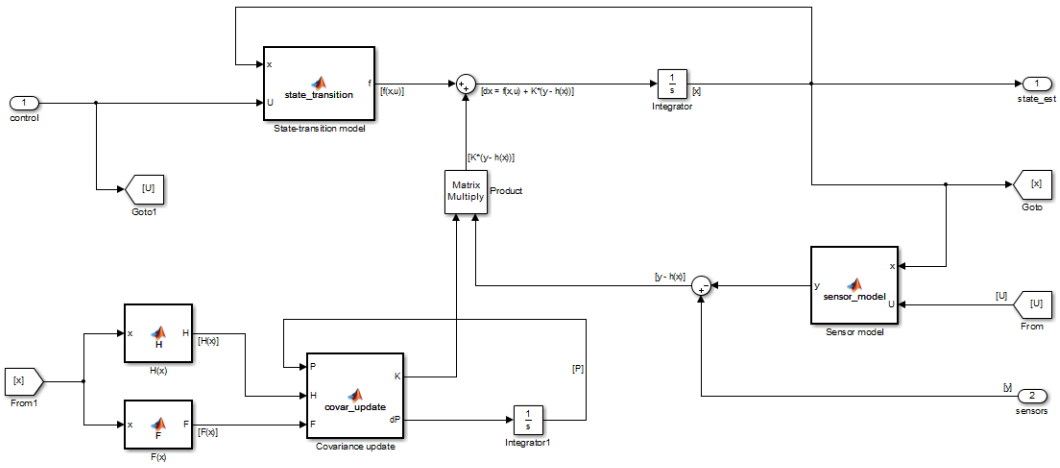


Figure 2.8: Simulink model of the continuous time EKF.

2.3 Kalman filter model

The implementation of the extended Kalman filter in both the continous and discrete time versions is the core of my work in Simulink. The most important parameters that influence the behaviour of the filter are the initial estimates of the expected value and the covariance of the state vector and the matrices Q and R, which represent covariances of the process and the measurement models respectively. The filter implementation must also know many other quantities related to the system and sensor models (mass of the quadrotor, biases of the sensors etc.), these paramenters used in the filter can be initialized independently from the corresponding one used in the real-world simulation, so it is possible to test the performance of the EKF when some of this values are not correctly estimated.

2.3.1 Continuous time model

The Simulink model of the continuous time EKF is shown in Figure 2.8. The input of the system is the output of the IMU block and the control variables. The blocks are explained below, the reference to the Matlab code in Appendix B is in brackets:

`State-transition model`: this block implements the system in Equation (1.63), without the negligible terms related to the gyroscope effect (the ones containing the terms with Ω_r. (Listing B.8);

`Sensor model`: this block implements the system in Equation (1.65) (Listing B.9);

`Covariance update`: this block computes both the covariance update according to eq. (1.11) and the Kalman gain according to eq. (1.12) (Listing B.10);

`F and H`: these two blocks compute the Jacobians of the state-transition model and the sensor model respectively around the current estimate, the structure of

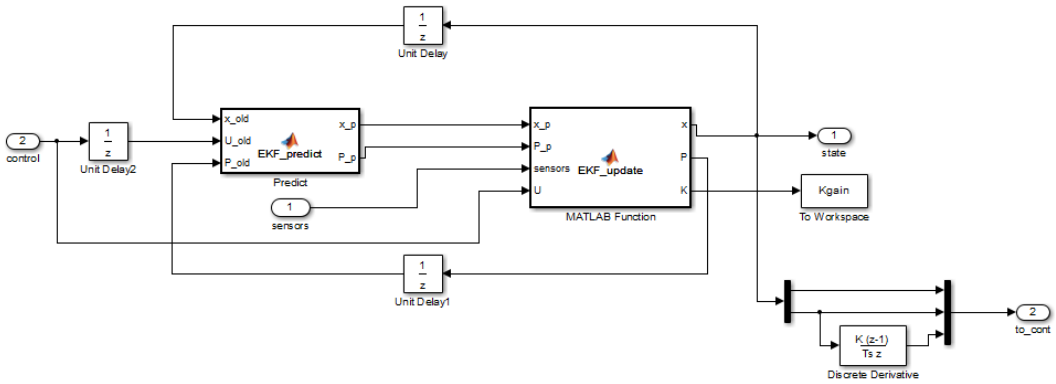


Figure 2.9: Simulink model of the discrete time EKF.

these sparse matrices is described in detail in Subsection 1.6.3 (Listing B.11 and Listing B.12).

The output of this subsystem is the estimate of the instantaneous value of the state variables at any time (since it is continuous, there are no time steps).

2.3.2 Discrete time model

The continuous time model proposed in the previous paragraph is useful to test the best performance achievable with the EKF, but it is not suitable for a digital implementation on the quadrotor, because digital systems always work in the discrete domain and the MEMS sensors themselves have a limited maximum output frequency[2]. In order to develop a model that can be useful for practical applications, the discrete time version of the EKF must be used.

Discretization of the model

The mathematical discipline that studies the algorithms to solve ordinary difference equations (ODEs) that do not admit an analytical solution is a part of the *numerical analysis*. There are diffrent methods to discretize a system of differential equations, from the very simple *forward Euler* to the most sophisticated ones, like the higher order Runge-Kutta methods, but all of these algorithms involve the intgration over discrete time steps (either fixed or variable). We will use the forward Euler metohd to discretize the state-transition model.

[2] Matlab and Simulink are computer programs themselves, so when a continuous time model is simulated, the system actually solves it using some sophisticated numerical algorithm that involves discretization.

Forward Euler. The forward Euler is the simplest numerical algorithm to solve an ordinary differential equation. Given the ODE:

$$\dot{y} = f(t, y(t))$$

we can approximate the derivative as:

$$\dot{y} \approx \frac{y(t+h) - y(t)}{\Delta t},$$

then by expliciting $y(t+h)$ and substituting $\dot{y}$:

$$y(t+h) = y(t) + f(t, y(t)) \cdot \Delta t. \tag{2.1}$$

In practice, a fixed time step h is chosen and the function in eq. (2.1) becomes a sequence:

$$y_{n+1} = y_n + f(t_n, y_n) \cdot \Delta t. \tag{2.2}$$

where t_n is an element of the sequence of sampling time instants $t_0, t_0+h, ..., t_0+nh$: In our case, we can trasform the Equation (1.63), which is in the form of eq. (1.13) to the form of eq. (1.14) by imposing:

$$\Phi_{k-1}(x_{k-1}, u_{k-1}) = y(t_{k-1}) + f(y(t_{k-1}), u(t_{k-1})) \cdot \Delta t. \tag{2.3}$$

The forward Euler method is not widely used, because it is less accurate and converges more slowly that the other algorithms, so a smaller time step (and a larger number of steps) is required to obtain a good solution. However, we are working on a system that has intrinsec uncertainty, so as long as the error introduced by the discretization is smaller than the one introduced by the plant and the measurement noieses, it can be handled by the extended Kalman filter.

Sensor sampling

The sensor model of the EKF is quite easy to discretize: susbstituting the function $x(t)$ with the values x_k at the sampling time instants t_k is enough. The discretization of the continuous time signals from the real IMU model is more complex: the well-known Nyquist-Shannon sampling theorem states that a signal can be sampled at the sampling rate $1/Ts$ without aliasing problems if and only if its frequency components are below the *Nyquist frequency* $f_N = 1/2Ts$. The practical consequence of this theorem is the necessary application of adequate lowpass filters to every signal that needs to be downsampled, these filters are called *anti-aliasing filters*.

The subsystem called `Sensor sampler` in Figure 2.5 applies an anti-aliasing FIR filter to the signals from the sensors and samples the filtered signal at the sampling time Ts using a zero-order hold method (the output signal is constant between time steps). The parameters of the anti-aliasing filter can be specified in the initialization file, the one that I used for the simulations is a second order Butterworth filter with cutoff frequency $0.8/Ts$.

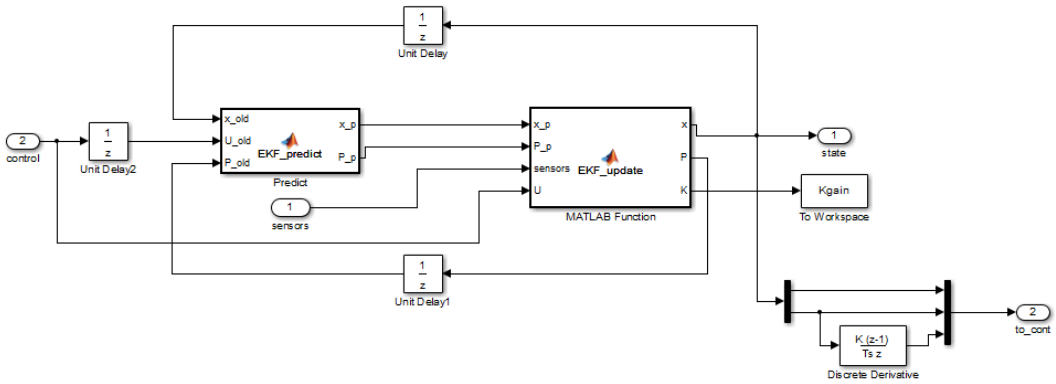


Figure 2.10: Simulink model of the discrete time EKF.

Control signal. The control signal, which contains the instantaneous values of the control variables U_k, is also filtered and sampled with the same anti-aliasing filter (and obviously the same sampling time) as the signals from the sensors.

System structure

Figure 2.9 shows the Simulink model of the discrete time EKF. The structure is simple: one block performs the prediction step (equations (1.3) and (1.4)) and the other one performs the update step (equations (1.5) and (1.6)), computing also the Kalman gain (eq. (1.7))[3]. The discrete state-transition and sensor models and their Jacobian matrices are computed in the same two blocks to obtain a simpler structure. This semplification is possible in the discrete time version because the state-transition model and the related Jacobian are needed only by the prediction step while the sensor model and its Jacobian are needed only by the update step (the two steps are not distinguished in the continuous time version). The Matlab code of the predict and update blocks is listed in Listing B.13 and Listing B.14 respectively.

The output `to_cont` on the bottom left of Figure 2.9 sends the state to the control system. Since the controller needs also the vertical speed, which is a state variable, it is computed as the derivative of the altitude z and added to the state vector.

[3] The equations in brackets actually describe the linear version of the Kalman filter, the necessary modifications to apply them to the extended version are described in Subsection 1.1.2.

Chapter 3

Testing and performance analysis

In this chapter, we analyze the performance of the Simulink implementation of the EKF extended Kalman filter.

The Simulink model has a large number of parameters that can affect the behaviour of the simulation. The default values of the parameters are listed in Table 3.1, these values are used unless otherwise specified and correspond to the ones in Listing B.1. If an experiments is performed with some parameters set to a different values, the ones that differ are listed in the description of the experiment.

3.1 Open loop performance analysis with simulated data

In the experiments presented in this section, the controller is not fed with the output of the filter, but with the actual state, so the actual trajectories are close to the target ones and the predictions of the filter can be easily compared to the real evolution of the state. Moreover we assume no calibration errors in the sensor and measurement models of the Kalman filter, i.e., all the parameters used by the filter have the same values of the corresponding ones used for the real system simulation.

The trajectories covered in this section are the following:

- hovering (no movement);
- altitude variation (without horizontal movements);
- horizontal movement along the X axis (actually a pitch angle variation);
- simple rotation on the Z axis (yaw);

[1]The magnetic field is specifies only as a versor, its modulus is ignored. The versor in the example corresponds to the orentation of the field in Milan, according to the website http://www.ngdc.noaa.gov/geomag-web/.

Physical constants	
Magnetic field versor[1]:	$[0.48, -0.01, -0.87]$
Gravitational acceleration:	9.81 m/s^2
Pressure at ground level:	101325 Pa
Dry air specific gas constant:	286.9 $J/(kg \cdot K)$
Air temperature:	283.15 K
Mechanical parameters	
Mass of the quadrotor:	0.5 kg
Thrust factor:	7.8^{-5} $(m \cdot kg)/(s^2 \cdot rad^2)$
Drag factor:	1.1^{-5}
Diagonal elements of the inertia matrix:	5^{-3} $kg \cdot m^2$, 5^{-3} $kg \cdot m^2$, 9^{-3} $kg \cdot m^2$
Distance between the motors and the center of mass:	0.25 m
Sensor parameters	
Accelerometer error variance (all axes):	0.01
Gyroscope error variance (all axes):	0.01
Magnetometer error variance (all axes):	0.01
Barometer error variance:	10
Kalman filter parameters	
Initial state estimate covariance:	$0.001 \cdot \boldsymbol{I_{10}}$
Measurement noise covariance:	corresponds to the actual sensor noise variances.
Process noise covariance:	$0.001 \cdot \boldsymbol{I_{10}}$
Sampling time for discrete EKF:	0.05 s (20 Hz)
Anti-aliasing filter for sensor discretization:	Butterworth order 1, cutoff 8 Hz

Table 3.1: Default parameters used in the simulations.

Variable	Continuous EKF	Discrete EKF
Velocity on X $[m/s]$	0.0013	-0.017
Velocity on Y $[m/s]$	-0.0189	-0.0687
Velocity on Z $[m/s]$	2.749^{-5}	7.637^{-5}
Roll angular rate $[rad/s]$	9.777^{-4}	8.535^{-4}
Pitch angular rate $[rad/s]$	1.732^{-4}	4.656^{-4}
Yaw angular rate $[rad/s]$	-5.238^{-5}	-6.988^{-5}
Roll angle $[rad]$	1.759^{-4}	9.347^{-4}
Pitch angle $[rad]$	3.112^{-5}	-3.623^{-4}
Yaw angle $[rad]$	-6.35^{-4}	-0.002
Altitude $[m]$	0.002	0.002

Table 3.2: Mean values of the estimated variables (hovering).

- an example of complex trajectory, where the quadrotor performs many linear and angular movements at the same time.

This set of trajectories may seem too limited with respect to the large number of possible manouvers, however the experiments show that the performance of the filter are similar (and quite good) in all this cases, so further tests seemed unnecessary at this stage.

3.1.1 Hovering

The simplest flight trajectory for a quadcopter is the hovering: in this case, the aircraft just mantains a fixed position in the air. The value of all the state variables is expected to be zero and not to change in time.

Figure 3.1 shows the predictions of both the continuous time (black) and discrete time (blue) EKF about the linear velocity components and the angular positions (the other state variables are not shown, because they are less relevant in this case). As expected, the estimates of the discrete time filter are less accurate, because it uses only a subset of the sensor and control data. However, the mean values of the estimates are more significative than the local error, because the variables are integrated over time to determine the behaviour of the system and in particular the position is known only as the integral of the velocity. Table 3.2 contains the mean values over time of the estimated state variables from both the implementations of the filter.

The linear velocity components on X and Y are the variables whose mean value differs more from zero in both the discrete and continuous time filter estimates, because they cannot be directly measured, but only inferred by integrating the data from the accelerometers. In particular, the estimate of the Y component by the discrete filter is particulary poor, because it settles on the value $-0.1\ m/s$, which is not negligible, expecially for indoor flight applications. According to the dynamic model, a negative acceleration on the Y axis is generated by a positive roll angle: as

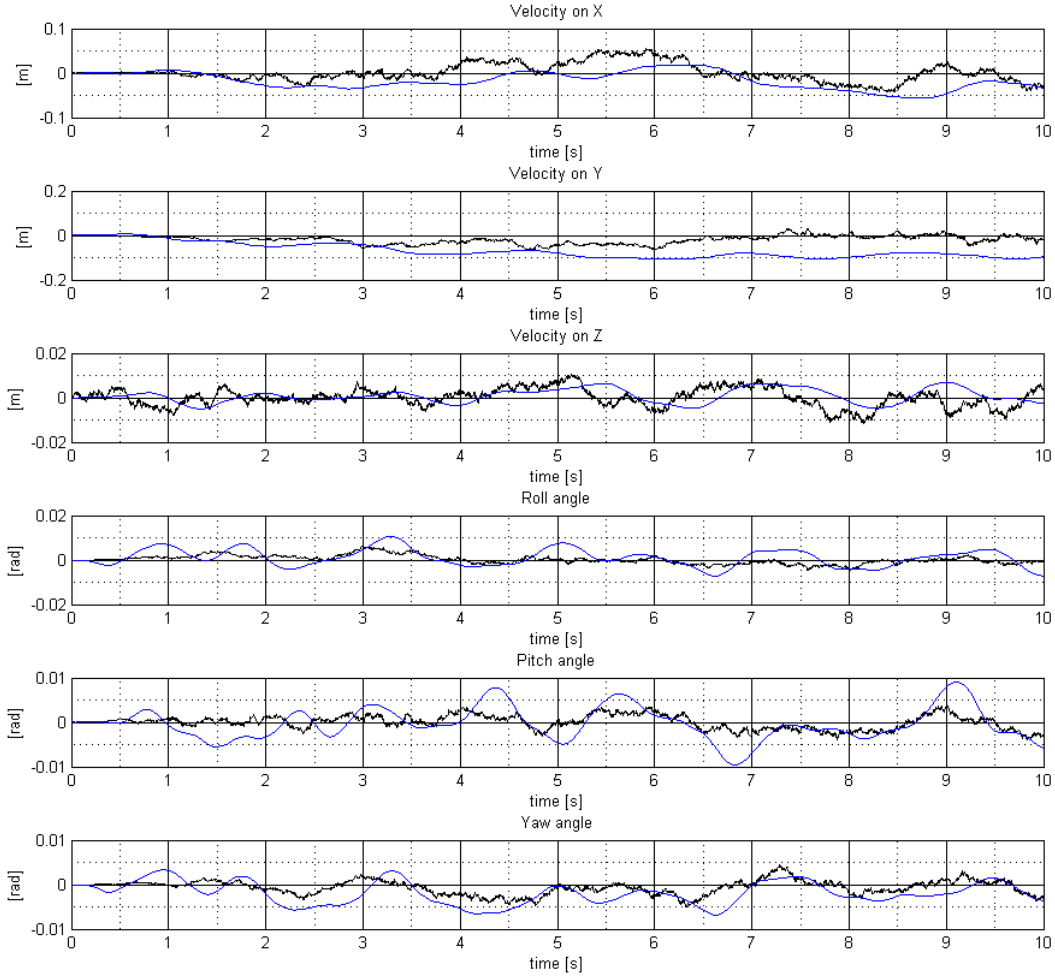


Figure 3.1: Hovering simulation

expected, the mean value of the estimated roll angle by the discrete time filter is much bigger than the one estimated by the continuous time filter. This error, however, is not systematic, because reproducing the simulation with different random seeds for the sensor noises leads to totally different results. The cause may be a small bias randomly introduced in the noise by the discretization process.

Another variable that the discrete time filter estimates with much less accuracy than the continuous one is the yaw angle, however this error is not likely to be a problem, because a bias of 0.002 *rad* is still irrelevant for the common practical applications.

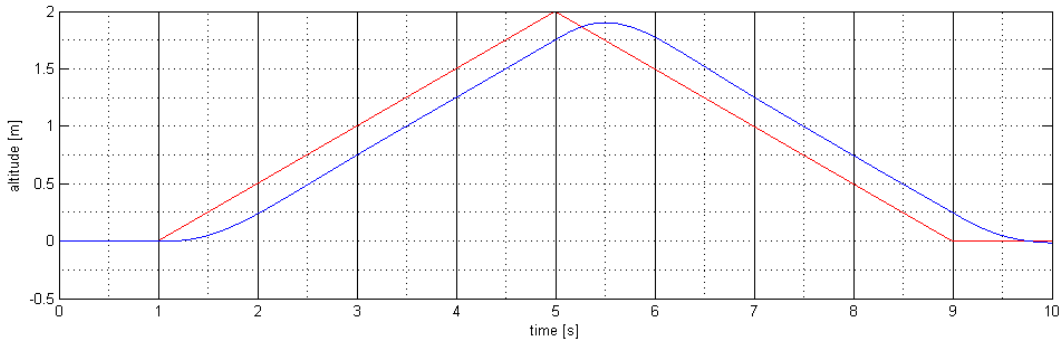


Figure 3.2: Altitude variation: the desired trajectory (red) and the actual one (blue).

3.1.2 Altitude variation

In this simulation, the target trajectory is a simple altitude variation: the quadrotor goes up for two meters and returns to the initial position. The desired trajectory and the actual one are shown in Figure 3.2, in red and blue respectively. As the figure shows, the PD controller lacks integral action, so it is not able to follow the ramp signal in real time and reach the maximum point.

Figure 3.3 compares the actual trajectory (the dashdotted red line) and the estimates of the two filters. While the continuous time filter estimate matches almost exactly the trajectory, so the red line is not even visible, the discrete time filter estimate is also very accurate, but it has a delay proportional to the order of the anti-aliasing filter. The magenta line shwos the result of a first attempt with a Butterworth filter of order 4, since the delay of more than 0.25 s appeared too large, the filter was reduced to order 1 (the one specified in Table 3.1, that will be used for all the simulations from now on).

3.1.3 Linear movement

The third simple trajectory analyzed here is a limited horizontal movement along the X axis. The movement along the X direction is produced by tilting forward the quadrotor for a short time: the positive pitch angle produces an acceleration that increases the velocity. The movement is then stopped by a backward tilt of the pitch angle of the same amount and duration of the forward one, which produces a corresponding negative acceleration. As shown in Figure 3.4, the controller is not able to follow the fast ramps of the target signal, so the maximum of the actual pitch angle is only half of the desired one.

Figure 3.5 compares the actual trajectory and the estimated values of the relevant state variables. The actual trajectory is represented as a red dashdotted line, the continuous time filter estimate is in black and the discrete time estimate is in blue. The resulting linear velocity is about $2m/s$ and the quadrotor moves forward for about 14 meters. The presence of a linear velocity on the Z axis when the pitch

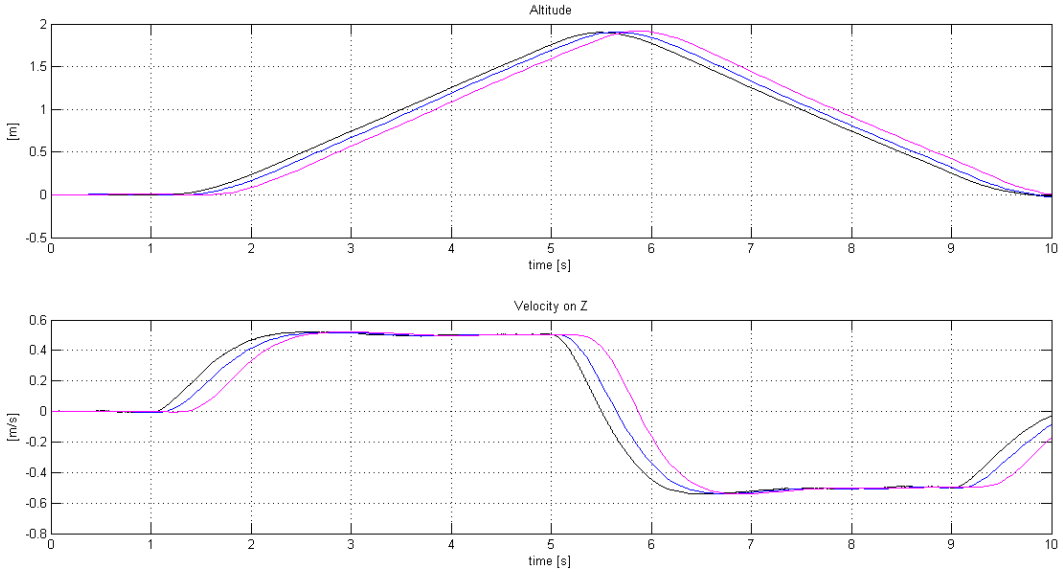


Figure 3.3: Altitude variation: the actual trajectory (red), the continuous time estimate (black), the discrete time estimate with an anti-aliasing filter of order 1 (blue) and the discrete time estimate with an anti-aliasing filter of order 4 (magenta) of the relevant state variables.

angle is different from zero does not imply a vertical movement: remembering that the state variables are expressed in the body frame, the velocity on Z is $w = \sin v_x$, where v_x is the linear velocity on the X axis of the earth frame. As for the altitude change experiment, the continuous time filter estimate is very accurate, while the discrete time estimate is also quite accurate, but it is delayed by the anti-aliasing filter.

3.1.4 Rotation around the Z axis

In this simulation, the target trajectory is a sequence of rotations on the Z axis: a clockwise rotation of one radian between two smaller counterclockwise rotations of 0.2 radians. As shown in Figure 3.6, the actual trajectory produced by the controller is significantly different with respect to the target one: it is sinusoidal, the amplitude is smaller and even the phase is delayed, so that at the end of the simulation the quadrotor is still at the maximum of the conterclockwise rotation, while it should have returned to the initial position.

The actual trajectories and the ones estimated by the continuous and discrete time version of the filter for the yaw angle and angular velocity are shown in Figure 3.7. The results are similar to the ones described in the previous experiments: the continuous time filter estimate is so accurate that the red line which marks the actual trajectory is not visible, the discrete time estimate is delayed as an effect of the anti-aliasing filter.

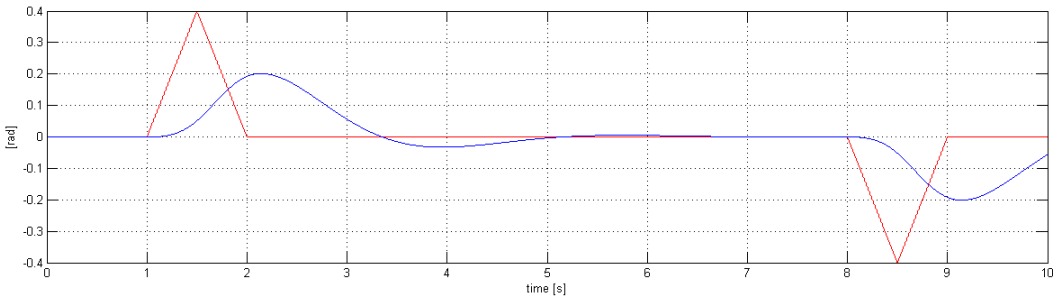


Figure 3.4: Linear movement: the desired trajectory (red) and the actual one (blue) of the pitch angle.

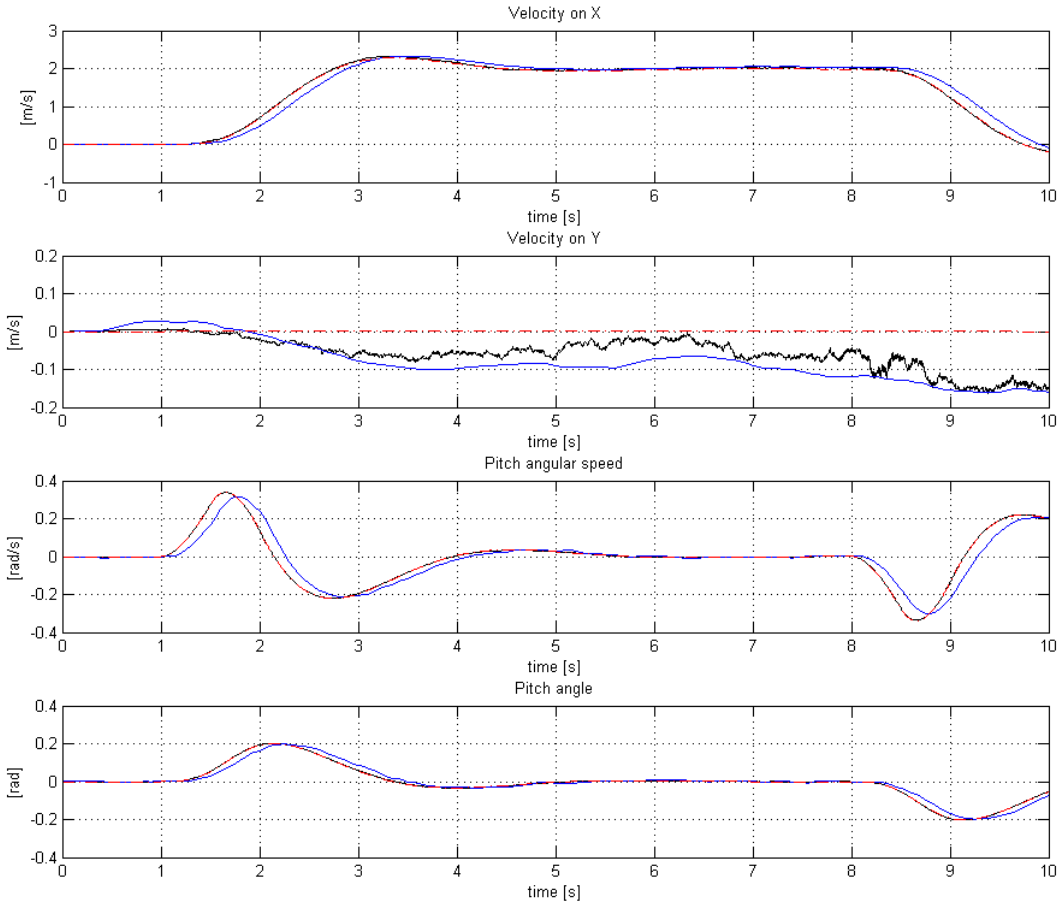


Figure 3.5: Linear movement: the actual trajectory (red), the continuous time estimate (black) and the discrete time estimate (blue) of the relevant variables.

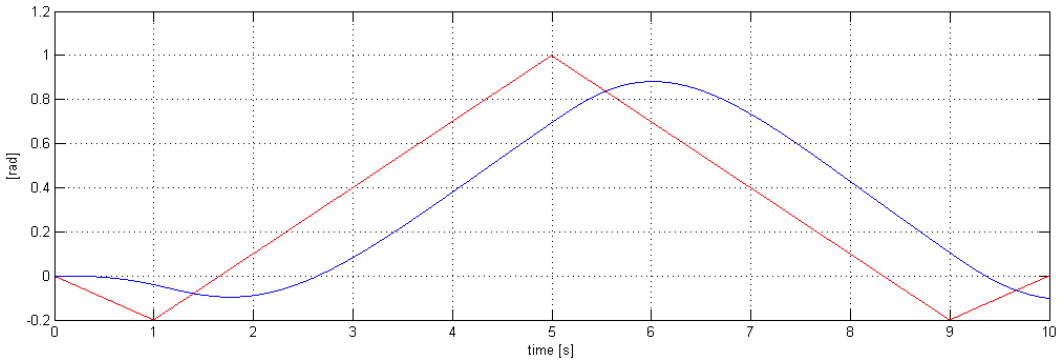


Figure 3.6: Rotation around the Z axis: the desired trajectory (red) and the actual one (blue) of the yaw angle.

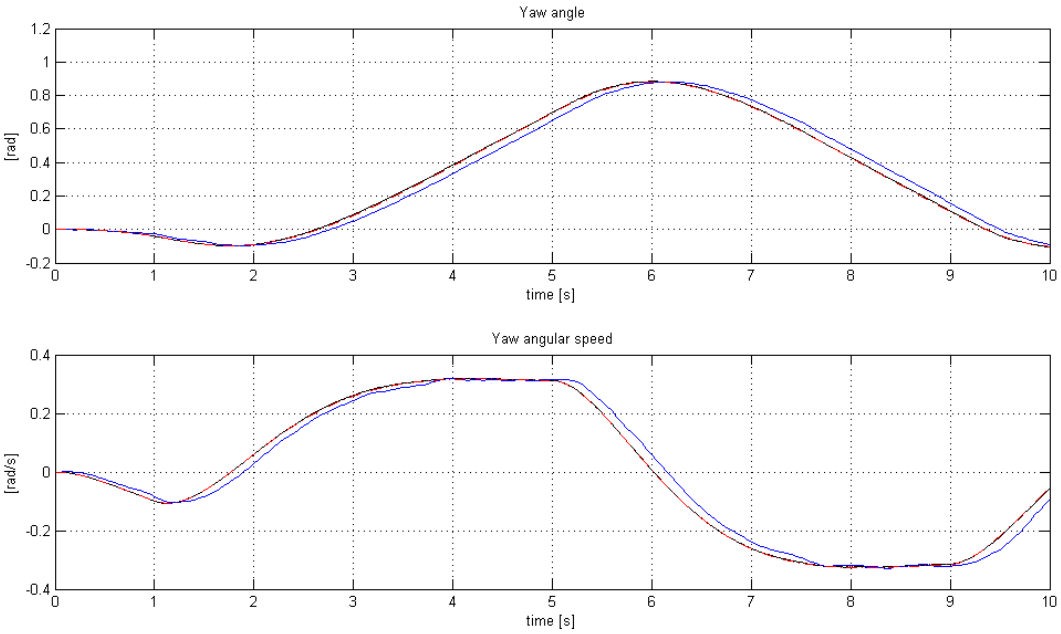


Figure 3.7: Yawing movement: the actual trajectory (red), the continuous time estimate (black) and the discrete time estimate (blue) of the relevant variables.

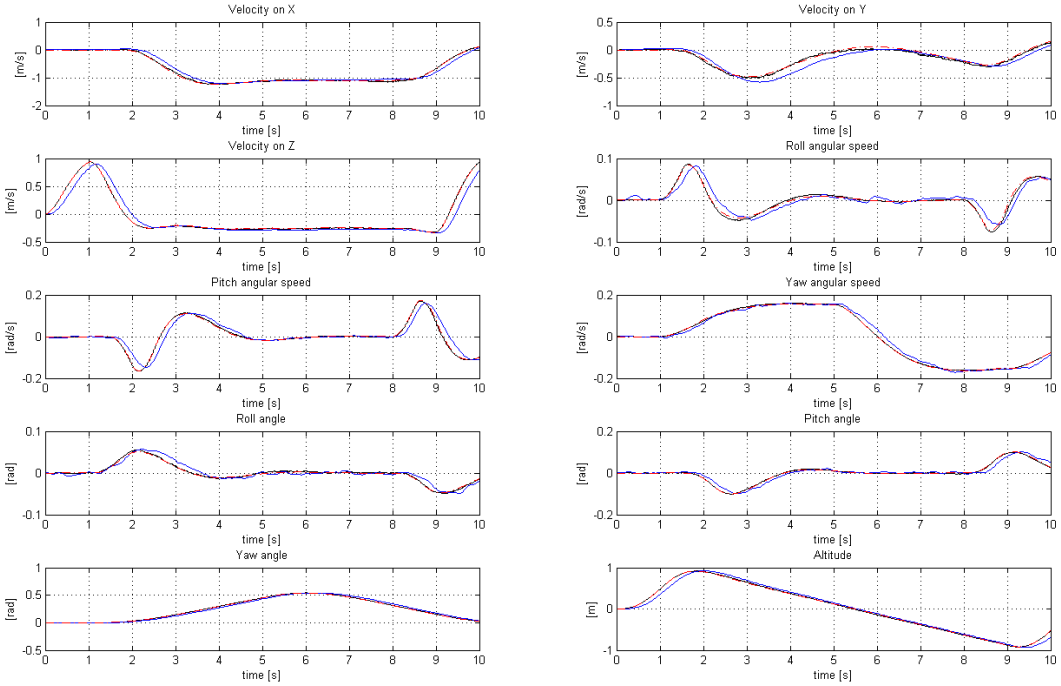


Figure 3.8: Complex movement: the actual trajectory (red), the continuous time estimate (black) and the discrete time estimate (blue) of the state variables.

3.1.5 Complex trajectory

This last simulation presents a complex trajectory, where the quadrotor moves on all the six degrees of freedom. In this case, all the variables are relevant and the results of the simulation are shown in Figure 3.8. As usual, the continuous time filter provides very accurate results and the discrete time one is also accurate, but its estimates are delayed. The only estimates that have a noticeable difference with respect to the target on the graphs in Figure 3.8 are the ones relative to the velocity on Y and the roll angular speed, but these differences are still very small.

3.2 Influence of the magnetic field direction on the estimation accuracy

The inclination of the magnetic field on the surface of the Earth varies from 0° at the magnetic equator to ±90° at the magnetic poles, it points downwards (negative sign) in the north emisphere and upwards (positive sign) in the south emisphere. In the previous simulations, the magnetic field inclination was supposed to be about −61°, a value that corresponds to the latitude of northern Italy. The experiments presented in this section show that the magnetic field inclination has a relevant influence on the accuracy of the attitude estimation with the Kalman filter.

Magnetic field inclination	Mean value of $\hat{\varphi}$
0°	-0.0101
-30°	-0.0100
-60°	-0.0058
-90°	-0.0016

Table 3.3: Mean value of the extimated roll angles with different magnetic inclinations (discrete time EKF).

Figures 3.9 and 3.10 show the estimates of the two versions of the Kalman filter with a hovering trajectory of 15 seconds and different values of the magnetic field inclination, from 0 to -90°. All the simulation were performed with the same random seed for sensor noise to obtain results that are directly comparable. The two figures, and in particular the one that regards the discrete time filter, which is less accurate, clearly show that the filter performance increases when the magnetic inclination is large and they are particularly poor when the inclination is zero. To explain this appearently strange behaviour, we must consider that the magnetometer, which is the primary sensor used to determine the Euler angles, cannot detect a rotation around the direction of the magnetic field vector. If the vector has the same direction as the X axis, the roll angle cannot be measured at all, so it can be inferred only by indirect measurements, like by integrating the data from the gyroscope, but the estimate is obviously less accurate and if the filter estimates a fake roll angle that cannot be corrected in the update step, according to the state transition model it will result in an acceleration on the Y axis that does not correspond to reality. The more the magnetic filed vector differs from the horizontal plane, the more the magnetometer is able to provide meaningful information about the attitude. The mean values of the estimated roll angle in Table 3.3 (referred to the discrete time filter, where the effect is more noticeable) confirm the explanation: the mean becomes closer to zero when the inclination increases.

The explanation given above can be proved by another experiment. If the magnetic field inclination is 0°, but the field is aligned with the Y axis instead of the X one[2], the angle that the filter cannot estimate well is now the pitch and this should produce a fake acceleration on the X axis. If the field is aligned to the bisector of the first and the third quadrants, the fake acceleration should be aligned with the bisector of the second and fourth quadrants (represented as an acceleration on bot the X and Y axis, with opposite signs). This third case is not so easy to understand: since the bisector is the fixed line, the movement that the magnetometer cannot sense is when a positive roll and a negative pitch occur togheter. In general, the fake acceleration is always more or less perpendicular to the direction of the magnetic field. These predictions are confirmed by the results in Figure 3.11.

[2]The orientation of the magnetic field vector (or its projection) on the XY plane is called *magnetic declination*.

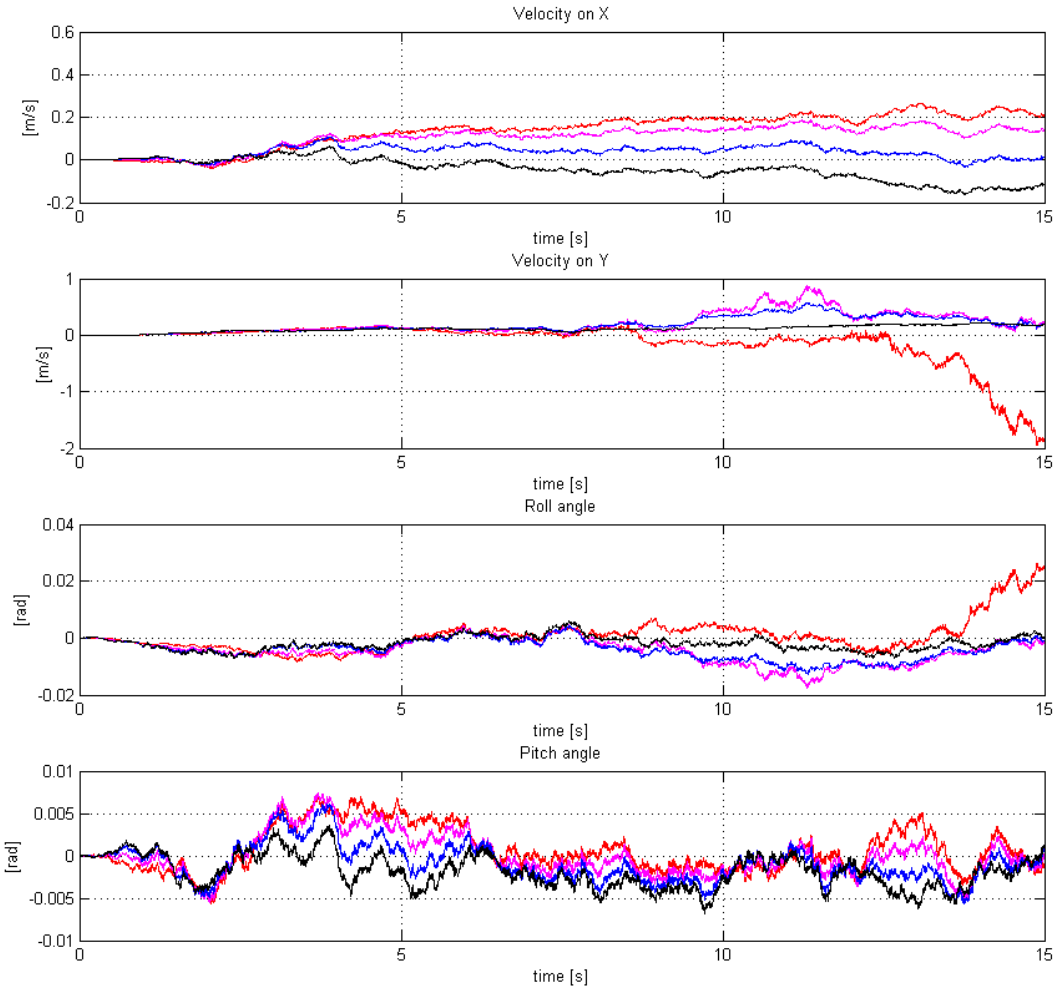


Figure 3.9: Continuous time filter performance with different values of the magnetic inclination: 0° (red), -30° (magenta), -60° (blue), -90° (black).

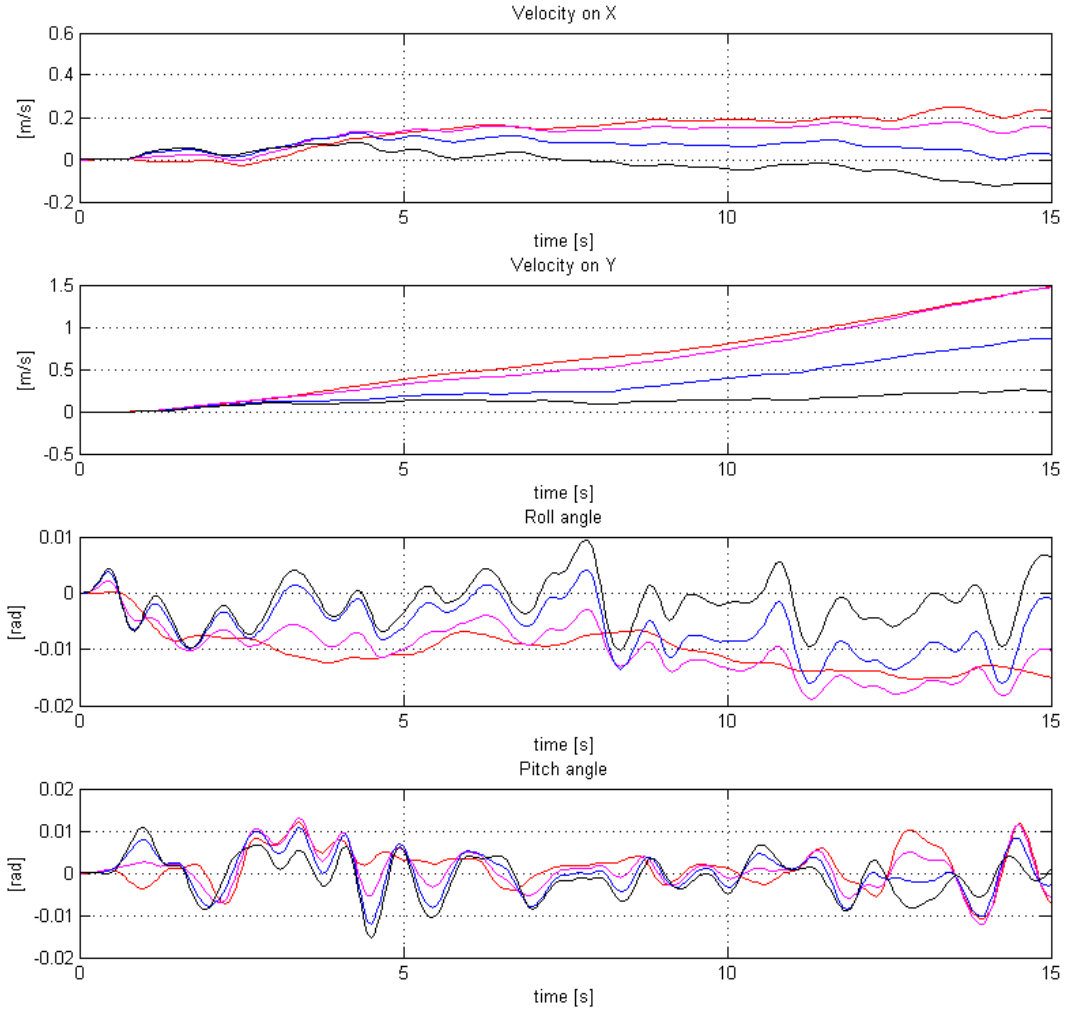


Figure 3.10: Discrete time filter performance with different values of the magnetic inclination: 0° (red), -30° (magenta), -60° (blue), -90° (black).

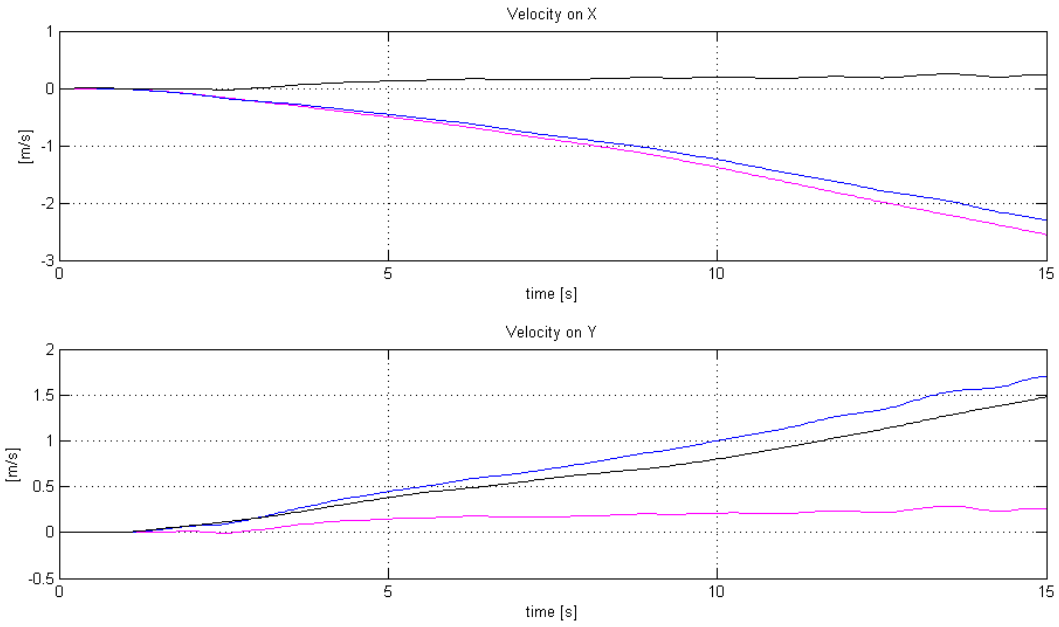


Figure 3.11: Estimated linear velocity components with different magnetic declinations: field vector parallel to the X axis (black), parallel to the Y axis (magenta) and parallel to the bisector of the first and the third quadrants (blue).

3.3 Closed loop simulations

The simulations analyzed in Section 3.1 show that the estimates of the extended Kalman filter are qualitatively good, but we want also to know how the controller performs when fed with these estimates instead of the actual state, which is obviously unknown in real applications. In this section, we present the results of some simulations where the control loop is closed around the EKF (both the continuous and dicrete time versions), i.e., its output is used as the input of the control system.

Augmented state The state variables defined in Subsection 1.6.1 are not enough for the controller: the derivative of the altitude is also needed. This derivative is computed from the value of the altitude itself using a standard Simulink block, as it is visible in the lower right angle of Figure 2.9 (the continuous time version of the derivative is computed in the same way, although that part of the model is not visible in the figures shown in Chapter 2).

3.3.1 Stability problems

Instability introduced by the anti-aliasing filter

It is known from control system theory that a time delay can lead to instability in a feedback system. In Subsection 3.1 we saw that the anti-aliasing filter introduces

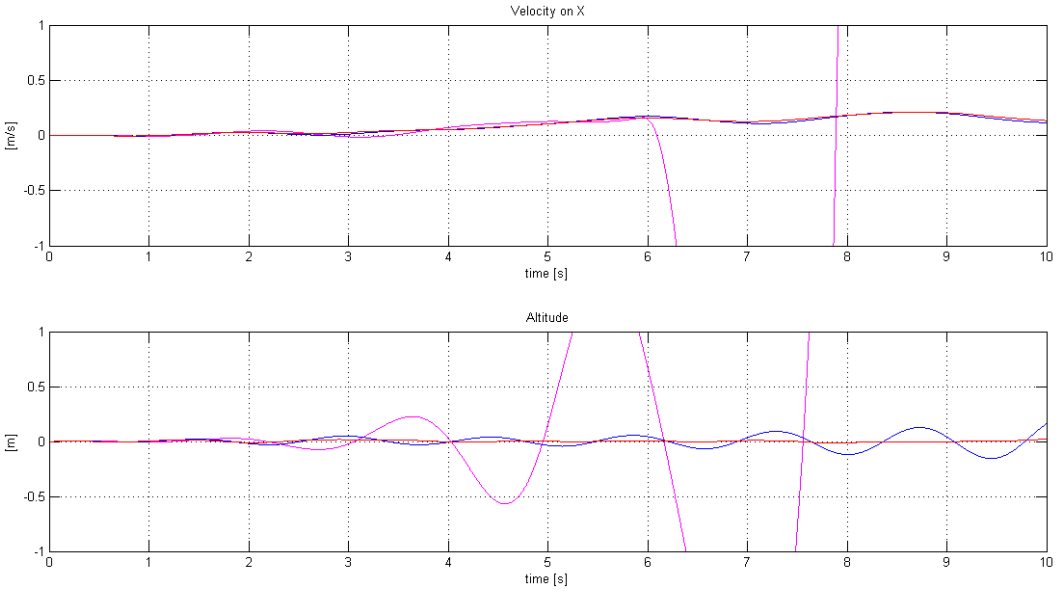


Figure 3.12: Stability of the closed loop simulation (hovering trajectory) using the discrete time EKF and anti-aliasing filters of order 4 (magenta), 2 (blue) and 1 (red).

a delay in the discrete time EKF response which is proportional to the order of the filter. Figure 3.12 shows the trajectories of the altitude and the velocity along the Y axis (the other state variables have similar behaviours) for hovering simulation in closed loop using the discrete time EKF with anti-aliasing filters of different order. The filter of order 4 which was used in the open loop simulations introduces a great instability, the filter of order 2 and 1 produce similar stable result for the Y axis velocity, but the altitude trajectory produced by the second order filter shows an oscillating behaviour that looks close to the limit of stability, this is probably a consequence of the fact that the computation of the derivative of z introduces an extra delay on this variable. Considering these results, the first order anti-aliasing filter alredy used for the open loop examples is the only one that is usable in closed loop simulation[3].

Instability in the continuous time filter

Another important source of instability regards the continuous time filter, which generates instability with almost all the trajectories (except hovering) and causes Simulink to terminate the simulation after few seconds with an error message. The problem is probably caused by the high frequency components in the response and

[3] A first order lowpass filter does not attenuate the high frequency as well as a fourth order one, however it is a minor problem with respect to the system instability. The delay can be reduced also by using a shorter discretization time step, but in a real world application the on board processor may not have enough computational power.

Quantity	Discrete time	Continuous time
Position on X	0.93 m	0.61 m
Position on Y	1.54 m	-0.15 m
Position on Z	0.02 m	0.01 m
Yaw angle	0.01 rad	-0.02 rad

Table 3.4: Drift values in hovering condition of the closed loop system after 10 seconds.

can be overcome by adding a lowpass filter to the output of the EKF block (visible in Figure 2.5 on the left). We choose a Butterworth filter of order 4 with a cutoff frequency of 50 Hz, which is used in all the following simulations except the hovering trajectory.

3.3.2 Hovering

The hovering trajectory is simple to analyze quantitatively, so it provides a lot of information about the overall performance of the system. The plots of the state variables in Figure 3.13 show that the discrete and continuous time versions of the filter produce similar results for some of them, like the velocity on X and the altitude, while the results are very different for some other ones, like yaw angle and the velocity on Y. Table 3.4 lists the final drift values from the fixed position after 10 seconds of simulation (these values are referred to the body frame and, excluding the yaw angular drift, they cannot be directly inferred from the state variables).

The data from Figure 3.13 and Table 3.4 confirm what we have alredy noted, that the linear velocity is more difficult to control than the other state variables, because it cannot be directly measured. The residual linear velocity is sufficient to cause a drift of almost two meters (considering both the X and Y axes) in ten seconds for the discrete time filter and about one meter for the continuous time version[4]. On the contrary, the altitude and the yaw angle are controlled with much higher precision.

3.3.3 Complex trajectories

In this subsection, we analyze the performance of the closed loop system with some trajectories composed by different moviments.

Forward moviment with altitude variation

The target trajectory is composed by three movements:

[4]The drift movements are random, so if the simulation is repeated with different random seeds for the sensor noises the result can be much diffrent. The values in the table are useful only to give a general idea of the control system precision.

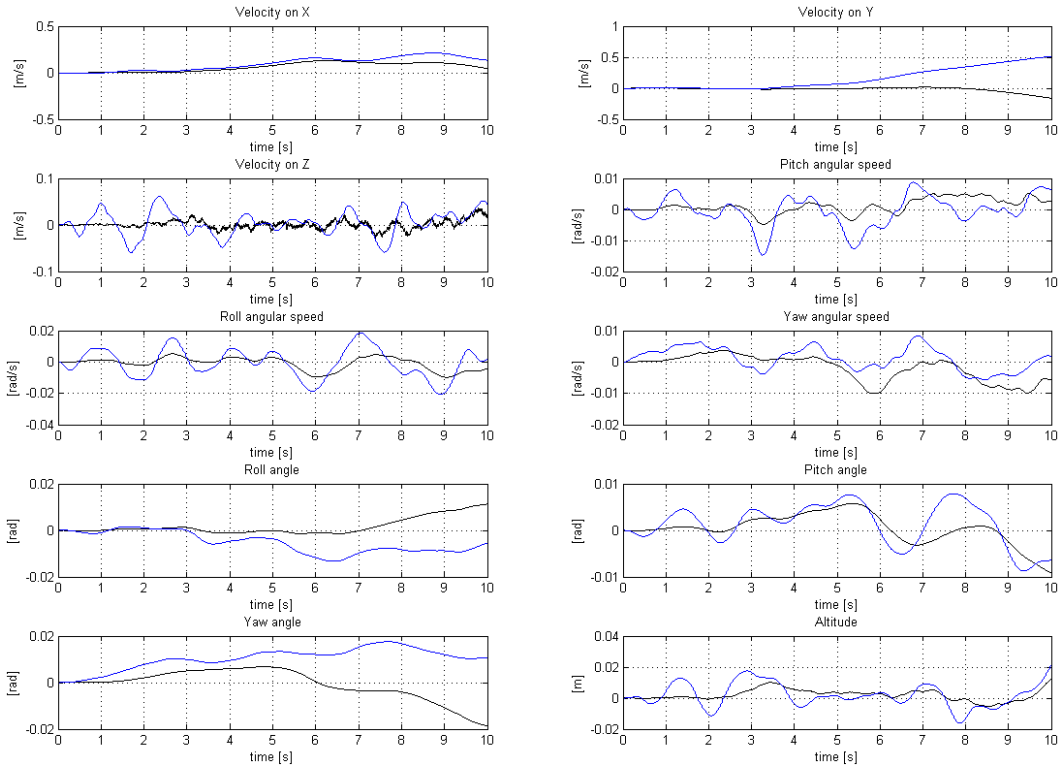


Figure 3.13: Trajectories of the state variables in a closed loop hovering simulation, continuous time filter (without the lowpass filter) in black and discrete time filter in blue.

1. a pitch angle tilt of 0.6 *rad* for one second, which produces an acceleration along the X axis;

2. an altitude reduction of 1 m, followed by a corresponding increase;

3. a negative pitch angle tilt of the same amount as the initial one, which should stop the linear movement.

As expected from the results in Figure 3.4, the actual pitch angles achieved by the controller will be about one half of the target one, in this case ± 0.3 *rad*.

Figure 3.14 shows the actual trajectories of the state variables in open loop (red dashdotted line), in closed loop with the continuous time version of the filter (black line) and in closed loop with the discrete time version of the filter (blue line). It can be noted that the relevant trajectories are followed quite well, in particular the value of z does not drop lower than the target, so the manouver can be performed safely even at low altitudes. As ususual, the velocity is hard to control, so there is a drift on the Y axis, however its modulus (1 m for the continuous time filter and 1.5 m for the discrete time one) is relatively small with respect to the main movement on the X axis (about 25 m).

Two perpendicular segments

This target trajectory is composed by the following movements:

1. a pitch angle tilt of 0.4 *rad* for one second, compensated by an opposite tilt one second later;

2. a yawing rotation of 90°;

3. another pitch angle tilt of 0.4 *rad* compensated by an opposite one, the subsequence is identical to the first one.

The resulting path should be similar to a couple of perpendicular segments of the same length. In pratice, the maximum pitch tilt is just 0.2 *rad* and the actual trajectory in the ideal case (the controller operates on the exact state variables) is the one showed in Figure 3.15. Since the quadrotor has a residual movement while it yaws, the second leg of the track is longer and there is a significant lateral drift that reduces the angle between the two segments.

As shown in Figure 3.16, the ideal trajectories and the ones obtained with the two versions of the filter are more different than the ones in the previous example. In particular, the residual velocity on Y, which generates the drift, is higher with the continuous time filter and the one obtained with the discrete time filter it is even lower that the one produced by the open loop control. This is the only case analyzed until now where the discrete time filter performs better than the continuous time one. On the contrary, the discrete time filter produces a velocity on X that is

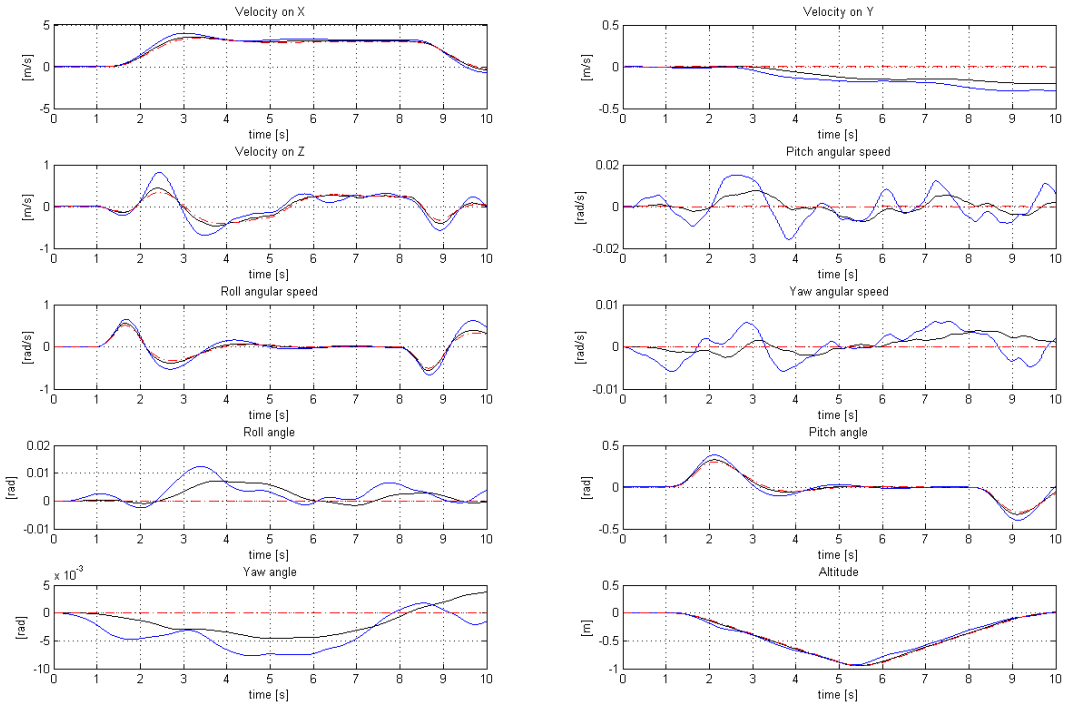


Figure 3.14: Trajectories of the state variables in closed loop: linear movement with altitude variation. The ideal trajectory is in red dashdotted, the one obtained with the discrete time filter is in blue and the one obtained with the continuous time filter is in black.

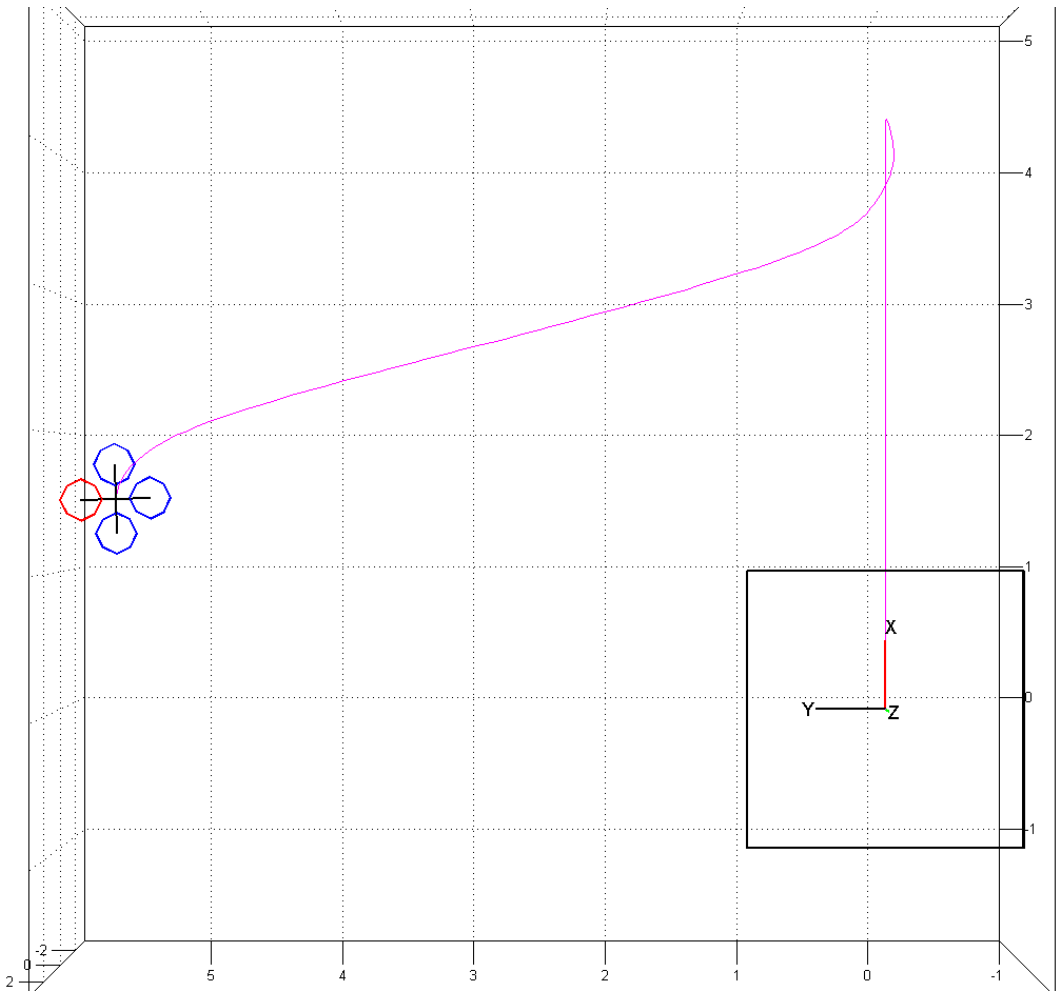


Figure 3.15: Actual trajectory of the quadrotor in the case of two perpendicualr segments. The trajectory is in magenta.

notably higher on the second leg of the path[5]. It can also be noted that the yaw is followed particularly well by both version of the filter.

Diagonal movement with altitude variations

This last target trajectory is composed by the following movements:

1. a pitch angle tilt of 0.5 *rad*, compensated by an opposite tilt six seconds later;

[5]Rembember that the state variables are expressed in body frame, so even if the two segments that compose the trajectory are perpendicular, since the quadrotor rotates on the Z axis between them, they are both seen as movements on the X axis.

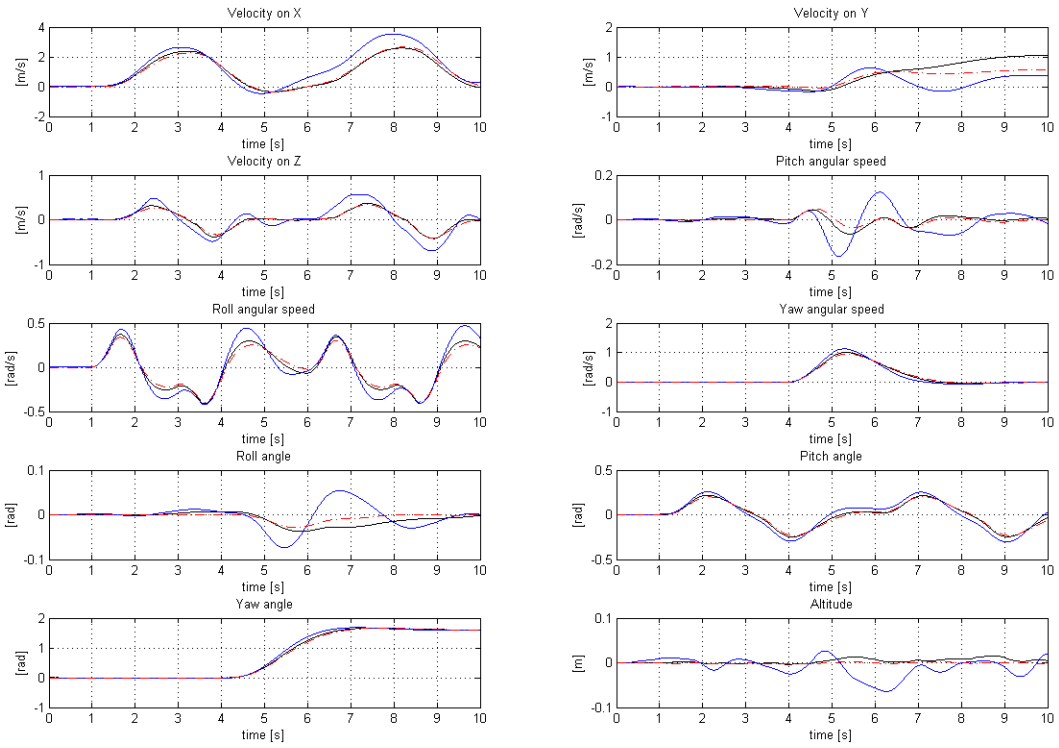


Figure 3.16: Trajectories of the state variables in closed loop: two perpendicular segments. The ideal trajectory is in red dashdotted, the one obtained with the discrete time filter is in blue and the one obtained with the continuous time filter is in black.

Control system	X	Y
Open loop	13.6 m	17.3 m
Closed loop discrete time	14.9 m	13.7 m
Cosed loop continuous time	12.8 m	8.9 m

Table 3.5: Coordinates of the end points of the diagonal trajectory.

2. a roll angle tilt of -0.5 rad, compensate by an opposite one six seconds later (both contemporary to the corresponding roll tilt);

3. an altititude reduction of 2 m, followed by an increase of 4 m and another two meters reduction (the quadrotor should return to the initial altitude at the end of the movement.

The effect of the two simultaneous tilts should be a diagonal movement along the bisector of the first quadrant, in pratice the velocity on Y axis produced by the open loop control is slightly greater than the one on the X axis, so the actual diagonal trajectory forms an obtuse angle with the X axis.

The trajectories of the state variables in Figure 3.17 show that the system with the discrete time filter in closed loop follows more precisely the velocity components on both the X and Y axes, but in particular the Y component is much lower than the target value with the continuous time filter in closed loop. Table 3.5 lists the coordinates of the end points of the trajectory with the open loop control and the two versions of the closed loop one. The more the values of the two coordinates are similar, the closer the achieved trajectory is to the ideal diagonal. Given the values in the table, the closed loop control with the discrete time filter seems to perform even better than the open loop one, however this must be just a casual compensation of the control and estimation errors, because in the open loop system the controller operates on the exact state (maximum information), so no real improvements are possibile without changing the control algorithm. From Figure 3.17 can also be noted that the closed loop control with both the filters produces a minimum altitude that is more than half a meter lower than the target one, this may be a problem if the manouver is performed near the ground.

Conclusion

The closed loop simulations presented in this subsection, although they provide quite a small dataset with respect to the huge number of possible trajectories, show that the quadrotor can be controlled using the discrete time version of the EKF to estimate the state and the control algorithm implemented by Bresciani in the pre-existing model [6] without instabilities (with a first order anti-aliasing filter), and the performance achieved, although far from being perfect, should be good enough to perform complex manouvers without flipping or crashing[6]. The closed loop control with the continuous time version of the filter has shown some problems, like the need

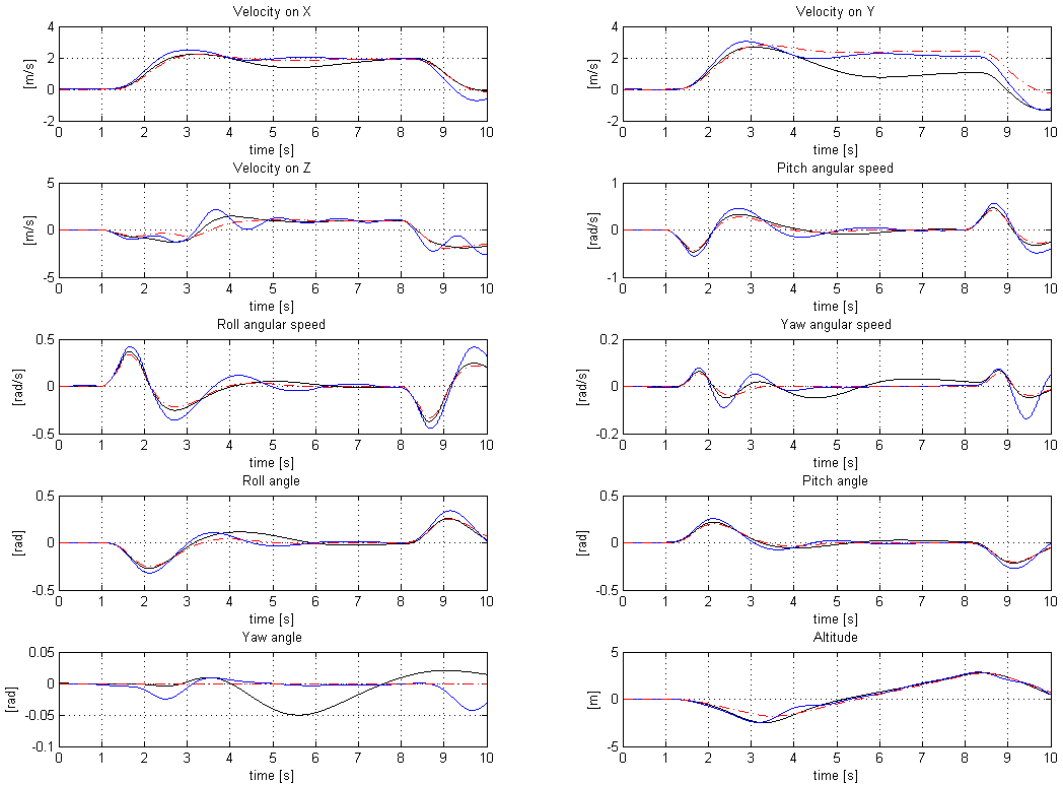


Figure 3.17: Trajectories of the state variables in closed loop: diagonal movement with altitude variations. The ideal trajectory is in red dashdotted, the one obtained with the discrete time filter is in blue and the one obtained with the continuous time filter is in black.

of a lowpass filter in series with it to produce a state estimate than can be used for a stable control. This lowpass filter is probably the cause of the bad performance of the system in some cases, which is worse than the ones obtained by the discrete time system, while in the open loop examples analyzed in Section 3.1 the continuous time EKF never performed worser than the discrete time one. Anyway, the discrete time system is more important in this case, because it is the one that can be implemented on a real quadrotor.

3.4 Simulations with errors in the estimates of the parameters

In the previous simulation, we have supposed that the parameters used in the extended Kalman filter are correctly estimated, so their values correspond to the real world quantities they refer to. However, in practice measurement and calibration errors may occur, so we are interested in analyzing the behaviour of the system when the filter parameters differ from the actual values.

3.4.1 Errors in physical parameters

Some of the parameters used by the Kalman filter, both in the state-transition and in the measurement models, represent the physical quantities that affect the system. These parameters might be set to incorrect values either because of erroneous measurements (high precision instruments are not always available) or small environmental changes during flight (expecially for groud pressure and temperature). The main physical parameters are:

- the mass of the quadrotor;
- the magnetic field versor;
- the athmosferic pressure at ground level;
- the gas constant of the air (which depends on the humidity);
- the air temperature;
- the diagonal values of the quadrotor inertia matrix.

The results of an hovering simulation where the estimated values of the parameters listed above are corrupted by a random generated error are shown in Figures 3.18 and 3.19 for the continuous and discrete time EKF respectively. The additive

[6]The main purpose of an attitude control system is to avoid instability and altitude loss during the manouvers, the control of the position and velocity is the task of the navigation system, which has to employ some sort of position measurements, but it is not discussed in this thesis. In this sense, the relatively small altitude loss obtained with the closed loop control in the third trajectory of this section is more relevant than the bigger horizontal drifts that occur in every simulation.

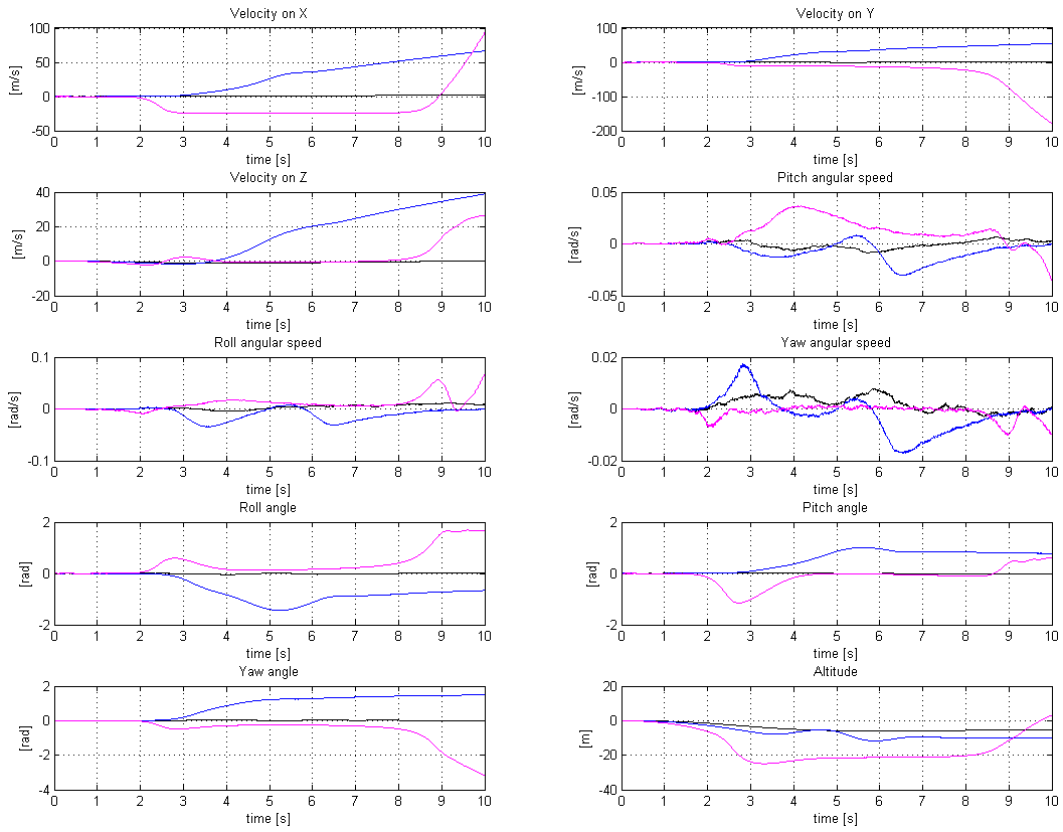


Figure 3.18: Continuous time filter estimates for an howering simulation with errors in the physical parameters of the Kalman filter. The errors are generated from a Gaussian distribution with a variance of 0.5% (black), 0.1% and 0.05% of the corresponding variable.

error is generated from a Gaussian distribution with different values of the variance. The continuous time filter shows a divergent behaviour, while the discrete time one appears to be much more error tolerant, however the altitude estimate is unacceptably wrong ($\sim 5\ m$) even with an error variance of just 0.05% in the parameters.

The high sensitivity to parameter estimation errors that emerges from these results is one of the main problems of this EKF implementation. The absolute value of the error added to the quadrotor mass parameter in the simulation, for example, was in the order of $10^{-4}\ kg$, which is about the weight of a feather. That means that even the smallest payload applied to the quadrotor would require a weighting on an high precision scale, not to mention what could happen if a leaf fell on it during the flight.

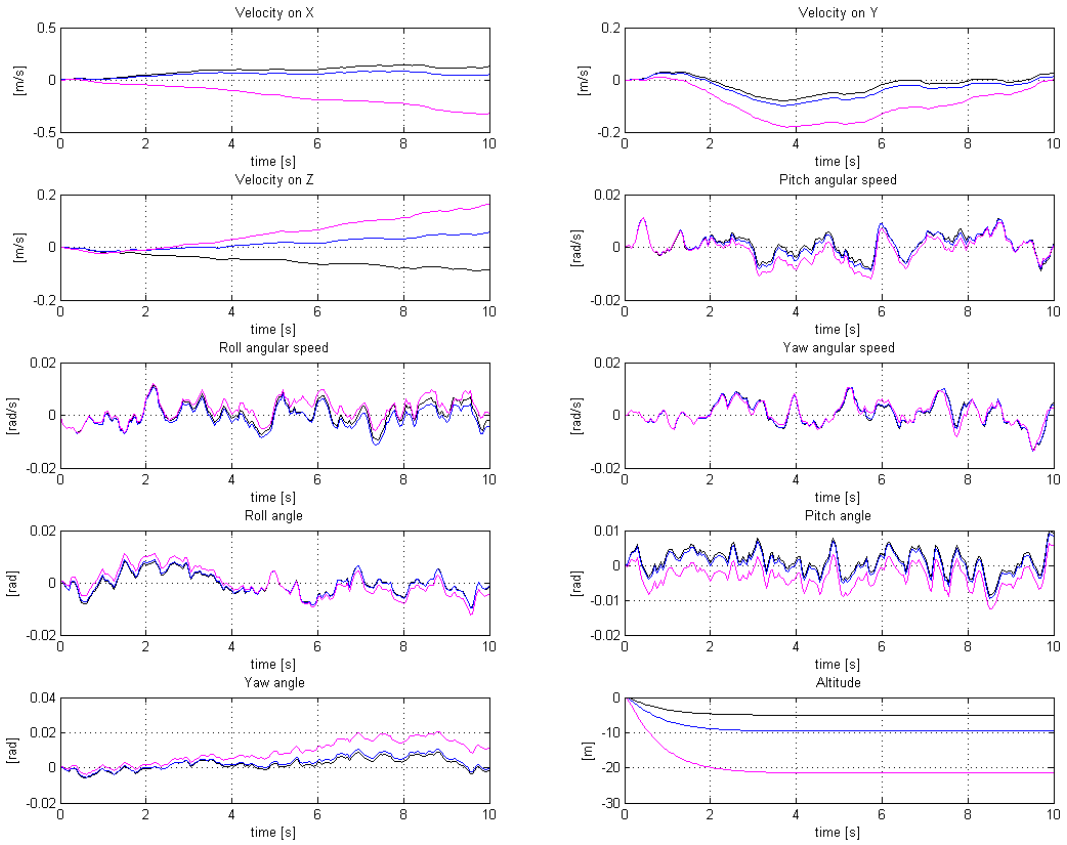


Figure 3.19: Discrete time filter estimates for an howering simulation with errors in the physical parameters of the Kalman filter. The errors are generated from a Gaussian distribution with a variance of 0.5% (black), 0.1% and 0.05% of the corresponding variable.

3.4.2 Errors in sensor calibration

In this subsection, we test the behaviour of the system when there are errors in the sensor calibration. According to Equation (1.45), the model of a three-axis MEMS sensor contains a misalignment matrix and a bias vector. In the first simulation, the values of the misalignment matrices were corrupted by a random error generated from a Gaussian distribution with diffrent variance values, the results are shown in Figures 3.20 and 3.21 (continuous and discrete time filter respectively). In the second simulation, the same type of error was applied to the bias vectors, the results are visible in Figures 3.22 and 3.23 (continuous and discrete time filter respectively).

The misalignment calibration errors produce quite a strange result with the continuous time filter: the blue lines in Figure 3.20, which corresponds to an error variance of 0.01, show a divergent behaviour for many variables, while the error variance of 0.05 (magenta lines) produces a quite large, but bounded estimation error. It is not easy to understand why the filter diverges with a sensor calibration error of 0.01 and not with a larger value such as 0.05, however if the simulation is repeated with different random error values generated with the same variance, the instability appears only in some cases, so maybe some particular sets of random values trigger a singulartity in the dynamic equations, while other do not. As for the errors in physical parameters, the discrete time Kalman filter is more error tolerant and predictable: the no instability is generated, and the larger the error variance value the lower the accuracy in the estimate (except for the roll angle and the related velocity on Y, but it is likely to be just a coincidence).

The bias calibration errors produce the same results with both the versions of the filter: while the variance of 0.05 causes a divergent behaviour, the two smaller values produce a relatively small estimation error. Since the discrete time filter appeared to be quite robust with respect to the sensor calibration error, a closed loop simulation using a complex trajectory was performed. The chosen trajectory combines a movement on Y (produced and stopped by roll tilts), an altitude variation and a yawing rotation, simulated with an error variance of 0.01 on both the misalignment matrices and the bias vectors. The results (limited to the most relevant variables) are shown in Figure 3.24, compared with the ones obtained by the control system in open loop (red dashdotted line). Although the closed loop trajectory cannot be defined satisfying, because of the large errors that are evident expecially in the altitude, the yaw angle and the velocity on Y, the system remains stable (at least for the time interval of the simulation) and does not produce dangerous movements, like flips or dives.

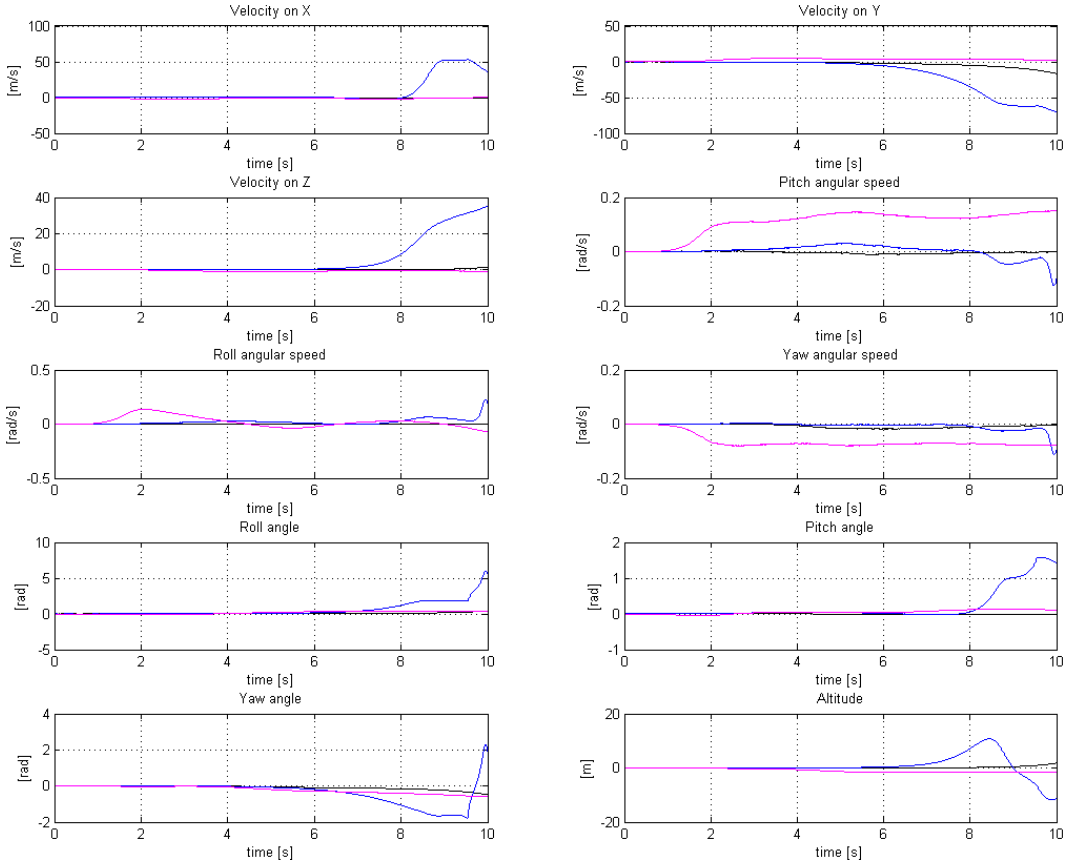


Figure 3.20: Continuous time filter estimates for an howering simulation with errors in the calibration of the sensor axis misalignment matrices. The errors are generated from a Gaussian distribution with variance of 0.005 (black), 0.01 (blue) and 0.05 (magenta).

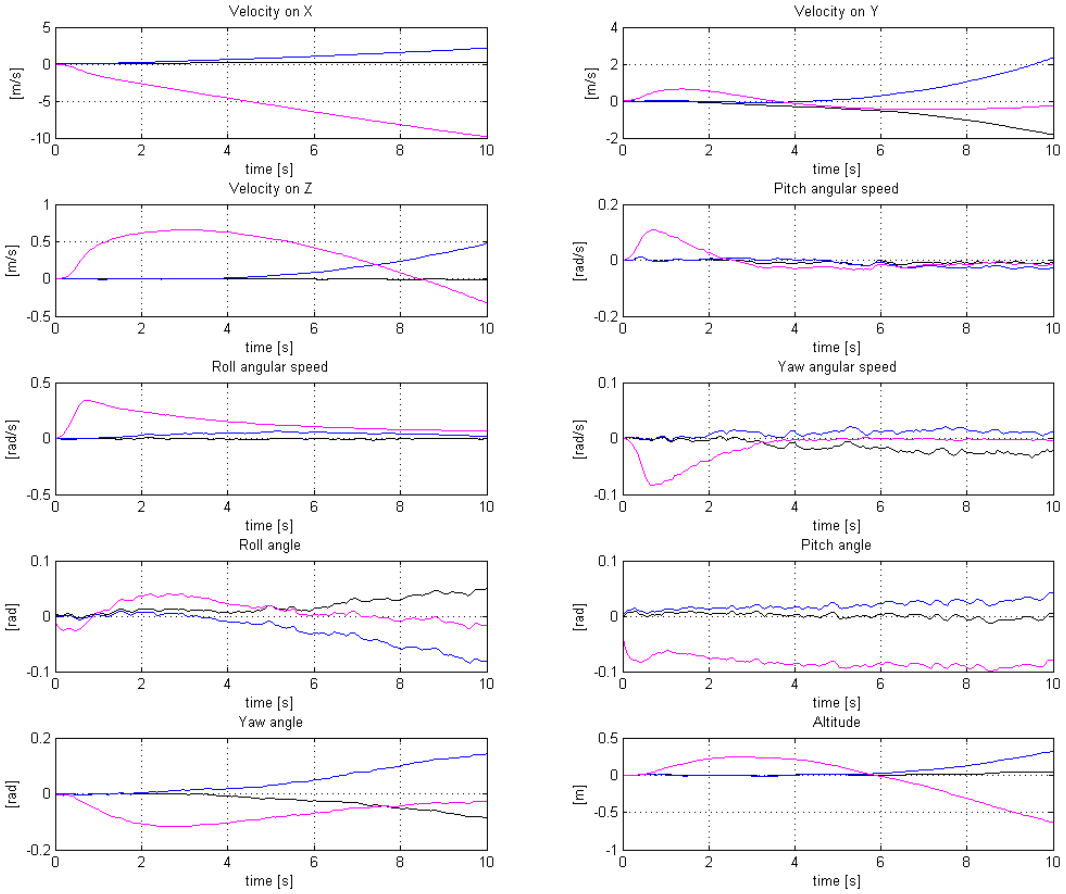


Figure 3.21: Discrete time filter estimates for an howering simulation with errors in the calibration of the sensor axis misalignment matrices. The errors are generated from a Gaussian distribution with variance of 0.005 (black), 0.01 (blue) and 0.05 (magenta).

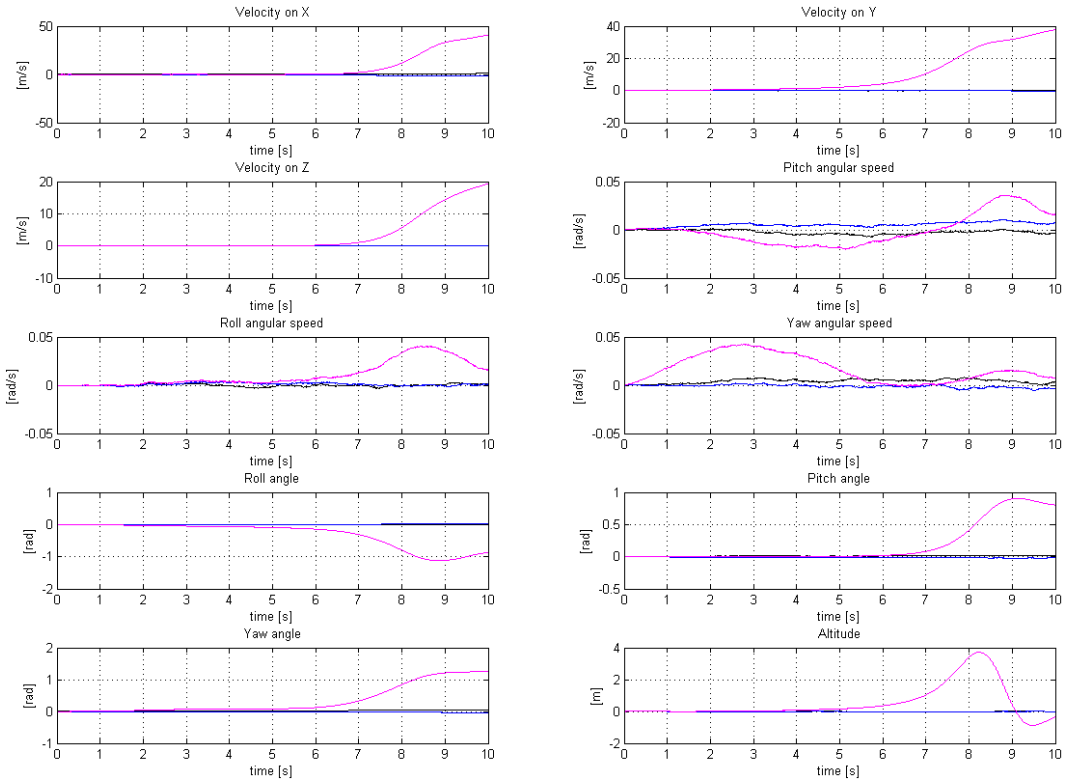


Figure 3.22: Continuous time filter estimates for an howering simulation with errors in the calibration of the sensor biases. The errors are generated from a Gaussian distribution with variance of 0.005 (black), 0.01 (blue) and 0.05 (magenta).

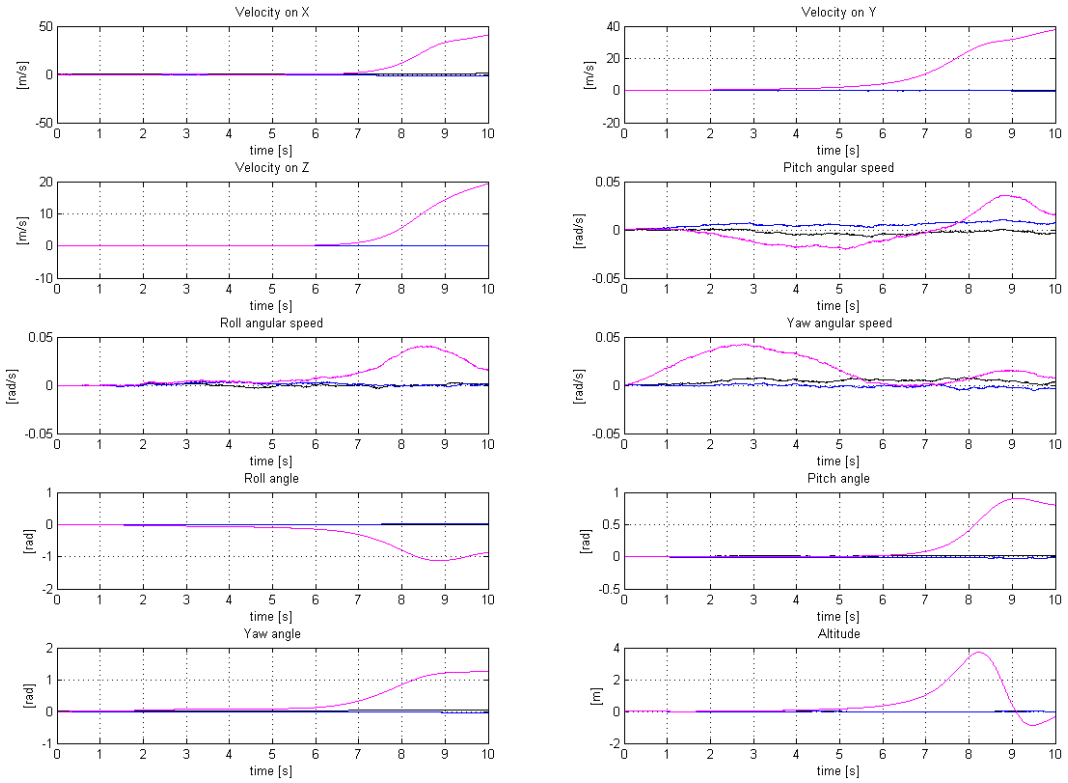


Figure 3.23: Discrete time filter estimates for an howering simulation with errors in the calibration of the sensor biases. The errors are generated from a Gaussian distribution with a variance of 0.005 (black), 0.01 (blue) and 0.05 (magenta).

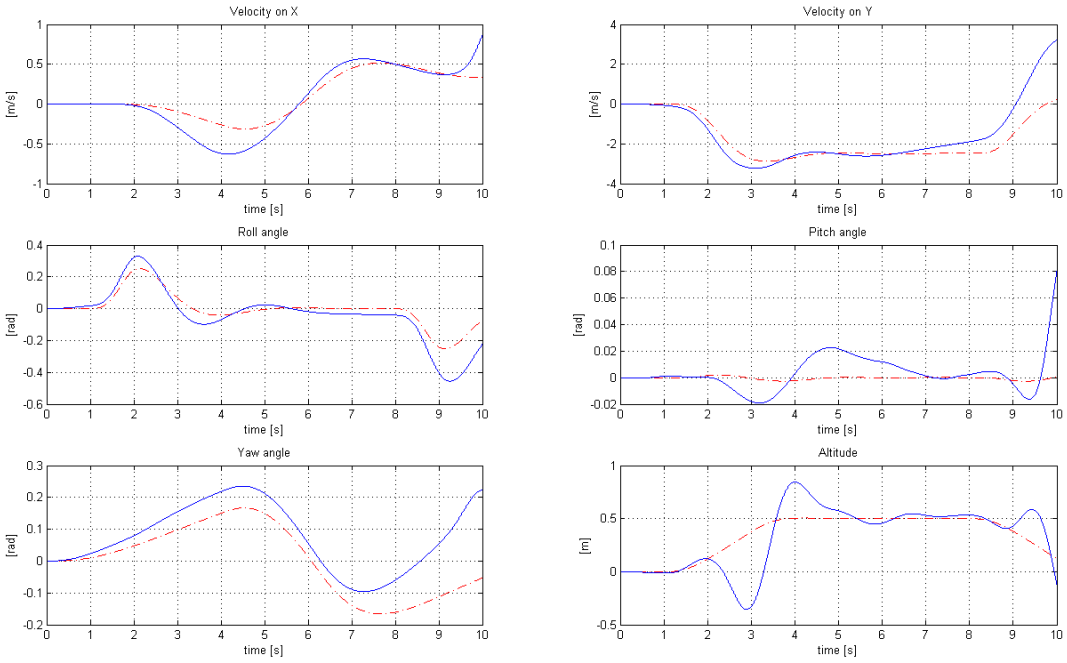


Figure 3.24: Closed loop simulation of a complex trajectory with calibration errors in both the sensor misalignment matrices and biases, the errors are generated from a Gaussian distribution with a variance of 0.01. Discrete time filter trajectories in blue and open loop system trajectorie in red dashdotted.

Chapter 4

Conclusion

The results presented in this thesis show that an implementation of the extended Kalman filter can provide an estimate of the attitude of a quadrotor that can be used as input by a PD control algorithm. If all the parameters are correctly set, the continuous time EKF has better accuracy, but since it is not implementable on an embedded system, it can be used only as a reference. The simulations show that a discrete time version of the filter working at a frequency of 20 Hz (suitable for an on-board implementation) can provide satisfying results in most situations, moreover it is much more robust to sensor calibration errors than the continuous time version.

The large number of parameters that must be set with high precision is still the main problem of this application of the extended Kalman filter. In particular, even small errors in the measurements of various physical quantities that influence both the sensor and the state-transition models may result in large estimation errors and even instability when the control loop is closed around the filter. The discrete time EKF seems more robust against sensor calibration errors, however they are less likely to occur in real world, since the calibration can be performed with high accuracy in a laboratory, while physical quantities like the atmosferic pressure, the temperature and even the mass of the aircraft if a payload is carried have to be measured at the moment for each flight.

Other limitations to the validity of the results presented in this thesis come from the assumptions used to simplify the simulation model, in particular:

- the dynamics are simulated by a system that exaclty matches the state-transition model of the Kalman filter, so a null Kalman gain would always lead to perfect estimates as long as the physical parameter are correctly set;

- the model does not take into account the response of the engines, which may be influent expecially in the case of fast manouvers, because they may not be enough reactive to change the angular rates of the propellers as required by the controller;

- the output of real-world MEMS sensors is not continuous in time, moreover they may operate at diffrerent frequencies and the samples may not be synchronized.

Because of the simplifications listed above, the thesis do not guarantee the factibility of a real world implementation of the described system, instead it provides only a theoretical analysis of its achievable performance and the related problems.

4.1 Further developments

The first improvement to the simulation model described in this thesis is the correction of the limitations listed in the previous paragrah. In particular, the performance of the filter when the state-transition model is also affected by random noise should be studied in depth, since the main purpose of the Kalman filter is the correction of both measurement and system errors.

Simulation with real data The Simulink model of the discrete time EKF can also be tested with real-world data. In this kind of experiment, the signals from the sensors and the controller are substituded with the ones registered by the IMU and the control unit of an actual quadrotor during a short flight. The estimates produced by the filter are then compared to the actual trajectory recorded by a sufficently accurate motion tracking system. The main difficulty of this experiment, apart from the alredy mentioned parameter calibration, is the synchronization of the data from the different sources, which is definitely non trivial, because different clocks, starting times and frequency are likely to be involved and interpolation may be necessary.

Embedded implementation of the system After the simulation stage, if the system is proven to be efficent and feasible, it can be implemented on board of a quadrotor. The problems that may arise at this stage are many. The real time measurement may come from the sensor at different time instant, so it may be better to design the update step of the filter in such a way that it can work for each sensor separately. The embedded hardware platform may also impose limitation to the maximum operating frequency of the filter (and the controller).

Navigation system As noted more than once in the thesis, this attitude control system cannot handle the position or the velocity of the quadrotor with accuracy (with the exception of the altitude), because the IMU sensors cannot maesure this quantities directly. The attitude control can be completed by a navigation system, which may be implemented with the Kalman filter too and may extend the same state-transition model. The navigation system must rely on some kind of position measurements, thay may come from a GPS for medium distance outdoor flights or may be vision based for indoor flight.

Bibliography

[1] DMW Abeywardena and SR Munasinghe. Performance analysis of a kalman filter based attitude estimator for a quad rotor uav. In *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*, pages 466–471. IEEE, 2010.

[2] Erdinc Altug, James P Ostrowski, and Camillo J 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. IEEE, 2003.

[3] Boris Andrievsky, Alexander Fradkov, and Dimitri Peaucelle. Adaptive control experiments for laas "helicopter" benchmark. In *2005 International Conference on Physics and Control, PhysCon 2005*, pages 760–765, 2005.

[4] Samir Bouabdallah, Andre Noth, and Roland Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2451–2456. IEEE, 2004.

[5] M Boutayeb, H Rafaralahy, and M Darouach. Convergence analysis of the extended kalman filter used as an observer for nonlinear deterministic discrete-time systems. *Automatic Control, IEEE Transactions on*, 42(4):581–586, 1997.

[6] Tommaso Bresciani. Modelling, identification and control of a quadrotor helicopter. Master's thesis, Lund university, October 2008.

[7] Richard Snowden Bucy and Peter D Joseph. *Filtering for stochastic processes with applications to guidance*, volume 326. American Mathematical Soc., 1987.

[8] Pedro Castillo, Alejandro Dzul, and Rogelio Lozano. Real-time stabilization and tracking of a four-rotor mini rotorcraft. *Control Systems Technology, IEEE Transactions on*, 12(4):510–516, 2004.

[9] Cosmin Coza and CJB Macnab. A new robust adaptive-fuzzy control method applied to quadrotor helicopter stabilization. In *Fuzzy Information Processing Society, 2006. NAFIPS 2006. Annual meeting of the North American*, pages 454–458. IEEE, 2006.

[10] Lefteris Doitsidis, Stephan Weiss, Alessandro Renzaglia, Markus W Achtelik, Elias Kosmatopoulos, Roland Siegwart, and Davide Scaramuzza. Optimal surveillance coverage for teams of micro aerial vehicles in gps-denied environments using onboard vision. *Autonomous Robots*, 33(1-2):173–188, 2012.

[11] J Dunfied, M Tarbouchi, and G Labonte. Neural network based control of a four rotor helicopter. In *Industrial Technology, 2004. IEEE ICIT'04. 2004 IEEE International Conference on*, volume 3, pages 1543–1548. IEEE, 2004.

[12] Davide Forti and Luca Lamparelli. Analisi e controllo della dinamica di un elicottero quadrotore. Master's thesis, Politecnico di Milano, 2009.

[13] M.S. Grewal and A.P. Andrews. *Kalman Filtering: Theory and Practice Using MATLAB*. Wiley, 2008.

[14] Nicolas Guenard, Tarek Hamel, and Laurent Eck. Control laws for the tele operation of an unmanned aerial vehicle known as an x4-flyer. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3249–3254. IEEE, 2006.

[15] Gabriel M Hoffmann, Haomiao Huang, Steven L Waslander, and Claire J Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proc. of the AIAA Guidance, Navigation, and Control Conference*, pages 1–20, 2007.

[16] Aldo Jaimes, Srinath Kota, and Jose Gomez. An approach to surveillance an area using swarm of fixed wing and quad-rotor unmanned aerial vehicles uav (s). In *System of Systems Engineering, 2008. SoSE'08. IEEE International Conference on*, pages 1–6. IEEE, 2008.

[17] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics, 1997.

[18] Rudolph E Kalman and Richard S Bucy. New results in linear filtering and prediction theory. *Journal of basic Engineering*, 83(3):95–108, 1961.

[19] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[20] Laszlo Kis, Gergely Regula, and Bela Lantos. Design and hardware-in-the-loop test of the embedded control system of an indoor quadrotor helicopter. In *Intelligent Solutions in Embedded Systems, 2008 International Workshop on*, pages 1–10. IEEE, 2008.

[21] Ilan Kroo and Peter Kunz. Development of the mesicopter: A miniature autonomous rotorcraft. In *American Helicopter Society (AHS) Vertical Lift Aircraft Design Conference, San Francisco, CA*, 2000.

[22] Barbara F La Scala, Robert R Bitmead, and Matthew R James. Conditions for stability of the extended kalman filter and their application to the frequency tracking problem. *Mathematics of Control, Signals and Systems*, 8(1):1–26, 1995.

[23] Quentin Lindsey, Daniel Mellinger, and Vijay Kumar. Construction of cubic structures with quadrotor teams. *Proc. Robotics: Science & Systems VII*, 2011.

[24] Cheng Liu, Zhaoying Zhou, and Xu Fu. Attitude determination for mavs using a kalman filter. *Tsinghua Science & Technology*, 13(5):593–597, 2008.

[25] Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. Pixhawk: A system for autonomous flight using onboard computer vision. In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pages 2992–2997. IEEE, 2011.

[26] Najib Metni, Tarek Hamel, and François Derkx. Visual tracking control of aerial robotic systems with adaptive depth estimation. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 6078–6084. IEEE, 2005.

[27] Yannick Morel and Alexander Leonessa. Direct adaptive tracking control of quadrotor aerial vehicles. In *Florida Conference on Recent Advances in Robotics*, pages 1–6, 2006.

[28] Talat Ozyagcilar and Freescale Semiconductor Inc. Calibrating an ecompass in the presence of hard and soft-iron interference. Application note, 01 2012. Document number AN4246.

[29] Konrad Reif, Stefan Gunther, Engin Yaz, and Rolf Unbehauen. Stochastic stability of the discrete-time extended kalman filter. *Automatic Control, IEEE Transactions on*, 44(4):714–728, 1999.

[30] Rush D Robinett, Gordon G Parker, Hanspeter Schaub, John L Junkins, JF BELLANTONI, and KW DODGE. A square root formulation of the kalman-schmidt filter. *AIAA journal*, 5(7):1309–1314, 1967.

[31] S Salazar-Cruz, A Palomino, and R Lozano. Trajectory tracking for a four rotor mini-aircraft. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 2505–2510. IEEE, 2005.

[32] Sebastian Scherer, Joern Rehder, Supreeth Achar, Hugh Cover, Andrew Chambers, Stephen Nuske, and Sanjiv Singh. River mapping from a flying robot: state estimation, river detection, and obstacle mapping. *Autonomous Robots*, 33(1-2):189–214, 2012.

[33] Alexandros Soumelidis, Péter Gáspár, Gergely Regula, and Béla Lantos. Control of an experimental mini quad-rotor uav. In *Control and Automation, 2008 16th Mediterranean Conference on*, pages 1252–1257. IEEE, 2008.

[34] Samanmalee Sugathadasa, Clyde Martin, and WP Dayawansa. Boundedness of covariance estimates of extended kalman filtering with directional measurements and linear state dynamics. In *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on*, volume 4, pages 3790–3795. IEEE, 2001.

[35] Metin Tarhan and Erdinç Altuğ. Ekf based attitude estimation and stabilization of a quadrotor uav using vanishing points in catadioptric images. *Journal of Intelligent & Robotic Systems*, 62(3-4):587–607, 2011.

[36] Abdelhamid Tayebi and Stephen McGilvray. Attitude stabilization of a vtol quadrotor aircraft. *Control Systems Technology, IEEE Transactions on*, 14(3):562–571, 2006.

[37] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. Intelligent robotics and autonomous agents series. Mit Press, 2005.

[38] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. IEEE, 2000.

Appendix A

Mathematical derivation of the Kalman filter

This appendix explains the mathematical fundations of the Kalman Filter. The first section provides some general notions about the probability theory applied to the robotics and proves the Bayes filter algorithm. Section A.2 contains the derivation of the Kalman filter in both the continuous and discrete time versions. It will be shown that the Kalman filter is just a particular case of the Bayes filter that can be applied when all the probability distributions are Gaussian.

A.1 Probability and Bayes filters

In robotics and autonomous systems theory, almost always the system decides its behaviour on the basis of the estimated value of some state variables. The state variables may represent many different things, like the position of the robot, some environmental conditions or the presence of other entities to interact with, but in any case, usually their value is not known a-priori and/or it cannot be directly measured. In order to make the right decisions, the system has to guess the expected value of the state variables, so the probability theory began to gain a major role in the field, leading to the so called *probabilistic robotics* [37].

A.1.1 Two fundamental theorems

Although a complete explanations of the basic concepts of the probabilty theory is beyond the scope of this appendix, there are two fundamental theorems that will be used in the mathematical derivations, so they are worth to be recalled.

Total probability theorem

The total probability theorem states that the probabilty distribution $p(x)$ can be computed from the *conditional probability* $p(x|y)$ (the probability of the event x

given the event y) and the total probability $p(y)$ using the formula:

$$p(x) = \int p(x|y)p(y) \tag{A.1}$$

Bayes rule

The Bayes rule relates the conditional probabilty of x given y to the conditial probability of y given x, according to the formula[1]:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\int p(y|x')p(x')dx'}. \tag{A.2}$$

The integrals in both (A.1) and (A.2) become summations if the domains of the random variables x and y are discrete.

A.1.2 The concept of belief

A key concept in probabilistic robotics is the *belief*. The belief represent the internal knowledge of the system about the probability distribution of the state variables. A robotic system can estimate the value of the state variables on the basis of both the measurements of the sensors and the past control actions. The system updates its previous estimate at every control action and every time it receives a new measurement, so the estimation is a recursive process. We can formally define the belief at time t as the probability distribution of the state variables conditioned by all the previous measurements and controls:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}). \tag{A.3}$$

Sometimes it is useful to compute the belief after the last control action u_t, but before the last measurement z_t. This probability distribution is usually called *prediction* and is denoted as $\overline{bel}(x_t)$:

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}). \tag{A.4}$$

The calculation of $bel(x_t)$ from $\overline{bel}(x_t)$ by incorporating the measurement z_t is called *measurement update*.

A.1.3 Bayes filter

The Bayes filter is a general recursive algorithm to calculate the believes. This algorithm is divided in two steps:

1. the prediction step calculates the prediction $\overline{bel}(x_t)$ from the previous belief $bel(x_{t-1})$ and the last control action u_t;

[1] This formulation assumes that $P(y) > 0$ as an existance condition of the fraction.

2. the measurement update step computes the final belief $bel(x_t)$ from the prediction $\overline{bel}(x_t)$ (obtained in the previous step) and the measurement z_t.

Like every recursive algorithm, the Bayes filter requires the initial condition at time t_0, in this case the probability distribution $bel(x_0)$. If the initial state of the system is known with certainty, the initial belief is a point mass distribution centered in x_0, if the state is totally unknown, $bel(x_0)$ may be a uniform distribution over the domain of x. If the initial state is only partially known, the initial belief can be any other probability distribution that is appropriate for the case, but the most frequent situations in pratice are the perfect knowledge or the complete ignorance.

Prediction step

In the prediction step, $\overline{bel}(x_t)$ is computed as the integral of the product of two probability distribution: the belief at time x_{t-1} $bel(x_{t-1})$ and the probability that the control u_t induces a transition from the state x_{t-1} to the state x_t. The equation of the prediction step is

$$\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx. \tag{A.5}$$

Measurement update step

In the measurement update step, the prediction $\overline{bel}(x_t)$ is multiplied by the probability that the measurement z_t may be observed in the state x_t. Since the product generally does not integrate to 1 (as required by the definition of probability distribution), the result is normalized my the normalization constant μ. The equation of this step is:

$$bel(x_t) = \nu p(z_t|x_t)\overline{bel}(x_t). \tag{A.6}$$

Any practical implementation of the Bayes filter requires the knowledge of three probability distributions: the initial belief $bel(x_t)$, the measurement probability $p(z_t|x_t)$ and the state-update probability $p(x_t|u_t, x_{t-1})$.

A.1.4 Mathematical derivation of the Bayes filter

The correctenss of the Bayes filter algorithm can be proved by induction. We have to prove that the filter can calculate the correct distribution $p(x_t|z_{1:t}, u_{1:t})$ given the distribution at the previous step $p(x_{t-1}|z_{1_t-1}, u_{1:t-1})$. Before proceding with the proof, we have to make an important assumption: we assume that the state x_t contains all the necessary information to make predictions about the future and in particular the knowledge of the past measurements and control would not help us predicting the measurement z_t[2]. Formally

$$p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t). \tag{A.7}$$

We start with the application of the Bayes rule in eq. (A.2) to the target belief:

$$p(x_t|z_{1:t}, u_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1_t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1_t})}, \tag{A.8}$$

if we consider the denominator as a scaling factor, the above equation becomes:

$$p(x_t|z_{1:t}, u_{1:t}) = \mu p(z_t|x_t, z_{1:t-1}, u_{1_t})p(x_t|z_{1:t-1}, u_{1:t}). \tag{A.9}$$

Considering the assumption in eq. (A.7) and recalling the definitions of $bel(x_t)$ and $\overline{bel}(x_t)$ the latter can be further simplified as:

$$bel(x_t) = \mu p(z_t|x_t)\overline{bel}(x_t), \tag{A.10}$$

which is the measurement update step in eq. (A.6). Now we can expand the term $\overline{bel}(x_t)$ according to the theorem of total probability in eq. (A.1) :

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) = \int p(x_t|x_{t-1}, z_{1:_t-1}, u_{1:t})p(x_{t-1}|z_{1:t-1}, u_{1:t})dx_{t-1}. \tag{A.11}$$

According to the assumption that the state is complete, if we know x_{t-1}, the past measurements and controls cannot provide us any information regarding x_t, so:

$$p(x_t|x_{t-1}, z_{1:_t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t). \tag{A.12}$$

We can also note that, if the controls are randomly chosen, the control u_t can be removed from the conditioning variables in $p(x_{t-1}|z_{1:t-1}, u_{1:t})$. Finally, eq. (A.11) becomes:

$$\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)p(x_{t-1}|z_{1:t-1}, u_{1:t-1})dx_{t-1}, \tag{A.13}$$

which is the prediction step equation (A.5).

A.2 Derivation of the Kalman Filter

The practical implementation of the Bayes filter can be very difficult or even impossible if the involved probability distributions have a complex shape. On the contrary, if all the distributions are Gaussian, the filter becomes simpler. It can be proved that an optimal estimator based on the Bayes filter can be developed if the following conditions are satisfied:

- the state variables have a continuous domain;
- the initial probability distribution of the state variables $bel(x_0)$ is Gaussian;

[2] If the state x satisfies this condition, it is called *complete*.

- the state-transition equation of the underlying dynamic model is linear;
- the measurement equation (the function from the state variables to the expected sensor output) is also linear;
- all the errors or noises in the system are additive and have a Gaussian distribution.

In this case, the optimal estimator is the Kalman filter, a particular case of the Bayes filter. The reason why the probability distributions are required to be Gaussian is the fact that a normal distribution is fully defined by just its mean and its variance. The linearity of the transformations and the noises ensures that the initial Gaussian distribution will remain Gaussian at any time in the future, because a linear transformation applied on a Gaussian function always changes it into another Gaussian.

A.2.1 Derivation of the discrete time Kalman filter

The derivation of the discrete Kalman filter algorithm that I propose in this appendix is based on the one presented in [13][3]. Before proceding with the derivation, there are two important principles that have to be exposed (the proof of these principles is too long to be included in this appendix):

Linear optimal estimator Given a random process $x(t)$, an estimator $\hat{x}(t)$ which is a function of the measurements $z(t)$ is optimal if it minimizes the expected value of a quadratic loss function of the error $x(t) - \hat{x(t)}$ with the probabilities conditioned by $z(t)$:

$$E\left\langle [x(t) - \hat{x(t)}]^T \boldsymbol{M} [x(t) - \hat{x(t)}] | z(t) \right\rangle, \tag{A.14}$$

where the matrix $\boldsymbol{M}$ is a symmetric, positive-definite *weighting matrix*. It is possible to proof that, if $x(t)$ and $z(t)$ are jointly Gaussian (i.e they are both normally distributed and the distribution of their joint probability is a multivariate Gaussian), the optimal estimator is a linear function of the measurements:

$$\hat{x}(t) = \sum_{i=1}^{k} \alpha_i z_i \tag{A.15}$$

Orthogonality principle The orthogonality principle states that, given a random process $x(t)$ and an estimator $\hat{x}(t)$ function of the measurements $z(t)$, the estimator achieves the minimum variance (this proposition is equivalent to say that it is

[3] The derivation proposed in [37], for example, is much more mathematically oriented, relying on the theory of probability.

optimal) if and only if it satisfies the two following equations:

$$E\left\langle x(t) - \hat{x}(t)\right\rangle = 0, \tag{A.16}$$

$$E\left\langle [x(t) - \hat{x}(t)]z(t)^T]\right\rangle = 0. \tag{A.17}$$

Equation (A.16) alone defines an *unbiased* estimator, while Equation (A.17) means that the estimation error of the optimal estimator is orthogonal with respect to the measurements. If the process $x(t)$ is zero-mean, Equation (A.16) is not necessary.

Now we can proceed with the derivation of the Kalman filter. Suppose that we want to update the current estimate of the state x_k of a dynamic system at time t_k with the information from the measurement z_k, which is related to the state by a linear equation $z_k = \boldsymbol{H}_k x_k + v_k$, where H_k is the measurement sensitivity matrix and v_k is the measurement noise. We also assume that the process x_k and the measurements z_k are jointly Gaussian, so the optimal estimator is a linear function of the *a priori* estimate $\hat{x_k}(-)$ and the measurement z_k:

$$\hat{x_k}(+) = \boldsymbol{K}_k^1 \hat{x_k}(-) + \overline{\boldsymbol{K}}_k z_k, \tag{A.18}$$

where $\hat{x_k}(+)$ is the updated estimate, $\hat{x_k}(-)$ is the a priori estimate and the coefficient matrices $\boldsymbol{K}_k^1$ and $\overline{\boldsymbol{K}}_k$ are the optimal gains to be determined. In particular, $\overline{\boldsymbol{K}}_k$ is the Kalman gain defined in Equation (1.7).

We have observed that the optimal estimator is the one that satisfies the orthogonality principle in Equation (A.17) (it is written for the continuous case, but it can be easly applied to the discrete one), so we exploit this theorem to find the unknown gains. If we substitute the state update formula (1.1) for $x(t)$ and Equation (A.18) for $\hat{x}(t)$, we obtain:

$$E\left\langle [\boldsymbol{\Phi}_{k-1}x_{k-1} + w_{k-1} - \boldsymbol{K}_k^1 \hat{x}_k(-) + \overline{\boldsymbol{K}}_k z_k]z_i^T\right\rangle = 0,\ i = 1, ..., k-1. \tag{A.19}$$

But considering the measurement model (1.2), Equation (A.19) can be further rewritten as

$$E\left\langle [\boldsymbol{\Phi}_{k-1}x_{k-1} - \boldsymbol{K}_k^1 \hat{x}_k(-) + \overline{\boldsymbol{K}}_k \boldsymbol{H}_k x_k - \overline{\boldsymbol{K}}_k v_k]z_i^T\right\rangle = 0,\ i = 1, ..., k-1. \tag{A.20}$$

The latter equation can be reduced through some passages:

$$\boldsymbol{\Phi}_k E\langle x_{k-1}z_k^T\rangle - \boldsymbol{K}_k^1 E\langle \hat{x}(-)_k z_i\rangle - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k \boldsymbol{\Phi}_k E\langle x_{k_1} z_i^T\rangle - \overline{\boldsymbol{K}}_k E\langle v_k z_i^T\rangle = 0, \tag{A.21}$$

$$\boldsymbol{\Phi}_k E\langle x_{k-1}z_k^T\rangle - \boldsymbol{K}_k^1 E\langle \hat{x}(-)_k z_i\rangle - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k \boldsymbol{\Phi}_k E\langle x_{k_1} z_i^T\rangle = 0, \tag{A.22}$$

$$E\left\langle [x_k - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k x_k - \boldsymbol{K}_k^1 x_k] - \boldsymbol{K}_k^1 (\hat{x}_k(-) - x_k)\right\rangle z_i^T = 0, \tag{A.23}$$

$$[1 - \boldsymbol{K}_k^1 - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k] E\langle x_k z_i^T\rangle. \tag{A.24}$$

In passage (A.22) we have canceled the term $E\langle v_k z_i^t\rangle$, because we assume that the error v_k is uncorrelated with the measurements z_i. Equation (A.24) is satisfied for any x_k if we choose

$$\boldsymbol{K}_k^1 = \boldsymbol{I} - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k. \tag{A.25}$$

Now we have to find the optimal value for $\overline{\boldsymbol{K}}_k$. We define the following errors:

$$\widetilde{x}_k(+) \equiv \hat{x}_k(+) - x_k, \tag{A.26}$$

$$\widetilde{x}_k(-) \equiv \hat{x}_k(-) - x_k, \tag{A.27}$$

$$\widetilde{z}_k \equiv z_k(-) - z_k = \boldsymbol{H}_k \hat{x}_k(-) - z_k. \tag{A.28}$$

Accroding to the orthogonality principle, we can write:

$$E\left\langle [x_k - \hat{x}_k]\widetilde{z}_k^T] \right\rangle = 0, \tag{A.29}$$

and subtracting Equation (A.17) from the discrete version of Equation (A.29):

$$E\left\langle [x_k - \hat{x}_k]\widetilde{z}_k^T] \right\rangle = 0. \tag{A.30}$$

Substituting the variables x_k, $\hat{x}_k(+)$ and $\widetilde{z}_k(-)$ respectively from Equations (1.1), (A.18) and (A.28):

$$E\left\langle [\boldsymbol{\Phi}_{k-1}x_{k-1} + w_{k-1} - \boldsymbol{K}_k^1 \hat{x}_k(-) - \overline{\boldsymbol{K}}_k z_k][\boldsymbol{H}_k \hat{x}_k(-) - z_k]^T \right\rangle = 0. \tag{A.31}$$

Considering that $E\langle w_k z_k^T \rangle = 0$, $E\langle w_k \hat{x}_k^T(-) \rangle = 0$ and $E\langle \widetilde{x}_k(-) v_k^T \rangle = 0$ and substituting for $\boldsymbol{K}_k^1$, z_k and $\widetilde{x}_k(-)$, Equation (A.31) can be rewritten as:

$$E\left\langle [-\hat{x}_k(-) + \overline{\boldsymbol{K}}_k \boldsymbol{H}_k \widetilde{x}_k(-) - \overline{\boldsymbol{K}}_k v_k][\boldsymbol{H}_k \widetilde{x}_k(-) - v_k]^T \right\rangle = 0 \tag{A.32}$$

The error covariance matrix before the update is defined as:

$$\boldsymbol{P}_k(-) = E\langle \widetilde{x}_k(-) \widetilde{x}_k^T(-) \rangle \tag{A.33}$$

and satisfies the equation:

$$[\boldsymbol{I} - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k]\boldsymbol{P}_k(-)\boldsymbol{H}_k^T - \overline{\boldsymbol{K}}_k \boldsymbol{R}_k = 0, \tag{A.34}$$

so the Kalman gain can be expressed as a function of the a priori covariance:

$$\overline{\boldsymbol{K}}_k = \boldsymbol{P}_k(-)\boldsymbol{H}_k^T [\boldsymbol{H}_k \boldsymbol{P}_k(-) \boldsymbol{H}_k^T + \boldsymbol{R}_k]^T \tag{A.35}$$

By substituting Equation (A.25) into Equation (A.18) we obtain:

$$\hat{x}_k(+) = \hat{x}_k(-) + \overline{\boldsymbol{K}}_k [z_k - \boldsymbol{H}_k \hat{x}_k(-)], \tag{A.36}$$

and after substituting z_k according to Equation (1.2), we can subtract x_k from both sides to obtain:

$$\widetilde{x}_k(+) = (\boldsymbol{I} - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k)\widetilde{x}_k(-) + \overline{\boldsymbol{K}}_k v_k. \tag{A.37}$$

By substituting the latter into the definition of the covariance matrix after the update $\boldsymbol{P}_k(+) = E\langle \widetilde{x}_k(+) \widetilde{(x)}_k^T(+) \rangle$ we obtain:

$$\boldsymbol{P}_k(+) = E\left\langle [\boldsymbol{I} - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k]\widetilde{x}_k(-)\widetilde{x}_k^T(-)[\boldsymbol{I} - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k]^T + \overline{\boldsymbol{K}}_k v_k v_k^T \overline{\boldsymbol{K}}_k^T \right\rangle \tag{A.38}$$

$$= (\boldsymbol{I} - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k)\boldsymbol{P}_k(-)(\boldsymbol{I} - \overline{\boldsymbol{K}}_k \boldsymbol{H}_k)^T + \overline{\boldsymbol{K}}_k \boldsymbol{R}_k \overline{\boldsymbol{K}}_k^T. \tag{A.39}$$

Equation (A.39) is called the Joseph form of the covariance update equation. It was derived by P. D. Joseph[7]. If we substitue $\overline{\boldsymbol{K}}_k$ from Equation (A.35), we can reduce it to the simpler form:

$$\boldsymbol{P}_k(+) = (\overline{\boldsymbol{K}}_k \boldsymbol{H}_k)\boldsymbol{P}_k(-), \tag{A.40}$$

which is commonly used for computations and corresponds to Equation (1.6).

The last equation of the Kalman filter that we have to derive is the error covariance extrapolation (1.4). We can subtract x_k from both sides of the state-update equation (1.3) and then substitue x_k from Equation (1.1) to obtain the propagation of the estimation error:

$$\begin{aligned}
\hat{x}_k(-) - x_k &= \boldsymbol{\Phi}_k \hat{x}_{k-1}(+) - x_k, \\
\widetilde{x}_k(-) &= \boldsymbol{\Phi}_k \hat{x}_{k-1}(+) - (\boldsymbol{\Phi}_k x_{k-1} + w_k) \\
\widetilde{x}_k(-) &= \boldsymbol{\Phi}_k [\hat{x}_{k-1}(+) - x_{k-1}] - w_{k-1} \\
\widetilde{x}_k(-) &= \boldsymbol{\Phi}_k \widetilde{x}_{k-1}(+) - w_{k-1}
\end{aligned}$$

We can use the latter in the definition of the error covariance:

$$\begin{aligned}
\boldsymbol{P}_k(-) &\equiv E\langle \widetilde{x}_k \widetilde{x}_k^T \rangle \\
&= \boldsymbol{\Phi}_{k-1} E\langle \widetilde{x}_{k-1}(+) \widetilde{x}_{k-1}^T(+) \rangle \boldsymbol{\Phi}_{k-1}^T + E\langle w_{k-1} w_{k-1}^T \rangle \\
&= \boldsymbol{\Phi}_{k-1} \boldsymbol{P}_{k-1}(+) \boldsymbol{\Phi}_{k-1} + \boldsymbol{Q}_{k-1},
\end{aligned}$$

which corresponds to Equation (1.4).

A.2.2 Derivation of the Kalman-Bucy filter

The derivation of the Kalman-Bucy filter (the continuous time version of the Kalman filter) is similar to the derivation of the discrete case. We want to finde an estimate $\hat{x}(t)$ of the random (vector) variable $x(t)$ which is a linear function of the measurements $z(t)$ and which minimizes the cost function in Equation (A.14). If we consider the time interval $\Delta t = [t_{k-1}, t_k]$, we can integrate the state-transition equation (1.8) as:

$$\boldsymbol{\Phi}(t_k, t_{k-1}) = \boldsymbol{\Phi}_k = \boldsymbol{I} + \boldsymbol{F}(t_{k-1})\Delta t + O(\Delta t^2), \tag{A.41}$$

where $O(\Delta t^2)$ represents the terms with powers of Δt greater than 1. The measurement noise becomes

$$\boldsymbol{R}_k = \frac{\boldsymbol{R}(t_k}{\Delta t} \tag{A.42}$$

and the process noise

$$\boldsymbol{Q}_k = \boldsymbol{G}(t_k)\boldsymbol{Q}(t_k)\boldsymbol{Q}(t_k)^T \Delta t. \tag{A.43}$$

If we combine the Equations (A.40) and (1.6) and substitute for the above relations, we have:

$$\boldsymbol{P}_k(-) = [\boldsymbol{I} + \boldsymbol{F}(t)\Delta t][I - \overline{\boldsymbol{K}}_{k-1}\boldsymbol{H}_{k-1}]\boldsymbol{P}_{k-1}(-)[\boldsymbol{I} + \boldsymbol{F}(t)\Delta t]^T + \boldsymbol{G}(t_k)\boldsymbol{Q}(t_k)\boldsymbol{Q}(t_k)^T \Delta t, \tag{A.44}$$

this last equation can be rewritten to explicit the variation of $P(-)$ over the time interval:

$$\begin{aligned}\frac{\boldsymbol{P}_k(-)-\boldsymbol{P}_{k-1}(-)}{\Delta t} &= \boldsymbol{F}(t)\boldsymbol{P}_{k-1}(-)+\boldsymbol{P}_{k-1}(-)\boldsymbol{F}(t)^T+\boldsymbol{G}(t_k)\boldsymbol{Q}(t_k)\boldsymbol{Q}(t_k)^T \\ &\quad -\frac{\overline{\boldsymbol{K}}_{k-1}\boldsymbol{H}_{k-1}\boldsymbol{P}_{k-1}(-)}{\Delta t}-\boldsymbol{F}(t)\overline{\boldsymbol{K}}_{k-1}\boldsymbol{H}_{k-1}\boldsymbol{P}_{k-1}(-)\boldsymbol{F}^T(t)\Delta t \\ &\quad +O(\Delta t^2).\end{aligned} \tag{A.45}$$

We can write the limit for $\Delta t \to 0$ in the Kalman gain expression in Equation (A.35):

$$\begin{aligned}\overline{\boldsymbol{K}}(t) &= \lim_{\Delta t\to 0}\left[\frac{\overline{\boldsymbol{K}}_k-1}{\Delta t}\right] \\ &= \lim_{\Delta t\to 0}\boldsymbol{P}_{k-1}(-)\boldsymbol{H}_{k-1}^T[\boldsymbol{H}_{k-1}\boldsymbol{P}_{k-1}(-)\boldsymbol{H}_{k-1}^T\Delta t+\boldsymbol{R}(t)]^{-1} \\ &= \boldsymbol{P}\boldsymbol{H}^T\boldsymbol{R}^{-1}.\end{aligned} \tag{A.46}$$

Substituting Equation (A.46) in Equation (A.45) and taking the limit, we obtain the Riccati equation[4]:

$$\dot{\boldsymbol{P}}(t)=\boldsymbol{F}(t)\boldsymbol{P}(t)+\boldsymbol{P}(t)\boldsymbol{P}(t)^T+\boldsymbol{G}(t)\boldsymbol{Q}(t)\boldsymbol{G}(t)^T-\overline{\boldsymbol{K}}(t)\boldsymbol{R}(t)\overline{\boldsymbol{K}}^T(t), \tag{A.47}$$

which corresponds to Equation (1.11).

[4] Note that the identity

$$\overline{\boldsymbol{K}}(t)\boldsymbol{R}(t)\overline{\boldsymbol{K}}^T(t)=\boldsymbol{P}(t)\boldsymbol{H}^T(t)\boldsymbol{R}^{-1}(t)\boldsymbol{R}(t)\boldsymbol{R}^{-1}(t)\boldsymbol{H}(t)\boldsymbol{P}(t)$$

has been used to simplify the equation.

Appendix B

Matlab code

In this appendix the complete Matlab code used by the Simulink model described in Chapter 2.

Initialization script

The script `init.m` initializes all the variables, the constants and the parameters used in the model. The values of some parameters have been changed from the configuration listed here during testing.

Listing B.1: Initialization script (`init.m`.

```
1  %-------------------------------------------------------------
2  % PARAMETERS FOR SYSTEM SIMULATION
3  %-------------------------------------------------------------
4  % This script initializes all the parameters required
5  % by the simulink model. The script must be called
6  % before executing the simulation.
7  %-------------------------------------------------------------
8
9  % Simulation time (seconds)
10 T_sim=10;
11
12 %-------------------------------------------------------------
13 % Parameters used by the motor block
14 %-------------------------------------------------------------
15 Vdd=11.1;
16 Km=38.46e-3;  % Km=4.3e-3;
17 Ke=Km;
18 R=0.67;
19 L=500e-6;
20 J=3.4e-5;
21 tau_elet=L/R;
22 tau_mech=R*J/(Km*Ke);
23
24 %-------------------------------------------------------------
```

```
25 % Environmental parameters
26 %--------------------------------------------------------------------
27 global g; % Gravitational acceleration
28 global mag_field; % Magnetic field vector
29 global P0; % Pressure at ground level [Pa].
30 global Ra; % Gas constant of the air [J/(Kg*K)].
31 global T0; % Temperature at ground level [K].
32
33 mag = [cos(1); -0.1; -sin(1)];
34 mag_field = mag / norm(mag);
35 g = 9.81;
36 P0 = 101325; % Mean value at sea level.
37 Ra = 286.9;
38 T0 = 293.15;
39
40 %--------------------------------------------------------------------
41 % Initial conditions
42 %--------------------------------------------------------------------
43 global x0;
44 global y0;
45 global z0;
46 global phi0;
47 global theta0;
48 global psi0;
49 global dx0;
50 global dy0;
51 global dz0;
52 global dphi0;
53 global dtheta0;
54 global dpsi0;
55
56 x0=0;
57 y0=0;
58 z0=0;
59 phi0=0;
60 theta0=0;
61 psi0=0;
62 dx0=0;
63 dy0=0;
64 dz0=0;
65 dphi0=0;
66 dtheta0=0;
67 dpsi0=0;
68
69 %--------------------------------------------------------------------
70 % Mechanical parameters
71 %--------------------------------------------------------------------
72 global m; % Mass of the quadrotor
73 global b; % Thrust factor
74 global d; % Drag factor
75
76 % Diagonal elements of the inertia matrix
77 global Ix;
```

```
78 global Iy;
79 global Iz;
80
81 global Jr; % Rotor inertia
82 global l;  % Center of mass-motor distance
83
84 m=0.5;
85 b=7.2e-5;   % 2.9e-5; calculated as b=(m*g/4) / (omega hoovering)^2
86 d=1.1e-5;
87 Ix=5e-3;
88 Iy=5e-3;
89 Iz=9e-3;
90 Jr=3.4e-5;
91 l=0.25;
92
93 % Graphics inizialization for draw_mod function
94 global index_view;
95 global old_position;
96
97 index_view = 0;
98 old_position = [0 0 0];
99
100
101 % ---------------------------------------------------------
102 % KALMAN FILTER PARAMETERS
103 % The following parameters are used by the blocks
104 % related to the Kalman filter. Many of them reproduce
105 % the parameters used in the system simulation, but
106 % are distinguished by the suffix _e.
107 % ---------------------------------------------------------
108 % Physical and environmental parameters
109 g_e = g;
110 m_e = m;
111 mf_e = mag_field;
112 P0_e = P0;
113 T0_e = T0;
114 Ra_e = Ra;
115 Ix_e = Ix;
116 Iy_e = Iy;
117 Iz_e = Iz;
118
119 % Sensor parameters
120 Ma_e = eye(3);
121 ba_e = zeros(3,1);
122 Mg_e = eye(3);
123 bg_e = zeros(3,1);
124 G_e = eye(3);
125 bm_e = zeros(3,1);
126 kb_e = 1;
127 bb_e = 0;
128
129 % Plant and measurement noise covariances
130 R_e = 0.01*eye(10); % Measurement noise
```

```
131 R_e(10,10) = 1;
132 Q_e = 0.005*eye(10); % Process noise
133
134 % Initial expected value and variance of the state vector.
135 Ex = zeros(10,1);
136 Varx = 0.05*eye(10);
137
138 % Lowpass filter parameters
139 [b_filt, a_filt] = butter(4, 50, 's');
140
141 % ------------------------------------------------------------
142 % Parameters for the discrete time EKF
143 % ------------------------------------------------------------
144 % Samplig time.
145 Ts = 0.05;
146
147 % Sensor anti-aliasing filter parameters
148 [zeros_aa, poles_aa, gain_aa] = butter(1, 0.8/Ts,'s');
149
150
151 disp('initialization done')
```

Control subsystem

The code of the control algorithm has not been changed from the model by Tommaso Bresciani [6]:

Listing B.2: Control algorithm.

```
1  function out=control(x, task, m, g, Ix, Iy, Iz)
2  % ------------------------------------------------------------
3  % This script applies the control laws to the input
4  % data inferred from the sensors to obtain the values
5  % of the four control variables Ui.
6  % ------------------------------------------------------------
7  %#codegen
8
9  % State inferred from the sensors
10 phi = x(1);   % angular positions
11 theta = x(2);
12 psi = x(3);
13 dphi = x(4); % angular velocities
14 dtheta = x(5);
15 dpsi = x(6);
16 z = x(7);
17 dz = x(8);
18
19 % Task data
20 heightd = task(1);
21 rolld = task(2);
22 pitchd = task(3);
23 yawd = task(4);
```

```
24
25 % Control parameters
26 k1 = 4;
27 k2 = 2;
28 k3 = 4;
29 k4 = 2;
30 k5 = 4;
31 k6 = 2;
32 k7 = 2;
33 k8 = 2;
34
35 % Control laws
36 control_1 = -k1*(z - heightd) - k2*dz;
37 U1 = (m*g+control_1)/(cos(phi)*cos(theta));
38 U2 = Ix*(-k3*(phi - rolld) - k4*dphi);
39 U3 = Iy*(-k5*(theta - pitchd) - k6*dtheta);
40 U4 = Iz*(-k7*(psi - yawd) - k8*dpsi);
41
42 out = zeros(4,1);
43 out(1) = U1;
44 out(2) = U2;
45 out(3) = U3;
46 out(4) = U4;
```

The control subsystem contains also the block `force2Vcontrol`:

Listing B.3: `forces2Vcontrol` block.

```
1  function [out, Omega] = force2Vcontrol(in, b, d, l)
2  %-------------------------------------------------------
3  % This script converts the values of the control
4  % variables Ui into the corresponding voltage values
5  % to power the motor block. Note that some values
6  % of the control value generate a complex output,
7  % in this case the control action is not feasible in
8  % real world.
9  %-------------------------------------------------------
10 %#codegen
11
12 % Control variables
13 U1=in(1);
14 U2=in(2);
15 U3=in(3);
16 U4=in(4);
17
18 % The control variables are converted into the corresponding
19 % angular rates
20 U=[b b b b;0 -l*b 0 l*b;-l*b 0 l*b 0;-d d -d d];
21 Omega=sqrt(complex(U\[U1;U2;U3;U4]));  %[rad/s]
22
23 % The angular rates are converted into the voltage values
24 Vcontrol=(Omega-1.4)/26;   %[V]
25
```

```
26 out = complex(zeros(4,1));
27 out(1)=Vcontrol(1);
28 out(2)=Vcontrol(2);
29 out(3)=Vcontrol(3);
30 out(4)=Vcontrol(4);
```

Dynamics subsystem

Here is the code of the block that implments the dynamic equations:

Listing B.4: Dynamic system equations.

```
1  function out=dynamic_system(in, m, b, d, g, Ix, Iy, Iz, Jr, l)
2  %-------------------------------------------------------------
3  % This script models the movements of the quadrotor
4  % as a dynamic system.
5  %-------------------------------------------------------------
6  %#codegen
7
8  % Angular position (Euler angles)
9  phi=in(4);    % Roll
10 theta=in(5); % Pitch
11 psi=in(6);    % Yaw
12
13 % Linear velocity components
14 dx=in(7);
15 dy=in(8);
16 dz=in(9);
17
18 % Angular velocity components
19 dphi=in(10);
20 dtheta=in(11);
21 dpsi=in(12);
22
23 % Angular rates of the rotors
24 omega=in(13:16);
25
26 % Driving forces computed from the rotor angular rates
27 U1=b*(omega(1)^2+omega(2)^2+omega(3)^2+omega(4)^2);
28 U2=l*b*(omega(4)^2-omega(2)^2);
29 U3=l*b*(omega(3)^2-omega(1)^2);
30 U4=d*(-omega(1)^2+omega(2)^2-omega(3)^2+omega(4)^2);
31 OMEGA=-omega(1)+omega(2)-omega(3)+omega(4);
32
33 %-------------------------------------------------------------
34 % Dynamic system equations
35 %-------------------------------------------------------------
36 out = zeros(12,1);
37 out(1)=dx;
38 out(2)=dy;
39 out(3)=dz;
40 out(4)=dphi;
```

```
41 out(5)=dtheta;
42 out(6)=dpsi;
43 out(7)=(cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi))*U1/m;
44 out(8)=(cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi))*U1/m;
45 out(9)=-g+(cos(phi)*cos(theta))*U1/m;
46 out(10)=dtheta*dpsi*(Iy-Iz)/Ix-Jr*dtheta*OMEGA/Ix+U2/Ix;
47 out(11)=dphi*dpsi*(Iz-Ix)/Iy+Jr*dphi*OMEGA/Iy+U3/Iy;
48 out(12)=dpsi*dtheta*(Ix-Iy)/Iz+U4/Iz;
```

IMU subsystem and library

The library created to make the useful blocks in the IMU model reusable contains two blocks implemented with Matlab code: `Earth to body frame` and `Euler to pqr conversion`.

Listing B.5: Earth to body frame conversion.

```
1  function y = fix2body(u, ang)
2  %#codegen
3  %-------------------------------------------------------
4  % This function converts the vector u from the earth
5  % frame to the body frame, given the orentation of the
6  % body in Euler angles.
7  %-------------------------------------------------------
8
9  % Euler angles
10 phi = ang(1);
11 theta = ang(2);
12 psi = ang(3);
13
14 % Rotation matrix
15 R = [cos(psi)*cos(theta), ...
       cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi),...
16      cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
17        sin(psi)*cos(theta), ...
              sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi),...
18        sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
19        -sin(theta) cos(theta)*sin(phi) cos(theta)*cos(phi)];
20
21 y = R'*u;
```

Listing B.6: Euler angles to pqr conversion.

```
1  function pqr = euler2pqr(ang, dang)
2  %#codegen
3  %-------------------------------------------------------
4  % This function converts the angular rates expressed
5  % in the earth frame to the body frame angular velocity
6  % components p, q and r. The function needs to know
7  % also the orientation of the body frame in Euler
```

```
8  % angles.
9  % ------------------------------------------------------------
10
11 % Euler angles
12 phi = ang(1);
13 theta = ang(2);
14 % Note: the angle psi is not actually needed by the function.
15
16 A = [1  0 -sin(theta);
17      0  cos(phi)  sin(phi)*cos(theta);
18      0 -sin(phi)  cos(phi)*cos(theta)];
19
20 pqr = A*dang;
```

The IMU subsystem contains also the code that computes the pressure as a function of the altitude, inside the `Environmental variables` block:

Listing B.7: Pressure as a function of the altitude.

```
1  function P = pressure_altitude(pos, P0, T, Ra, g)
2  %#codegen
3  % ------------------------------------------------------------
4  % This function calculates the value of the
5  % atmospheric pressure given the altitude and
6  % the temperature.
7  % The pressure at sea level P0 is assumed to be a
8  % constant parameter.
9  % ------------------------------------------------------------
10
11 % Extract the altitude from the position vector.
12 z = pos(3);
13
14 P = P0*exp(-(g*z)/(T*Ra));
```

Continuous time EKF

The continuous time extended Kalman filter implementation contains many bocks with Matlab code: the state-transition model, the sensor model, the covariance update and the Jacobians are all computed in different blocks.

Listing B.8: State-transitiom model.

```
1  function f = state_transition(x, U, Ix_e, Iy_e, Iz_e, g_e, m_e)
2  %#codegen
3  %
4  % ------------------------------------------------------------
5  % Nonlinear state-transition model of the EKF
6  % ------------------------------------------------------------
7
8  % ------------------------------------------------------------
9  % State variables
```

```
10 %-------------------------------------------------------------
11 % Linear velocity components
12 u = x(1);
13 v = x(2);
14 w = x(3);
15
16 % Angular velocity components
17 p = x(4);
18 q = x(5);
19 r = x(6);
20
21 % Angular position (Euler angles)
22 phi = x(7);
23 theta = x(8);
24
25 f = zeros(10,1);
26
27 %-------------------------------------------------------------
28 % State-transition equations
29 %-------------------------------------------------------------
30 f(1) = r*v - q*w + g_e*sin(theta);
31 f(2) = p*w - r*u - g_e*cos(theta)*sin(phi);
32 f(3) = q*u - p*v - g_e*cos(theta)*cos(phi) + U(1)/m_e;
33 f(4) = (Iy_e - Iz_e)*q*r/Ix_e + U(2)/Ix_e;
34 f(5) = (Iz_e - Ix_e)*p*r/Iy_e + U(3)/Iy_e;
35 f(6) = (Ix_e - Iy_e)*p*q/Iz_e + U(4)/Iz_e;
36 f(7) = p + sin(phi)*tan(theta)*q + cos(phi)*tan(theta)*r;
37 f(8) = cos(phi)*q - sin(phi)*r;
38 f(9) = (sin(phi)*q)/cos(theta) + (cos(phi)*r)/cos(theta);
39 f(10) = -sin(theta)*u - cos(theta)*sin(phi)*v + cos(theta)*cos(phi)*w;
```

Listing B.9: Sensor model.

```
1  function y = sensor_model(x, U, Ma_e, ba_e, Mg_e, bg_e, G_e, mf_e, ...
       bm_e, ...
2                            kb_e, P0_e, g_e, Ra_e, T0_e, bb_e, m_e)
3  %#codegen
4  %-------------------------------------------------------------
5  % Sensor model of the EKF
6  %
7  % These equations compute the expected sensor output
8  % for a given value of x. The equations have a lot of
9  % parameters, which represent the estimated or measured
10 % values of the environmental variables and the sensor
11 % calibration parameters.
12 %-------------------------------------------------------------
13
14 %-------------------------------------------------------------
15 % State variables
16 %-------------------------------------------------------------
17 % Linear velocity components
18 u = x(1);
```

```
19 v = x(2);
20 w = x(3);
21
22 % Angular velocity components
23 p = x(4);
24 q = x(5);
25 r = x(6);
26
27 % Angular position (Euler angles)
28 phi = x(7);
29 theta = x(8);
30 psi = x(9);
31
32 % Altitude
33 z = x(10);
34
35 %------------------------------------------------------------
36 % Rotation matrix
37 %------------------------------------------------------------
38 R = [cos(psi)*cos(theta), ...
      cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi),...
39          cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
40        sin(psi)*cos(theta), ...
            sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi),...
41          sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
42       -sin(theta) cos(theta)*sin(phi) cos(theta)*cos(phi)];
43
44 %------------------------------------------------------------
45 % Sensor equations
46 %------------------------------------------------------------
47 % Accelerometer
48 ya = Ma_e*[r*v - q*w; p*w - r*u; q*u - p*v + U(1)/m_e] + ba_e;
49
50 % Gyroscope
51 yg = Mg_e*[p; q; r] + bg_e;
52
53 % Magnetometer
54 ym = G_e*R'*mf_e + bm_e;
55
56 % Altimeter (barometer)
57 yb = kb_e*P0_e*exp(-(g_e*z)/(Ra_e*T0_e)) + bb_e;
58
59 % Output
60 y = [ya; yg; ym; yb];
```

Listing B.10: Covariance update.

```
1 function [K, dP] = covar_update(P, H, F, Q_e, R_e)
2 %#codegen
3 %------------------------------------------------------------
4 % Covariance update and Kalman gain
5 %
```

```
% This differential equation updates the covariance
% matrix of the probability distribution and computes
% the Kalman gain.
% -------------------------------------------------------

% Kalman gain
K = P*H'/R_e;

% Covariance update
dP = F*P + P*F' - K*R_e*K' + Q_e;
```

Listing B.11: Jacobian of the state-transition model.

```
function H = H(x, Ma_e, Mg_e, G_e, kb_e, g_e, P0_e, Ra_e, T0_e, mf_e)
%#codegen
% -------------------------------------------------------
% Jacobian matrix that linearizes the sensor model
% equations.
% -------------------------------------------------------

% -------------------------------------------------------
% State variables
% -------------------------------------------------------
% Linear velocity components
u = x(1);
v = x(2);
w = x(3);

% Angular velocity components
p = x(4);
q = x(5);
r = x(6);

% Angular position (Euler angles)
phi = x(7);
theta = x(8);
psi = x(9);

% Altitude
z = x(10);

% -------------------------------------------------------
% Partial derivatives of the rotation matrix with
% respect to phi, theta and psi.
% -------------------------------------------------------
Rdphi = [
    0, 0, 0;

    cos(psi)*sin(theta)*cos(phi) + sin(psi)*sin(phi),...
    sin(psi)*sin(theta)*cos(phi) - cos(psi)*sin(phi),...
    cos(theta)*cos(psi);

```

```
40      -cos(psi)*sin(theta)*sin(phi) + sin(psi)*cos(phi),...
41      -sin(psi)*sin(theta)*sin(phi) - cos(psi)*cos(phi),...
42      -cos(theta)*sin(phi)
43      ];
44
45  Rdtheta = [
46      -cos(psi)*sin(theta),...
47      -sin(psi)*sin(theta),...
48      -cos(theta);
49
50      cos(psi)*cos(theta)*sin(phi),...
51      sin(psi)*cos(theta)*sin(phi),...
52      -sin(theta)*sin(phi);
53
54      cos(psi)*cos(theta)*cos(phi),...
55      sin(psi)*cos(theta)*cos(phi),...
56      -sin(theta)*cos(phi)
57      ];
58
59  Rdpsi = [
60      -sin(psi)*cos(theta), cos(psi)*cos(theta), 0;
61
62      -sin(psi)*sin(theta)*sin(phi) - cos(psi)*cos(phi),...
63      cos(psi)*sin(theta)*sin(phi) - sin(psi)*cos(phi),...
64      0;
65
66      -sin(psi)*sin(theta)*cos(phi) + cos(psi)*sin(phi),...
67      cos(psi)*sin(theta)*cos(phi) + sin(psi)*sin(phi),...
68      0
69      ];
70
71
72  % ------------------------------------------------------------
73  % Linearizations of the sensor model equations
74  % ------------------------------------------------------------
75  % Accelerometer
76  J1 = Ma_e*[0  r -q; -r  0  p; q -p  0];
77  J2 = Ma_e*[0 -w  v; w  0 -u; -v  u  0];
78
79  % Magnetometer
80  J3 = [G_e*Rdphi*mf_e G_e*Rdtheta*mf_e G_e*Rdpsi*mf_e];
81
82  % Altimeter
83  Ja = -(kb_e*P0_e*g_e)/(Ra_e*T0_e)*exp(-(g_e*z)/(Ra_e*T0_e));
84
85  % Final matrix
86   H = eye(10);
87   H(1:3,1:3) = J1;
88   H(1:3,4:6) = J2;
89   H(4:6,4:6) = Mg_e;
90   H(7:9,7:9) = J3;
91   H(10,10) = Ja;
```

Listing B.12: Jacobian of the sensor model.

```
function F = F(x, g_e, Ix_e, Iy_e, Iz_e)
%#codegen

% -----------------------------------------------------------
% Jacobian matrix that linearizes the state-transition
% model equations.
% -----------------------------------------------------------

% -----------------------------------------
% State variables
% -----------------------------------------
% Linear velocity components
u = x(1);
v = x(2);
w = x(3);

% Angular velocity components
p = x(4);
q = x(5);
r = x(6);

% Angular position (Euler angles)
phi = x(7);
theta = x(8);

% Note: the variables psi and z are not required for the Jacobian of
% the state-transition model

% Initialize the matrix
F = zeros(10);

% -----------------------------------------
% Nonzero entries of the Jacobian matrix.
% -----------------------------------------
F(1,2) = r;
F(1,3) = -q;
F(1,5) = -w;
F(1,6) = v;
F(1,8) = g_e*cos(theta);
F(2,1) = -r;
F(2,3) = p;
F(2,4) = w;
F(2,6) = -u;
F(2,7) = -g_e*cos(theta)*cos(phi);
F(2,8) = g_e*sin(theta)*sin(phi);
F(3,1) = q;
F(3,2) = -p;
F(3,4) = -v;
F(3,5) = u;
F(3,7) = g_e*cos(theta)*sin(phi);
F(3,8) = g_e*sin(theta)*cos(phi);
F(4,5) = (Iy_e - Iz_e)*r/Ix_e;
```

```
53 F(4,6) = (Iy_e - Iz_e)*q/Ix_e;
54 F(5,4) = (Iz_e - Ix_e)*r/Iy_e;
55 F(5,6) = (Iz_e - Ix_e)*p/Iy_e;
56 F(6,4) = (Ix_e - Iy_e)*q/Iz_e;
57 F(6,5) = (Ix_e - Iy_e)*p/Iz_e;
58 F(7,4) = 1;
59 F(7,5) = sin(phi)*tan(theta);
60 F(7,6) = cos(phi)*tan(theta);
61 F(7,7) = cos(phi)*tan(theta)*q - sin(phi)*cos(theta)*q;
62 F(7,8) = (sin(phi)*q + cos(phi)*r)/cos(theta)^2;
63 F(8,5) = cos(phi);
64 F(8,6) = -sin(phi);
65 F(8,7) = -cos(phi)*r - sin(phi)*q;
66 F(9,5) = sin(phi)/cos(theta);
67 F(9,6) = cos(phi)/cos(theta);
68 F(9,7) = (cos(phi)*q - sin(phi)*r)/cos(theta);
69 F(9,8) = sin(theta)*(sin(phi)*r + cos(phi)*q)/cos(theta)^2;
70 F(10,1) = -sin(theta);
71 F(10,2) = cos(theta)*sin(phi);
72 F(10,3) = cos(theta)*cos(phi);
73 F(10,7) = cos(theta)*cos(phi)*v - cos(theta)*sin(phi)*w;
74 F(10,8) = -cos(theta)*u - sin(theta)*sin(phi)*v - ...
       cos(theta)*sin(theta)*w;
```

Discrete time EKF

The discrete time version of the extended Kalman filter contains only two blocks with Matlab code: one perform the prediction step and the other one performs the update step.

Listing B.13: Prediction step of the discrete time EKF.

```
1  function [x_p, P_p] = EKF_predict(x_old, U_old, P_old, Ix_e, Iy_e, ...
       Iz_e, ...
2                                     g_e, m_e, Ts, Q_e)
3  %#codegen
4  % ------------------------------------------------------------
5  % PREDICTION STEP of the discrete time EKF
6  % ------------------------------------------------------------
7
8  % ------------------------------------------------------------
9  % State variables
10 % ------------------------------------------------------------
11 % Linear velocity components
12 u_old = x_old(1);
13 v_old = x_old(2);
14 w_old = x_old(3);
15
16 % Angular velocity components
17 p_old = x_old(4);
18 q_old = x_old(5);
```

```
19 r_old = x_old(6);
20
21 % Angular position (Euler angles)
22 phi_old = x_old(7);
23 theta_old = x_old(8);
24
25 % Note: the state variables psi_old and z_old are not required
26 % in this step.
27
28 %-------------------------------------------------------
29 % State prediction
30 %-------------------------------------------------------
31 delta_x = zeros(10,1);
32
33 % State equations discretized using the forward Euler method.
34 delta_x(1) = r_old*v_old - q_old*w_old + g_e*sin(theta_old);
35 delta_x(2) = p_old*w_old - r_old*u_old - g_e*cos(theta_old)*sin(phi_old);
36 delta_x(3) = q_old*u_old - p_old*v_old - g_e*cos(theta_old)*cos(phi_old) +...
37     U_old(1)/m_e;
38 delta_x(4) = (Iy_e - Iz_e)*q_old*r_old/Ix_e + U_old(2)/Ix_e;
39 delta_x(5) = (Iz_e - Ix_e)*p_old*r_old/Iy_e + U_old(3)/Iy_e;
40 delta_x(6) = (Ix_e - Iy_e)*p_old*q_old/Iz_e + U_old(4)/Iz_e;
41 delta_x(7) = p_old + sin(phi_old)*tan(theta_old)*q_old +...
42     cos(phi_old)*tan(theta_old)*r_old;
43 delta_x(8) = cos(phi_old)*q_old - sin(phi_old)*r_old;
44 delta_x(9) = (sin(phi_old)*q_old)/cos(theta_old) +...
45     (cos(phi_old)*r_old)/cos(theta_old);
46 delta_x(10) = -sin(theta_old)*u_old - cos(theta_old)*sin(phi_old)*v_old +...
47     cos(theta_old)*cos(phi_old)*w_old;
48
49 x_p = x_old + Ts*delta_x;
50
51 %-------------------------------------------
52 % Covariance prediction
53 %-------------------------------------------
54 % Partial derivatives of the state equations.
55 F_old = eye(10);
56 F_old(1,2) = r_old;
57 F_old(1,3) = -q_old;
58 F_old(1,5) = -w_old;
59 F_old(1,6) = v_old;
60 F_old(1,8) = g_e*cos(theta_old);
61 F_old(2,1) = -r_old;
62 F_old(2,3) = p_old;
63 F_old(2,4) = w_old;
64 F_old(2,6) = -u_old;
65 F_old(2,7) = -g_e*cos(theta_old)*cos(phi_old);
66 F_old(2,8) = g_e*sin(theta_old)*sin(phi_old);
67 F_old(3,1) = q_old;
68 F_old(3,2) = -p_old;
69 F_old(3,4) = -v_old;
70 F_old(3,5) = u_old;
71 F_old(3,7) = g_e*cos(theta_old)*sin(phi_old);
```

```
72  F_old(3,8) = g_e*sin(theta_old)*cos(phi_old);
73  F_old(4,5) = (Iy_e - Iz_e)*r_old/Ix_e;
74  F_old(4,6) = (Iy_e - Iz_e)*q_old/Ix_e;
75  F_old(5,4) = (Iz_e - Ix_e)*r_old/Iy_e;
76  F_old(5,6) = (Iz_e - Ix_e)*p_old/Iy_e;
77  F_old(6,4) = (Ix_e - Iy_e)*q_old/Iz_e;
78  F_old(6,5) = (Ix_e - Iy_e)*p_old/Iz_e;
79  F_old(7,4) = 1;
80  F_old(7,5) = sin(phi_old)*tan(theta_old);
81  F_old(7,6) = cos(phi_old)*tan(theta_old);
82  F_old(7,7) = 1 + cos(phi_old)*tan(theta_old)*q_old - ...
        sin(phi_old)*cos(theta_old)*q_old;
83  F_old(7,8) = (sin(phi_old)*q_old + cos(phi_old)*r_old)/cos(theta_old)^2;
84  F_old(8,5) = cos(phi_old);
85  F_old(8,6) = -sin(phi_old);
86  F_old(8,7) = -cos(phi_old)*r_old - sin(phi_old)*q_old;
87  F_old(9,5) = sin(phi_old)/cos(theta_old);
88  F_old(9,6) = cos(phi_old)/cos(theta_old);
89  F_old(9,7) = (cos(phi_old)*q_old - sin(phi_old)*r_old)/cos(theta_old);
90  F_old(9,8) = sin(theta_old)*(sin(phi_old)*r_old + ...
        cos(phi_old)*q_old)/cos(theta_old)^2;
91  F_old(10,1) = -sin(theta_old);
92  F_old(10,2) = cos(theta_old)*sin(phi_old);
93  F_old(10,3) = cos(theta_old)*cos(phi_old);
94  F_old(10,7) = cos(theta_old)*cos(phi_old)*v_old - ...
        cos(theta_old)*sin(phi_old)*w_old;
95  F_old(10,8) = -cos(theta_old)*u_old - sin(theta_old)*sin(phi_old)*v_old ...
        -...
96       cos(theta_old)*sin(theta_old)*w_old;
97
98  F_old = Ts*F_old;
99
100 % Covariance matrix update
101 P_p = F_old*P_old*F_old' + Q_e;
```

Listing B.14: Update step of the discrete time EKF.

```
1  function [x, P, K] = EKF_update(x_p, P_p, sensors, U, Ma_e, ba_e, Mg_e, ...
       bg_e, ...
2       G_e, mf_e, bm_e, kb_e, P0_e, g_e, Ra_e, T0_e, bb_e, m_e, R_e)
3  %#codegen
4  %-------------------------------------------------------------------
5  % UPDATE STEP of the EKF
6  %-------------------------------------------------------------------
7
8  %-------------------------------------------------------------------
9  % State variables
10 %-------------------------------------------------------------------
11 % Linear velocity components
12 u = x_p(1);
13 v = x_p(2);
14 w = x_p(3);
```

```
15
16 % Angular velocity components
17 p = x_p(4);
18 q = x_p(5);
19 r = x_p(6);
20
21 % Angular position (Euler angles)
22 phi = x_p(7);
23 theta = x_p(8);
24 psi = x_p(9);
25
26 % Altitude
27 z = x_p(10);
28
29 %-------------------------------------------------------
30 % Rotation matrix
31 %-------------------------------------------------------
32 R = [cos(psi)*cos(theta), ...
       cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi),...
33      cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi);
34        sin(psi)*cos(theta), ...
              sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi),...
35        sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi);
36        -sin(theta) cos(theta)*sin(phi) cos(theta)*cos(phi)];
37
38 %-------------------------------------------------------
39 % Sensor equations
40 %-------------------------------------------------------
41 % Accelerometer
42 ya = Ma_e*[r*v - q*w; p*w - r*u; q*u - p*v + U(1)/m_e] + ba_e;
43
44 % Gyroscope
45 yg = Mg_e*[p; q; r] + bg_e;
46
47 % Magnetometer
48 ym = G_e*R'*mf_e + bm_e;
49
50 % Altimeter (barometer)
51 yb = kb_e*P0_e*exp(-(g_e*z)/(Ra_e*T0_e)) + bb_e;
52
53
54 %-------------------------------------------------------
55 % Partial derivatives of the rotation matrix
56 % with respect to phi, theta and psi.
57 %-------------------------------------------------------
58 Rdphi = [
59      0, 0, 0;
60
61      cos(psi)*sin(theta)*cos(phi) + sin(psi)*sin(phi),...
62      sin(psi)*sin(theta)*cos(phi) - cos(psi)*sin(phi),...
63      cos(theta)*cos(psi);
64
65      -cos(psi)*sin(theta)*sin(phi) + sin(psi)*cos(phi),...
```

```
66      -sin(psi)*sin(theta)*sin(phi) - cos(psi)*cos(phi),...
67      -cos(theta)*sin(phi)
68      ];
69
70  Rdtheta = [
71      -cos(psi)*sin(theta),...
72      -sin(psi)*sin(theta),...
73      -cos(theta);
74
75      cos(psi)*cos(theta)*sin(phi),...
76      sin(psi)*cos(theta)*sin(phi),...
77      -sin(theta)*sin(phi);
78
79      cos(psi)*cos(theta)*cos(phi),...
80      sin(psi)*cos(theta)*cos(phi),...
81      -sin(theta)*cos(phi)
82      ];
83
84  Rdpsi = [
85      -sin(psi)*cos(theta), cos(psi)*cos(theta), 0;
86
87      -sin(psi)*sin(theta)*sin(phi) - cos(psi)*cos(phi),...
88      cos(psi)*sin(theta)*sin(phi) - sin(psi)*cos(phi),...
89      0;
90
91      -sin(psi)*sin(theta)*cos(phi) + cos(psi)*sin(phi),...
92      cos(psi)*sin(theta)*cos(phi) + sin(psi)*sin(phi),...
93      0
94      ];
95
96  %-------------------------------------------------------
97  % Jacobian of the sensor equations.
98  %-------------------------------------------------------
99  % Accelerometer
100 J1 = Ma_e*[0  r -q; -r  0  p; q -p  0];
101 J2 = Ma_e*[0 -w  v; w  0 -u; -v  u  0];
102
103 % Magnetometer
104 J3 = [G_e*Rdphi*mf_e G_e*Rdtheta*mf_e G_e*Rdpsi*mf_e];
105
106 % Altimeter
107 Ja = -(kb_e*P0_e*g_e)/(Ra_e*T0_e)*exp(-(g_e*z)/(Ra_e*T0_e));
108
109 % Final matrix
110  H = zeros(10);
111  H(1:3,1:3) = J1;
112  H(1:3,4:6) = J2;
113  H(4:6,4:6) = Mg_e;
114  H(7:9,7:9) = J3;
115  H(10,10) = Ja;
116
117 %-------------------------------------------------------
118 % State update
```

```
119 % ------------------------------------------------------------
120 % Measurement residual
121 y_res = sensors - [ya; yg; ym; yb];
122
123 % Innovation covariance
124 S = H*P_p*H' + R_e;
125
126 % Kalman gain
127 K = P_p*H'/S;
128
129 % Updated state estimate
130 x = x_p + K*y_res;
131
132 % Updated estimated covariance
133 P = (eye(10) - H*K)*P_p;
```

3D animation

The code of the block `3D scope` draws the quadrotor in a tridimensional environment at different time frames, producing an animation.

Listing B.15: Code of the block `3D scope`.

```
1  function draw_mod(position)
2  % ------------------------------------------------------------
3  % Draws the quadrotor in 3d during the simulation.
4  % This script is invoked once for each animation frame.
5  % ------------------------------------------------------------
6  %#codegen
7
8  global old_position;
9  global index_view;
10 global quad;
11
12 x=position(1);
13 y=position(2);
14 z=position(3);
15 phi=position(4);
16 theta=position(5);
17 psi=position(6);
18
19 % Code executed only the first time that the script is
20 % called.
21 if index_view == 0
22
23     % Initialize the figure
24     screen = get(0,'screensize');
25     visual = figure(1);
26     set(visual,'position',[2 65 screen(3)-4 screen(4)-170]);
27     clf(visual);
28     hold on;
29     cameratoolbar('show');
```

```
30      axis vis3d;
31      view(3);
32      zoom(0.6);
33
34      % The following two lines can be deleted for better performances.
35      set(gcf,'menubar','figure','renderer','opengl');
36      set(gca,'Visible','On','Box','On','XGrid', 'on','YGrid', ...
            'on','ZGrid',...
37          'on','projection','perspective');
38
39      % Draw fixed frame reference
40      line([0,0.5],[0,0],[0,0],'linewidth',2,'color','red');
41      line([0,0],[0,0.5],[0,0],'linewidth',2,'color','black');
42      line([0,0],[0,0],[0,0.5],'linewidth',2,'color','green');
43      line([-1,1],[1,1],[0,0],'linewidth',2,'color','black');
44      line([-1,1],[-1,-1],[0,0],'linewidth',2,'color','black');
45      line([1,1],[-1,1],[0,0],'linewidth',2,'color','black');
46      line([-1,-1],[-1,1],[0,0],'linewidth',2,'color','black');
47
48      text(0.6,0,0,'X','fontsize',13);
49      text(0,0.6,0,'Y','fontsize',13);
50      text(0,0,0.6,'Z','fontsize',13);
51
52  % This part is not executed the first time the script
53  % is called.
54  else
55
56      % Delete the quadrotor drawing in the old position
57      drawnow;
58      delete(quad.a);
59      delete(quad.b);
60      delete(quad.c);
61      delete(quad.d);
62      delete(quad.e);
63      delete(quad.f);
64      line([old_position(1),x],[old_position(2),y],[old_position(3),z],...
65          'linewidth',1,'color','yellow');
66  end
67
68  % Rotation matrix
69  rot=[cos(psi)*cos(theta), ...
        -sin(psi)*cos(phi)+cos(psi)*sin(theta)*sin(phi),...
70       sin(psi)*sin(phi)+cos(psi)*sin(theta)*cos(phi);...
71        sin(psi)*cos(theta),  ...
              cos(psi)*cos(phi)+sin(psi)*sin(theta)*sin(phi),...
72       -cos(psi)*sin(phi)+sin(psi)*sin(theta)*cos(phi);...
73       -sin(theta), cos(theta)*sin(phi), cos(theta)*cos(phi)];
74
75  % Quadrotor sketch
76  circ_x=[0.15 0.1 0 -0.1 -0.15 -0.1 0 0.1 0.15];
77  circ_y=[0 0.1 0.15 0.1 0 -0.1 -0.15 -0.1 0];
78  circ_z=[0 0 0 0 0 0 0 0 0];
79
```

```
80 % Draw the quadrotor
81 points=[-0.25 0.25;0 0;0 0];
82 quad.a=line(x+rot(1,:)*points,y+rot(2,:)*points,z+rot(3,:)*points,...
83     'linewidth',2,'color','black');
84 points=[0 0;-0.25 0.25;0 0];
85 quad.b=line(x+rot(1,:)*points,y+rot(2,:)*points,z+rot(3,:)*points,...
86     'linewidth',2,'color','black');
87 points=[circ_x+0.25;circ_y;circ_z];
88 quad.c=line(x+rot(1,:)*points,y+rot(2,:)*points,z+rot(3,:)*points,...
89     'linewidth',2,'color','red');
90 points=[circ_x;circ_y+0.25;circ_z];
91 quad.d=line(x+rot(1,:)*points,y+rot(2,:)*points,z+rot(3,:)*points,...
92     'linewidth',2,'color','blue');
93 points=[circ_x-0.25;circ_y;circ_z];
94 quad.e=line(x+rot(1,:)*points,y+rot(2,:)*points,z+rot(3,:)*points,...
95     'linewidth',2,'color','blue');
96 points=[circ_x;circ_y-0.25;circ_z];
97 quad.f=line(x+rot(1,:)*points,y+rot(2,:)*points,z+rot(3,:)*points,...
98     'linewidth',2,'color','blue');
99
100 % Save the current position for the path plot
101 old_position=[x,y,z];
102
103 % Set the camera position and target
104 camtarget_x=position(1)/2;
105 camtarget_y=position(2)/2;
106 camtarget_z=position(3)/2;
107 campos_x=(camtarget_x/2+camtarget_y)*6-2;
108 campos_y=(camtarget_y/2-camtarget_x)*6-1;
109 campos_z=camtarget_z+sqrt(campos_x^2+campos_y^2)/6+3;
110
111 camtarget([camtarget_x,camtarget_y,camtarget_z]);
112 campos([campos_x,campos_y,campos_z]);
113
114 % Count the iterations
115 index_view = index_view + 1;
```