UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
MECÁNICA - MECATRÓNICA

DISEÑO, IMPLEMENTACIÓN Y VALIDACIÓN EXPERIMENTAL DE ALGORITMOS DE ESTIMACIÓN PARA LA NAVEGACIÓN DE VEHÍCULOS AUTÓNOMOS

TESIS
QUE PARA OPTAR POR EL GRADO DE:
DOCTOR EN INGENIERÍA

PRESENTA:
GIOVANNA AHUATZIN FLORES

TUTOR
DR. YU TANG XU, FI-UNAM

MIEMBROS DEL COMITÉ TUTOR
DR. MARCELO LÓPEZ PARRA, FI-UNAM
DR. VÍCTOR JAVIER GONZÁLEZ VILLELA, FI-UNAM
DR. EDMUNDO GABRIEL ROCHA COZÁTL, FI-UNAM
DR. CARLOS ROMO FUENTES, FI-UNAM

Juriquilla, Querétaro, México, septiembre 2022.

Universidad Nacional Autonoma de Mexico

Faculty of Engineering

Design, Implementation, and Experimental Validation of Estimation Algorithms for Autonomous Vehicle Navigation

T H E S I S

This dissertation is submitted for the degree of:

Doctor in Mechatronics Engineering

PRESENTS:

Giovanna Ahuatzin Flores

ADVISOR:
Dr. Yu Tang

Juriquilla, Querétaro, September 2022

Dedication

To my parents:

For their understanding, love, unconditional support, help; for inculcating in me the principles and values that have contributed to my personal and academic formation. For always encouraging me to follow my dreams.

To my family:

For being a source of inspiration in my life, I love you all.

To my husband:

For being a great support in my life, for being my life partner and being there for me, always.

To my gandmom † Sofía:

As a tribute to her life, for being a very strong woman and for the unconditional love she gave that transcends time and space.

To my friends:

For supporting me and giving me words of encouragement throughout my life, for sharing your dreams and thoughts with me.

Acknowledgements

My most sincere gratitude to all the people and institutions who made possible the accomplishment of this research work and for all the support throughout the past five years.

First of all, I am grateful to my advisor, Dr. Yu Tang Xu for all his valuable lectures, patience, guidance and mentoring from whom I have learned many lessons during my doctoral studies and who has contributed greatly to my academic development.

I would like to thank my committee: Dr. Marcelo López Parra, Dr. Víctor Javier González Villela, Dr. Edmundo Gabriel Rocha Cózatl and Dr. Carlos Romo Fuentes, for their valuable discussions and feedback that have enriched this work; for reviewing this dissertation and for all the support that they have given me at all times.

I am very grateful with the Universidad Nacional Autonoma de Mexico through the Facultad de Ingeniería, the Unidad de Alta Tecnología campus Juriquilla in Querétaro, for the facilities, infrastructure and financial support under the grants PAPIIT-UNAM IN112421 and PAPIIT-UNAM IT103920, highlighting that all experiments were carried out at the National Laboratory of Automobile and Aerospace Engineering LN-INGEA.

A special recognition and my deepest gratitude to CONACYT for the financial support through the scholarship I received during my doctoral studies and the project SEP-CONACyT No. 253677, for promoting technological development in the universities and for the training of human resources in the country.

Thank you to our research group integrated by Eduardo Espíndola López, Rodolfo Ramírez, Larry Serratos and Miguel Angel Verdi Resendiz because they have been an enormous support in the exchange of ideas, in the process of brainstorming, for discussions and collaborations in the realization of the projects. Also, I appreciate all members of UAT community, those who have accompanied me throughout my academic career and from whom I have learned a lot: Mireya Abigailh Ortega Ontiveros, Osiris Ricardo Torres, Socorro Armenta Servin, Rafael Chávez Moreno, Guadalupe Ortega Ontiveros, Juan Carlos Sánchez Villegas, Cristóbal David and Jaime Rueda.

A big thank you, I would like to express to Dr. Scott Palo, who leads Maxwell Cubesat Project for accepting me as a collaborator for a research stay at the University of Colorado at Boulder, Colorado, United States of America, through the Ann and H.J. Smead Department of Aerospace Engineering Sciences; for all his support during the doctoral stay that I completed during november 2019 to october 2020. Thank you for all the experiences and teachings that impacted my academic life in understanding the Attitude Determination and Control System in depth. Thanks also to all the team members with whom I had the good fortune to work: Aaron Aboaf, Arunima Prakash, Anastasia Muszynski, Vikas Natajara,

Matt Zola, Elliott Harrod, Robert Redfern and Bailey Roker.

I really appreciate the expertise and knowledge shared by Matt Keat (Ardupilot community), Dr. Antonio Arciniega Nevarez (Universidad Autónoma de Guanajuato, México), MEng. Salvador Martínez Regil and MEng. Jaime Correa Rodríguez.

Thanks to the love, patience, accompaniment and understanding of my beloved husband, Lorenzo Hernández Díaz for trusting in me, believing that I can achieve my dreams and supporting my life projects.

To my parents, Rosalva Flores Mendieta and Celedonio Enrique Ahuatzin Morales, who have always trusted me, for motivating me every day, for being an unconditional support in my life. To my whole family, for being an important motivation that encourages me to move forward and are an example of hard work and effort; to my brothers Víctor Manuel Ahuatzin Flores, Omar Enrique Ahuatzin Flores, my aunt Reyna Flores Mendieta; to my cousins Andrea Paola Flores Mendieta and Alonso Flores Mendieta. To my sister, Diana Abilene Ahuatzin Flores who is a source of inspiration and an example of courage and strength. To my beloved nephew and nieces for making my life joyful: Víctor Ahuatzin Briones, Fátima Abilene Ahuatzin Briones and Daniela Danaé Ahuatzin Martínez. To my sisters-in-law, whom I admire and respect very much: María Antonieta Briones Conde and Daya Martínez. To my uncle Ruben Flores Mendieta, who fomented in me the curiosity for science and space exploration and to his wife, Leticia Lugo for her countless support.

To my beloved grandmom Sofía Mendieta Flores, to whom I owe everything I am. I will be eternally grateful for everything she did for me, her legacy continues in my life. I always remember her words of wisdom and unconditional love.

To my uncle Sergio Flores Mendieta and aunts Teresa González, Yolanda Flores Maldonado, Silvia Ahuatzin Morales, to my cousin Karina Jiménez and their families as well as my dear great aunt Ana María Arellano Velázquez, to the Hernández Díaz, Ahuatzin Morales families; those who are always in my thoughts and in my heart.

To my dear friends, Roxana Hernández Carro, Daisy Elena Martínez Vázquez, Guadalupe Nallely Espíritu Salvador, José Antonio Arciniega Nevárez, Angélica Sánchez Cortés, María Esther Sevilla López, Oralia Martínez Cázares, Julio Mata, Angel Figueroa Soto, Rosy Aparicio, Shareni Muñoz, Yadira Lizeth Barreto, Jaqueline Vázquez Corona and Iraís Bautista Guzmán, for being part of my life, for all the moments we have shared together and for being there for me at every instant.

Finally, I am very thankful to my beloved cat Shodi, who accompanied me countless sleepless nights while I was studying and she is a very important member of my family.

Abstract

This thesis presents a novel design of a nonlinear observer for navigation of autonomous vehicles, in particular quadrotors, using the information from vector and position measurements directly, available from low-cost sensors like IMU or CCD cameras and a GPS sensor. First, a nonlinear observer of angular velocity is proposed, which ensures global exponential stability proven by the Lyapunov analysis. The estimated angular velocity is then used to estimate the attitude relying on the Explicit Complementary Filter (ECF), which is almost globally asymptotically convergent. The angular velocity observer in cascade with ECF constitutes an observer with cascade structure, guaranteeing almost global asymptotic stability. In contrast with the common approach where the attitude is first estimated and then using the estimated attitude to estimate the angular velocity, the approach proposed here has the advantage that the angular velocity observer is decoupled from the attitude observer. Using the position measurement provided by a GPS, a translational state observer is designed with global exponential stability. The overall observer consists of three cascades modules to estimate the angular velocity, the attitude, and the linear velocity together with the position of a quadrotor, each module is designed independently, ensuring their functionality and robustness to failure. This fact facilitates the convergence analysis and the implementation of the observer in practice.

The performance of the proposed observer was validated by numerical simulations in realistic scenarios, noisy measurements and uncertainties of the inertia matrix were included. Furthermore, an experimental platform based on a quadrotor developed in the Mechatronic Laboratory in the National Laboratory of Automotive and Aerospace Engineering at the *Unidad de Alta Tecnología (UAT)* was developed, and experiments were carried out on this platform to further validate the design of the observer.

The results of the numerical simulations and experiments have shown the convergence to zero of the estimation error, in accordance with the analysis results. Therefore, the proposed observer provides a reliable solution for autonomous vehicle navigation in real-world applications.

Resumen

Esta tesis presenta una propuesta novedosa para el diseñar un observador para la navegación de vehículos autónomos, en particular drones tipo quadrotor, utilizando directamente las mediciones vectoriales y de posición, que se pueden obtener de sensores de bajo costos como IMU o cámaras CCD y un sensor de GPS. En primer lugar, se propone un observador no lineal de velocidad angular, cuya convergencia global exponencial es demostrada mediante análisis de Lyapunov. La velocidad angular estimada se utiliza entonces para estimar la orientación basándose en el Filtro Complementario Explícito (ECF por sus siglas en inglés *Explicit Complementary Filter*), cuya convergencia es casi global y asintótica. El observador de velocidad angular junto con el ECF constituye un observador no lineal completo del estado rotacional con estructura en cascada, garantizando la estabilidad asintótica casi global. Comparando con los resultados reportados en la literatura, el observador propuesto se basa en mediciones vectorial y el diseño del observador del estado rotacional permite desacoplar el observador de velocidad angular con el observador de orientación. Más aún, Usando las medidas proporcionadas por un sensor GPS, un observador del estado translacional (velocidad lineal y posición) es diseñado con estabilidad global exponencial. El observador en conjunto consiste de tres módulos en cascada para estimar la velocidad angular, la orientación y la velocidad lineal junto con la posición de un quadrotor; cada módulo fue diseñado de manera independiente, garantizando su funcionalidad y robustez ante fallas. Esta estructura en cascada facilita el análisis de convergencia y su implementación del observador propuesto.

El desempeño del observador propuesto fue validado a través de simulaciones numéricas en escenarios prácticos, en donde se incluyeron mediciones de sensores contaminadas por ruido y la incertidumbre paramétrica en la matriz de inercia.

Se diseñó y se construyó una plataforma experimental basada en un quadrotor en el Laboratorio de Mecatrónica dentro del Laboratorio Nacional de Ingeniería Automotriz y Aeroespacial en la Unidad de Alta Tecnología de la Facultad de Ingeniería. En esta plataforma se validó el diseño del observador en el quadrotor. Los resultados obtenidos tanto de las simulaciones numéricas como de los experimentos muestran concordancia con los resultados teóricos, proporcionando confiabilidad del observador propuesto en aplicaciones para vehículos autónomos.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we consider the problem of estimating state variables for autonomous vehicle (AV) navigation using on-board sensors. Autonomous vehicles (AVs) require motion stabilization or trajectory tracking, despite external disturbances and sensor imperfections such as drifts in the measured variable and scaling uncertainty. The control system in charge of executing this task needs to know the states of the AV, i.e., position, linear velocity, orientation, and angular velocity, to command orders to the actuators that will regulate the orientation of the vehicle. In a real-world application, it is crucial to develop efficient algorithms that can estimate the state of an AV using the information provided by the onboard sensors. Position and linear velocity can be obtained using the Global Position System (GPS); angular velocity can be obtained by gyroscopes attached to the body, but these sensors are very susceptible to failure; while for attitude there are no sensors that measure it directly, so it is estimated by fusing data from different sensors. Thus, a central question in state estimation tasks is to reconstruct the rotational state (attitude and angular velocity) of the AV [15] so that the implementation of the control strategy helps to ensure a successful mission. Using the reconstructed rotational state, the translational state (position and linear velocity) can be posted as a *linear* estimation/filtering problem, and many systematic solutions, such as Luenberger observer and Kalman filter, are available [23, 37]. Both rotational state and translational state estimation problem are addressed in this thesis.

Since the late 1960s, the problem of estimating the attitude of rigid bodies has been deeply studied and, currently, has gained importance due to the appearance of new schemes for sensor fusion, the development of nonlinear control algorithms, and creation of platforms as test beds. Furthermore, the number and complexity of aerial vehicle applications are increasing continuously, and the attitude and angular velocity estimation techniques involved must also be improved to provide better performance and greater stability of such systems.

This chapter gives a review of the literature on related topics of rotational ans translational state estimation. The problem description, the main objectives of the thesis and the contributions are stated.

1.1 Literature review

1.1.1 Review of the attitude observer

Attitude determination is fundamental for navigation applications to control the orientation of autonomous vehicles. Attitude Determination and Control Subsystem (ADCS) is responsible to accomplish two main objectives: to estimate the orientation of the vehicle as accurately as possible and to point the vehicle to the desired orientation. If the vehicle is not in the desired orientation, then the control system compares the actual orientation with the desired orientation and calculates the error, this information is used to know how much action is needed for each actuator so that the vehicle can be brought to the desired orientation.

Most systems require precise knowledge of the vehicle's orientation to acquire photographs and/or video of a specific point on Earth, to point the vehicle to a specific point in space or to perform tracking and guidance maneuvers, depending on the mission objective. For example, in the case of satellites, pointing the solar panels towards the Sun is vital to recharge the battery and extend the life of the mission or may be required to point to a specific point for Earth observation. In the case of Unmanned Aerial Vehicles (UAVs), the attitude can be significant for cooperative or collaborative tasks between vehicles.

The attitude of a rigid body is represented mathematically through the Euler angles, unit quaternions, rotation matrix, or Rodriguez parameters. A very complete survey of the orientation representations can be found in [43] where twelve different orientation representations and their transformations between them are presented.

The problem of attitude estimation has been deeply explored since 1964, when Black [11] proposed determining attitude using only two vector observations and its corresponding inertial vectors. The basis of attitude estimation was introduced by Wahba in 1965 known as "Wahba's problem" which is formulated as: A body-frame measurement of two non-collinear vectors is available from sensors such as accelerometer, magnetometer, coarse sun sensors, or star trackers; in addition, the information of two reference vectors in the inertial frame must also be known. Then, the direction cosine matrix or rotation matrix, which relates the two sets of measurements, can be estimated. The first solution to solve the Wahba's problem is the TRIAD algorithm. Many solutions have been proposed for attitude estimation, classified mainly in two categories: static or deterministic algorithms and dynamic or stochastic algorithms.

Deterministic approaches include the TRIAD triaxial orientation determination algorithm, Davenport's q-Method, QUaternion EStimator (QUEST), Fast Optimal Attitude Matrix (FOAM), Markley's SVD method, EULER-2 and EULER-n methods, and ESOQ and ESOQ-2 algorithms [30, 42].

The TRI-axial Attitude Determination method (TRIAD) [23, 42, 47], computes the attitude rotation matrix from two non-collinear vectors in the body and inertial frame. Expressed in the inertial frame, vectors v_1^I and v_2^I and vectors v_1^B and v_2^B in the body frame, where $v_1^I = Rv_1^B$, $v_2^I = Rv_2^B$. This method constructs a third vector with the information of

the other two as:

$$s_1 = v_1^I \qquad s_2 = \frac{v_1^I \times v_2^I}{|v_1^I \times v_2^I|} \qquad s_3 = s_1 \times s_2$$

$$r_1 = v_1^B \qquad r_2 = \frac{v_1^B \times v_2^B}{|v_1^B \times v_2^B|} \qquad r_3 = r_1 \times r_2$$

The rotation matrix can be calculated as:

$$R = \sum_{i=1}^{3} s_i r_i^T = [s_1 \quad s_2 \quad s_3][r_1 \quad r_2 \quad r_3]$$

The TRIAD method is easy to implement, but not all the information of the two vector measurements is used, moreover, if more than two sensors are available, we cannot use this method, and if noise is considered, there is no guarantee that the estimated R remains in the rotation group of $SO(3)$. In addition, this method does not consider the reliability of the sensors. To solve these problems, other solutions have been proposed.

The Davenport's method, transformed Wahba's problem into the problem of finding the largest eigenvalue λ_{max} of the symmetric matrix $K \in \mathbb{R}^{4\times4}$.

$$K \triangleq \begin{bmatrix} \sigma & z^T \\ z & S - \sigma I_{3\times3} \end{bmatrix}$$

where $S = B + B^T$, $\sigma = tr(B)$, $B = \sum_{i=1}^{n} k_i v_i^B \times v_i^I$ and $z = [B_{23} - B_{32} \quad B_{31} - B_{13} \quad B_{12} - B_{21}]^T$ and the attitude estimate β can be calculated by solving the equation: $K\beta = \lambda\beta$.

The QUaternion ESTimator (QUEST) algorithm uses quaternion parametrization and is based on Davenport's method. This algorithm solved the following equation:

$$\begin{bmatrix} \sigma & z^T \\ z & S - \sigma I_{3\times3} \end{bmatrix} \begin{pmatrix} 1 \\ q \end{pmatrix} = \lambda_{max} \begin{pmatrix} 1 \\ q \end{pmatrix}$$

where $q = ((\lambda_{max} + m\sigma)I_{3\times3} - S)^{-1}$ is the attitude estimation. This method was proposed by Shuster and was implemented in the Magsat mission in 1979. Numerical analysis shows that this method is not the best at finding the eigenvalues of its characteristic equation, which makes it less robust than Davenport's method [30] but reliable in practice.

The FOAM algorithm was proposed by Markley in 1993; it solves the Wahba problem using only the sensor measurements without taking into account the system model. Compared to the other deterministic methods, it is faster and more robust, although it does not use quaternion parameterization.

On the other hand, dynamics methods first find the best estimate of the true system state using a dynamic model, and then use measurements assuming that they have random noise or corrupted data. Filtering methods can generally provide a more accurate attitude estimate than deterministic methods because they incorporate the memory of past observations.

The Extended Kalman Filter (EKF) is the most widely used algorithm for attitude estimation in autonomous vehicles among dynamic solutions. It has been recognized that

EKF-based approaches are the most widely used for two main reasons, their proven reliability and ease of incorporating other measurement sources that can improve the quality of the estimate or even provide an estimate of additional quantities such as altitude or vertical velocity [14]. It uses a three-step iteration process: updating the measurement, resetting the state vector, and propagating to the next measurement time. Some variants of this algorithm are Multiplicative KF, Additive KF, Unscented KF, Invariant KF. The Kalman filter requires state models and observations to be linear, some of the variants do deal with nonlinearities, but the computational development is increased.

The most recent advances in solving the attitude estimation problem are nonlinear observers and filter-based methods. Nonlinear attitude estimation observers have emerged that consider the problem when a vehicle is susceptible of strong accelerations to improve the accuracy of the estimated attitude by using accelerometer, magnetometer, gyroscope and complemented with GPS or Inertial navigation systems (INS) measurements. Examples of this research line are as follows. Manohy et al. [28] propose the orientation estimation problem formulated directly on the special orthogonal group SO(3). They termed the *direct, passive and complementary filter*. The latter uses gravitational or magnetic field directions obtained from an IMU plus gyroscope readings. This observer does not require an online algebraic reconstruction of attitude and is ideally suited for implementation on embedded hardware platforms due to its low complexity. However, it suffers from possible discontinuities in the bias correction signal when the equivalent rotation angle of the estimated quaternion approaches $\pm\pi$ *rad*. The explicit complementary filter (ECF) proposed in [28] is composed of 1) a prediction term based on the measurement ω and 2) a correction term derived from the vector measurements:

$$\dot{\hat{R}} = \hat{R}_\sigma S(\omega + k_p \sigma_R), \quad \hat{R}(0) = \hat{R}_0 \in SO(3) \tag{1.1}$$

$$\sigma_R = \sum_{i=1}^{n} k_i S(b_i)\hat{b}_i \tag{1.2}$$

where $\hat{R}$ is the attitude estimate, $k_p > 0$ is a positive gain, and $\hat{b}_i = \hat{R}^T r_i$. The expression (1.1) represents the dynamics of a generic *complementary filter* on $SO(3)$.

A weakness of the ECF (1.1) is its requirement of the measured angular velocity ω, which is obtained by a low-cost MEMS gyros in a typical AV application. The main problem with a MEMS gyros is its bias and reliability. The gyro bias which must be corrected [28]. The reliability calls for a backup solution as that we propose in the next section.

Hua [22] proposed an attitude and velocity observer based on the Explicit Complementary Filter, that depends only on inertial measurements, Global Navigation Satellite Systems (GNSS) velocity measurements, and magnetometer measurements. This observer exploits the fact that the vehicle's acceleration vector in the navigation frame is implicitly available in the derivative of the GNSS velocity and, therefore, it can be compared to the accelerometer measurement to help determine the attitude. Stability analysis results state semi-global convergence.

Roberts and Tayebi [38] proposed two nonlinear observers that estimate the orientation of a rigid body using GPS, accelerometer and magnetometer. This observer considers the dynamics of the linear velocity in the correction term resulting in semiglobal exponential convergence. This observer belong to the class of *velocity-aided* attitude observer.

When the objective consists in combining the estimation of the attitude and the filtering of the linear velocity, some invariant attitude observers have been proposed recently [7, 12, 31]. These observers are GPS–aided and are named *invariant* in the sense that they preserve the properties of the Lie Group. A complete overview related to nonlinear attitude observers can be found in [15, 18, 23]. The attitude determination problem has been solved by several authors under the assumption that the attitude and angular rate information is known and available. But, the research becomes more challenging when angular velocity measurements are unreliable or unavailable due to faulty gyroscopes. This leads us to the problem of estimating the angular velocity of a rigid body.

1.1.2 Review of the angular velocity observer

In order to estimate the angular velocity of a rigid body, there are essentially three methods to solve this. The first consists of measuring the angular velocity by means of a sensor, but these sensors are relatively expensive and susceptible to failures, as case studies are presented [27] [8]. For this reason, it is encouraged to select other types of solutions. The second method is known as *two- step approach*, where the first step is to calculate the orientation from vector measurements and use it in the second step to reconstruct the angular velocity. A third technique is to employ a nonlinear observer using vector measurements from on-board sensors to directly calculate the angular velocity and use this estimate to reconstruct the attitude of a rigid body.

The advantage of this type of solution is that it does not require the use of gyroscope measurements and does not require knowing the orientation to estimate the angular velocity. The first paper to propose this solution was [45] relying solely on body vector measurements, which guarantees almost global asymptotic stability; an improved version of this method was presented in [8] where the unstable equilibria of closed-loop dynamics were reduced and the auxiliary error system does not use the inertial fixed reference vectors as in [45]. In this context, the first work that proposed a nonlinear observer that achieves global uniform exponential convergence in the estimate of angular velocity is [41]. This observer consists of a copy of the system plus correction terms and a dynamic extension based on an invariant-manifold framework. Another interesting proposal presented is [9] in which the authors design the angular velocity observer directly on $SO(3) \times \mathbb{R}^3$, leading to results of global exponential stability. Recent progress in the development of new nonlinear observers for implementation and the growing development in new control schemes allow for an active area for research.

Arguments of these observers depend on the angular velocity obtained from a gyroscope plus correction terms that required magnetometer, accelerometer, and GPS/INS measurements; these works assume that the angular velocity is known at all times. However, this is not always possible for real-world applications, since gyroscopes are prone to failure. If the gyroscope data is not available or the gyroscope malfunction, the mission may be compromised, the angular velocity cannot be estimated, and the attitude cannot be estimated. This is the reason why new proposals are encouraged for attitude estimation without gyroscope knowledge.

1.1.3 Review of the translational state observer

Several nonlinear navigation observers have been reported to estimate both the rotational (attitude and angular velocity) and translational state (position and linear velocity) of general autonomous vehicles moving in a 3D space. This approach can be found in literature as the problem of full state estimation for vehicles navigating in a three dimensional space. In order to achieve reliable estimates, these techniques assume that rigid body is equipped with IMU fused with sensors that provide partial or full position measurements such as GPS, motion capture systems, LIDARs, altimeter or Inertial navigation systems (INS). Previous works on this area of research are:

Rehbinder and Ghosh in Ref. [37] used a CCD camera as a vector measurement device used with inertial sensors to estimate the pose (attitude and position) of a rigid body. They formulated that attitude estimation can be detached from the position estimation. Furthermore, that once the attitude is estimated, the position can be treated as a linear implicit problem.

Refs. [6, 10, 19] proposed a navigation observer for UAV using vector measurements and gyro rate of an IMU aided by an INS or a general position sensor to estimate attitude, gyro bias, and translational state. Navigation observers in these references employ the key idea from attitude estimation in [28, 37] to first estimate attitude and gyro bias using vector measurements and the gyro rate simultaneously, and then the translational state is estimated using the attitude estimate.

Using IMU sensors fused with INS are typically used in this class of solutions. But, in case that only the INS is used, the outputs may give unreliable readings since measurement errors and unknown initial conditions cause the estimation error to deviate over time [10]. Using INS assisted with GPS, correct the position estimates over time and keep errors small and bounded. Other types of sensors for position information such as motion cameras systems (such as *Optitrack, Vicon*), GNSS, bearing measurements UltraWide Band (UWB) sensors are excellent alternatives with high measurement accuracy but, therefore, expensive.

For low-cost solutions using commercial GPS sensors combined with nonlinear navigation observers are an excellent alternative to estimate the translational state to produce reliable estimates. For this reason, it is necessary to investigate new proposals within this context.

1.2 Problem description

This thesis focuses on the problem of the design, implementation, and validation of an angular velocity and attitude observer for the navigation of an AV using vector measurements directly without any attitude representation such as rotation matrix or unit quaternions. Vector measurements may be obtained from low-cost sensors such as an IMU or a CCD camera. Contrary to the commonly used two-step method [15], in this thesis we propose a new approach to estimate the rotational state that gives rise to a cascaded observer structure. The observer design is based upon the Lyapunov analysis, ensuring almost global asymptotic convergence of the estimated state.

The proposed observer is validated by numerical simulations under realistic scenarios. Experimental results show further the observer in a real-world application. The experiments

are carried out on a platform, consisting of a quadrotor, sensors and processor on board, and supporting software, constructed in the Mechanical Laboratory of the National Laboratory of Automotive and Aerospace Engineering, *Unidad de Alta Tecnología (UAT)* UNAM campus Juriquilla. This project arises from the need to validate our own control and estimation algorithms developed at the UAT.

1.3 Objectives

The general objective is the design, implementation, and experimental validation of a navigation observer for autonomous vehicles, in particular for UAVs quadrotors.

Specific goals

- Design of a rotational state observer using vector measurements from low-cost sensors, such as an IMU and a CCD camera, to estimate the angular velocity and the attitude of UAVs.
- Design of a rotational state observer using additional position measurements from a GPS sensor to estimate the linear velocity and the position of UAVs.
- Ensuring convergence to the true state in the proposed observer based on the Lyapunov analysis.
- To analyze the performance of the observer through numerical simulations of different conditions and situations.
- To research on suitable open architecture UAVs and technologies that allow the free use of hardware and software to reprogram the control board to test our own algorithms.
- To build a quadrotor for the experimental validation of the proposed system that meets the necessary hardware requirements.
- Create and program different autonomous missions to be executed on the quadrotor.
- Conduct a static thrust test to measure the thrust coefficient and the torque coefficient and perform experiments in the lab to identify the parameters of the quadrotor.
- To run real-time flight experiments in an outdoor environment, analyze flight data.

1.4 Hypothesis

- It is possible to design an angular velocity and attitude observer using directly the vector measurements from a low-cost IMU, in a framework where gyro information is not needed for the calculation.
- It is possible to design a translational state observer that requires GPS inputs in cascade with the rotational state observer.
- It is possible to build a low-cost experimental platform based on a quadrotor to validate the navigation observer.

1.5 Contributions

The contributions of this thesis are a novel deign of navigation observer for autonomous vehicles such as quadrotor, analyzed by Lyapunov stabilty theory and validated by numerical simulations and experiments in laboratory. Specifically:

- An angular velocity observer design that ensures global exponential convergence of the error dynamics, using directly vector measurements from sensors, such as accelerometers, magnetometers, and CCD cameras. Compared to other works in the literature, this algorithm does not need gyroscope measurements or attitude to perform the calculation.
- An attitude observer in cascade with the angular velocity observer based on the explicit complementary filter (ECF) proposed by [28] to estimate the attitude of a rigid body. The proposed angular velocity observer together with the EFC form an overall rotational state observer with a cascaded structure that guarantees almost global asymptotic convergence to the true rotational state. This approach facilitates the converge analysis of both the angular velocity and the attitude observer.
- A translational state observer with exponential convergence, designed independently of the rotational state observer, using the position measurement provided by a GPS sensor.
- An experimental platform, built in the Mechatronics laboratory to test and validate the angular velocity and attitude observer characterized by a low-cost open architecture quadrotor, whose onboard controller uses a *Pixhawk* and is ideal for academic research.

Part of these results have been reported in the work Ahuatzín, Tang and Espíndola (2022) [2].

1.6 Thesis outline

The remainder of this thesis is organized as follows.

Chapter 2. This chapter provides a mathematical background necessary for stability analysis, also presents attitude kinematics and dynamics of a rigid body and some identities for attitude representation.

Chapter 3. This chapter presents a detailed description of the angular velocity observer design and its stability proof through the Lyapunov analysis.

Chapter 4. This chapter designs the attitude observer using the estimated angular velocity in the previous chapter based on the the Explicit Complementary Filter. A rigorous proof of its stability analysis is presented by means of Lyapunov theory.

Chapter 5. This chapter is dedicated to the design of the linear velocity and position observer. From IMU measurements together with a GPS sensor, a new observer for translational state is proposed. Global exponential stability is demonstrated.

Chapter 6. In this chapter, numerical simulations are illustrated to test the performance of the proposed observer. Three scenarios are considered: an ideal case where two reference

vectors in the inertial frame are perpendicular and the on-board vector measurements are noise- free. The second scenario simulates a more realistic situation, where the angle between the reference vectors is small and the vector measurements are contaminated by Gaussian noise. The third scenario shows the robustness of the observer under uncertainty in the inertia matrix.

Chapter 7. This chapter describes the experimental platform developed in the Mechatronics Laboratory based on a quadrotor that worked as a test bed in order to analyze the behavior of the proposed observer in real-world applications. Experimental results are presented and analyzed.

Chapter 8. Finally, in Chapter 8, conclusions and a discussion for future work are presented.

Chapter 2

Mathematical tools

This chapter presents the mathematical tools necessary for the design of the angular velocity and attitude observers and the analysis of the convergence. A brief introduction to the main attitude representations such as the rotation matrix, Euler angles, and quaternions is also presented. Finally, the rotational kinematics and dynamics for an autonomous vehicle are described, which are essential for the design of the proposed observer.

2.1 Attitude representations

To describe the attitude of a rigid body, it is necessary to establish coordinate frames to establish a reference with respect to which the attitude is measured. The following concepts are needed:

- A coordinate system defined in a convenient way according to the application called *Inertial Reference frame*, *I*.
- A coordinate system fixed on the center of mass of the rigid body of interest called *Body Frame B*. All sensors are attached rigidly on the vehicle, thus the measured states are expressed in the body-fixed frame B.

Since there is a variety of attitude representations to choose from, the decision of which one to adopt provides an additional degree of freedom in observer design. This decision is important since different attitude representations may lead to different solutions, appropriate for the underlined application. Some advantages and disadvantages of various attitude representations are presented.

2.1.1 Rotation matrix

The rotation matrix is a linear transformation in Euclidean Space used to rotate vectors and map coordinates from one reference frame to another. Consider two reference frames A and B, and let the position of the rigid body in B denoted by ${}^{B}p \in \mathbb{R}^3$. Then, the position of the rigid body with respect to the reference frame A, is given by:

$$ {}^{A}p = {}^{A}_{B}R\, {}^{B}p, \tag{2.1} $$

where ${}^A_B R$ denotes the rotation matrix of B to A. A rotation matrix R belongs to the group of *Ortogonal Special Matrices* of dimension 3, denoted by $SO(3) = \{R \in \mathbb{R}^{3\times3} | RR^T = I, det(R) = 1\}$.

2.1.2 Euler angles

Representing an attitude by rotation matrices requires 9 scalars. However, the attitude can be defined using fewer parameters. An alternative that uses only 3 parameters is the so-called *Euler angles* representation. This representation is defined by the angles that each reference axis B can rotate, in sequential order, to match the reference frame A.

There are different conventions related to Euler angles that are associated with the order of the axes on which the rotation is to be done. For the case where the convention is $Z-Y-X$, the reference B is first rotated around the z axis by an angle γ, which gives us a reference frame B'. Then B' is rotated around the y axis by an angle *beta* to obtain the reference frame B''. Finally B'' is rotated around the x axis, by an angle *alpha* such that the final reference matches A. This process is illustrated in the figure.

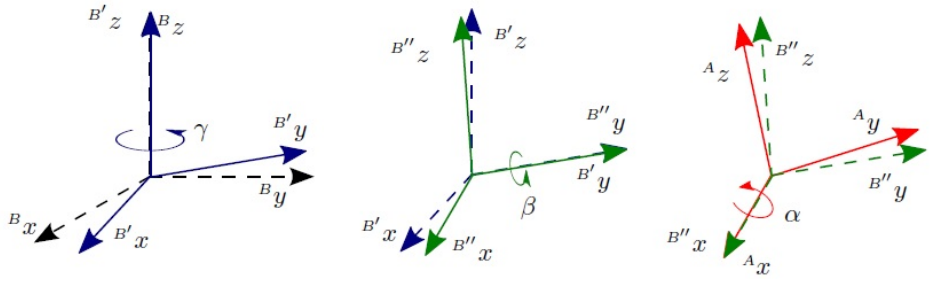


Figure 2.1: Euler angle from B towards A.

Rotation matrix can be obtained from Euler angles using successive rotations around the principal axes, i.e.,

$$ {}^A_B R = R_z(\gamma) R_y(\beta) R_x(\alpha), \tag{2.2} $$

where

$$ R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c\alpha & -s\alpha \\ 0 & s\alpha & c\alpha \end{bmatrix}, R_y(\beta) = \begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}, R_z(\gamma) = \begin{bmatrix} c\gamma & -s\gamma & 0 \\ s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3} $$

and $c\alpha = cos(\alpha)$, $s\alpha = sin(\alpha)$, $c\beta = cos(\beta)$, $s\beta = sin(\beta)$, $c\gamma = cos(\gamma)$, $s\gamma = sin(\gamma)$.

Euler angles can be calculated given a rotation matrix, using the following relations

$$ \begin{aligned} \beta &= atan2\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right), \\ \gamma &= atan2(r_{21}/c\beta, r_{11}/c\beta), \\ \alpha &= atan2(r_{32}/c\beta, r_{33}/c\beta), \end{aligned} $$

where $r_{ij} = \left[{}^{A}_{B}R\right]_{ij}$, $c\beta \neq 0$, and $atan2(x, y)$ is the inverse tangent in the fourth quadrant, which is defined as $atan2(x, y) = 2atan\left(\frac{x}{\sqrt{x^2 + y^2 + z^2}}\right)$. Note that this function will be undefined if $\beta = \pm\frac{\pi}{2}$ rad. The existence of singularities is the main disadvantage of the Euler angle representation. To mitigate this problem, the following convention is sometimes used. Choosing $\gamma = 0$ rad and, if $\beta = \frac{\pi}{2}$, then $\alpha = atan2(r_{12}, r_{22})$. On the other hand, if $\beta = -"\frac{\pi}{2}$ rad, then $-\alpha = atan2(r_{12}, r_{22})$ is taken.

2.1.3 Unit quaternions

The unit quaternion expresses a rotation matrix as a homogeneous quadratic function of the quaternion elements. No trigonometric evaluations or other transcendental functions are required, so they avoid singularities. Unit quaternions are more efficient at specifying rotations than the rotation matrix itself, having only four components instead of nine, and obey only one constraint, the norm constraint, instead of the six constraints imposed on the orientation matrix by orthogonality [30].

Unit quaternions are given by

$$q = q_0 + iq_1 + jq_2 + kq_3 \tag{2.4}$$

where q_0, q_1, q_2 and q_3 are real numbers that satisfy $q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$, are components of a quaternion q and i, j and k are imaginary unit vectors that satisfy $i^2 = j^2 = k^2 = -1$, $ij = -ji = k$, $jk = -kj = i$, $ki = -ik = j$.

A quaternion can also be expressed through its scalar and vector parts, and belongs to the unit sphere of dimension 3 embedded in the space $\mathbb{R}^4$ defined as $\mathcal{S}^3 = \{q \in \mathbb{R}^4 | q^T q = 1\}$, i.e., $q = (q_0, q_v) \in \mathcal{S}^3$, where $q_0 \in \mathbb{R}$ is the scalar part of the quaternion and $q_v = [q_1, q_2, q_3]^T \in \mathbb{R}^3$ is the vector part. With the following product of quaternions $\star$

$$\begin{bmatrix} q_0 \\ q_v \end{bmatrix} \star \begin{bmatrix} \bar{q_0} \\ \bar{q_v} \end{bmatrix} = \begin{bmatrix} q_0\bar{q_0} - q_v^T\bar{q_v} \\ q_0\bar{q_v} + \bar{q_0}q_v + q_v \times \bar{q_v} \end{bmatrix} \tag{2.5}$$

it defines a group whose unit element is $\mathbf{1} = (1, 0)$. Note that the quaternion product is not commutative but associative.

The transformation between a rotation matrix R and a unit quaternion q is uniquely defined by Rodrigues formula:

$$R = I_3 + 2q_0S(q_v) + 2S(q_v)^2. \tag{2.6}$$

Transforming a rotation matrix into quaternions is not so straightforward, since two unit quaternions $\pm q$ correspond to the same rotation matrix R

$$q_0 = \pm\frac{1}{2}\sqrt{1 + tr(R)}, \qquad S(q_v) = \frac{R - R^T}{4q_0}.$$

2.2 Attitude kinematics and dynamics

The kinematics, which relates the angular velocity with the time derivative of the unit quaternion, is given by

$$\dot{R} = RS(\omega), \tag{2.7}$$

where ω is the angular velocity of the rigid body expressed in the body frame and the skew-symmetric operator $S(\cdot)$ satisfied $S(u)v = u \times v$, $\forall u,\ v \in \mathcal{S}^3$. Notice that a rotation matrix has nine scalar components with six constraints.

Let u be a vector $\in \mathbb{R}^3$. The skew-symmetric operator $S(\cdot)$ is defined as:

$$S(u) = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \tag{2.8}$$

Lemma 2.2.1 (Properties for S matrix) *$\forall\ x,\ y \in \mathbb{R}^3$ and $R \in SO(3)$ satisfies the properties*

a1. $S^T(x) = -S(x)$,

a2. $S(x)y = -S(y)x$,

a3. $S(x)S(y) = yx^T - x^Ty$,

a4. $S^2(x) = xx^T - x^Tx$,

a5. $S(S(x)y) = S(x)S(y) - S(y)S(x)$,

a6. $S(R(x)) = RS(x)R^T$

In terms of unit quaternions, the rigid body kinematics can be expressed as

$$\dot{q} = \frac{1}{2}J(q)\omega, \tag{2.9}$$

where $J(q) \in \mathbb{R}^{4\times 3}$ is defined as

$$J(q) = \begin{bmatrix} -q_v^T \\ q_0 I + S(q_v) \end{bmatrix}. \tag{2.10}$$

Some of the properties of this matrix are:

Lemma 2.2.2 (Properties of the matrix $J(q)$) *$\forall\ x,\ y \in \mathbb{R}^4$ the matrix $J(q)$ satisfies the*

properties

b1. $J^T(x)J(x) = \left\|x\right\|_2^2,$

b2. $J^T(x)y = 0 \;\Leftrightarrow y = kx, \quad k \in \mathbb{R},$

b3. $J(\alpha x + \beta y) = \alpha J(x) + \beta J(y) \quad \alpha, \beta \;\in \mathbb{R},$

b4. $J^T(x)y = -J^T(y)x,$

b5. $\left\|J(x)\right\|_2 = \left\|x\right\|_2,$

b6. $\dfrac{d}{dt}(J(x)) = J(\dot{x}),$

The attitude dynamics for a rigid body is given by

$$M\dot{\omega} = S(M\omega)\omega + \tau, \tag{2.11}$$

where $M \in \mathbb{R}^{3\times3}$, $M = M^T > 0$, is the constant inertia matrix and $\tau(t) \in \mathbb{R}^3$ is the control torque vector, both expressed in the body frame.

We recall some identities and math tools useful for the stability analysis.

- The anti-symmetric operator in square matrix space is defined as $P_a^\vee(H) = \frac{1}{2}(H - H^T) \quad \forall H \in \mathbb{R}^{3\times3}$.
- For matrices $A,\ B \in \mathbb{R}^{n\times n}$, the Lie-bracket is defined as $[A, B] = AB - BA$.
- $\forall x \in \mathbb{R}^3$ and $H \in \mathbb{R}^{3\times3}$, $H^T = H$, the trace $tr(HS(x)) = 0$.
- $\forall x,\ y \in \mathbb{R}^3$, $tr(xy^T) = x^Ty$, $S(x \times y) = (yx^T - xy^T)/2$.

2.3 Translational kinematics and dynamics

The inertial position of the center of mass of a rigid body is denoted by $p \in \mathbb{R}^3$ and $v \in \mathbb{R}^3$ is the linear velocity expressed in the inertial frame. The translational kinematics is given by

$$\dot{p} = v. \tag{2.12}$$

Taking into account the thrust $f \in \mathbb{R}$ applied to the center of the mass, the translational dynamics of the rigid body is given by

$$\dot{v} = ge_3 - \frac{f}{m}Re_3, \tag{2.13}$$

where g represents the acceleration of gravity, m is the mass of the vehicle and $e_3 = [0\ 0\ 1]^T$ is the z axis of the inertial frame.

2.4 Sensor modeling

A commercial IMU is commonly composed of a triaxial magnetometer, a triaxial accelerometer, and a triaxial gyroscope.

Accelerometer: The accelerometer provides a measurement of the linear acceleration of the rigid body minus the gravitational acceleration. In reality, these measurements are corrupted by bias and noise. We consider that the raw measurements are modeled as

$$a_B = a_{true} + b_a + \eta_a \tag{2.14}$$

where a_B is the accelerometer measurement, b_a the bias $\in \mathbb{R}^3$, $\eta_a \in \mathbb{R}^3$ the noise. When the vehicle is in stationary motion, the true acceleration a_{true} is given by

$$a_{true} = -e_3 R^T g_0 \tag{2.15}$$

where $\{e_1,\ e_2,\ e_3\} \in \mathbb{R}^3$ is the orthogonal basis of the inertia frame and $g_0 = [0\ 0\ 9.8]^T$ is the acceleration of gravity.

Magnetometer: The magnetometer measures the Earth's magnetic field in the body frame

$$m_B = m_{true} + b_m + \eta_m \tag{2.16}$$

where m_B is the magnetic field measurement, b_m the bias $\in \mathbb{R}^3$ and $\eta_m \in \mathbb{R}^3$ the noise. The component m_{true} is related to the magnetic field of Earth m_I by

$$m_{true} = R^T m_I \tag{2.17}$$

Chapter 3

Design of an angular velocity observer

In this chapter, the problem statement is introduced, followed by a detailed description regarding the mathematical framework for the design of the nonlinear observer as well as the proof of its convergence through Lyapunov analysis.

3.1 Problem statement

Consider a rigid body whose kinematics and dynamics are described by (2.7) and (2.11), respectively. The problem addressed in this chapter is to design an observer to estimate the angular velocity ω using the vector measurements $b_i \in \mathbb{R}^3$, $i = 1, ..., n$, available from on-board sensors such as accelerometers, magnetometers, or CCD cameras expressed in the body frame B, given known constant reference vectors $r_i \in \mathbb{R}^3$ in the inertial frame I. The relationship between vectors b_i and r_i is given by

$$b_i = R^T r_i. \tag{3.1}$$

Without loss of generality, the reference vectors r_i as well as its measurement in the body frame b_i are normalized, i.e., $\|r_i\| = \|b_i\| = 1$.

The following assumptions are made:

A1: There exist at least two vectors, say r_1 and r_2, that are non-coplanar.

A2: The angular velocity is bounded, i.e., $\|\omega\| \leq \omega_{max}$, with $\omega_{max} > 0$ unknown.

A3: The inertia matrix M and the torque τ are known.

3.2 Angular velocity observer

3.2.1 Analytical form of the observer

From the dynamics of a rigid body

$$\sum_p : \quad M\dot{\omega} = S(M\omega)\omega + \tau \tag{3.2}$$

We propose the following angular velocity observer in its analytical form

$$\sum_{o}: \quad M\dot{\hat{\omega}} = S(M\hat{\omega})\hat{\omega} + K_f(\omega - \hat{\omega}) + \tau \tag{3.3}$$

where $\hat{\omega}$ represents the estimated angular velocity and ω the true angular velocity. K_f is the gain matrix to be designed. Observe that observer (3.3) consists of the attitude dynamics plus a correction term. Note that the proposal of (3.3) is for convergence analysis purposes and therefore is not directly implementable, as it depends on ω. Its implementation will be given later.

The objectives in the design of the observer are as follows:

1. Designing the gain matrix K_f such that the estimated angular velocity $\hat{\omega}$ converges to true angular velocity ω asymptotically, i.e, $\hat{\omega} - \omega \to 0$ as $t \to \infty$.

2. To eliminate ω so that the proposed observer can be implementable using only the vector measurements v_i, for $i = 1,\ 2,\ ...,\ n$.

3.2.2 Useful results

The following results are needed to construct the mathematical framework of the angular velocity observer.

Reduced kinematics.

Let $b_i \in \mathbb{R}^3$, $i = 1, ..., n$ be the vector measurements in body frame B and their corresponding inertial reference vectors $r_i \in \mathbb{R}^3$, $i = 1, ..., n$.

Lemma 3.2.1 Reduced Attitude kinematics. *Consider b_i and r_i given in (3.1). Then, the reduced attitude kinematics is given by*

$$\dot{b}_i = S(\omega)b_i. \tag{3.4}$$

Proof: Differentiating directly b_i in (3.1) and using (2.7) leads to

$$\dot{b}_i = \dot{R}^T r_i = S^T(\omega)R^T r_i = -S(\omega)R^T r_i = -S(\omega)b_i = S(b_i)\omega$$

where we have used Property $a1$ and Property $a2$ of Lemma 2.2.1 of a skew-matrix S.

Filtered vector measurements.

Let b_{if} be the filtered vector measurement of b_i, and obeys

$$\dot{b}_{if} = \gamma_f(b_i - b_{if}), \quad b_{if}(0) = b_i(0), \tag{3.5}$$

with $\gamma_f > 0$ being the filter gain. The following technical lemma, proved in [16], establishes the intuitive fact that the difference between the input and output of the filter can be arbitrarily small with a sufficiently large filter gain.

Lemma 3.2.2 *$\forall \epsilon_f > 0$, there exists a $\overline{\gamma}_f := \bar{\omega}_{max}/\epsilon_f$, being $\bar{\omega}_{max} < \infty$ the bandwidth of the system, such that $\|b_i - b_{if}\| < \epsilon_f$ for all $t \geq 0$, provided that $\gamma_f > \overline{\gamma}_f$.*

Gain matrix.

Let the matrices $K,\ K_f \in \mathbb{R}^{3\times 3}$ be defined as

$$K := \sum_{i=1}^{n} S^T(b_i)\Lambda_i S(b_i) \tag{3.6}$$

$$K_f := \sum_{i=1}^{n} S^T(b_{if})\Lambda_i S(b_i), \tag{3.7}$$

where $\Lambda_i \in \mathbb{R}^{3\times 3}$ is a symmetric positive definite gain matrix

The following lemmas gives some properties of these matrices, useful in the convergence analysis in the sequel.

Lemma 3.2.3 *Under Assumption A1, the following statements hold*

i. *K is symmetric matrix positive definite $K = K^T > 0$.*

ii. $K\omega = \sum_{i=1}^{n} S^T(b_i)\Lambda_i S(b_i)\omega = \sum_{i=1}^{n} S^T(b_i)\Lambda_i \dot{b}_i$

iii. $K_f\omega = \sum_{i=1}^{n} S^T(b_{if})\Lambda_i S(b_i)\omega = \sum_{i=1}^{n} S^T(b_{if})\Lambda_i \dot{b}_i$

iv. $K_f = \sum_{i=1}^{n} S^T(b_{if})\Lambda_i S(b_i) = \sum_{i=1}^{n} S^T(b_i)\Lambda_i S(b_i) + \sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i) = K + \sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i)$. *Equivalently,* $K_f - K = \sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i)$.

v. $\frac{d}{dt}\sum_{i=1}^{n} S^T(b_{if})\Lambda_i b_i = K_f\omega + \sum_{i=1}^{n} S^T(\dot{b}_{if})\Lambda_i b_i = K_f\omega - \gamma_f \sum_{i=1}^{n} S^T(\Lambda_i b_i)(b_i - b_{if})$

Proof i. By definition, a symmetric matrix K, with real entries, is positive definite if and only if the real number $y^T K y$ is positive for all non-zero $y \in \mathbb{R}^3$. Consider $\forall y \in \mathbb{R}^3$

$$\begin{aligned} y^T K y =& \sum_{i=1}^{n} y^T S^T(b_i)\Lambda_i S(b_i) y \\ =& \sum_{i=1}^{n} (S(b_i)y)^T \Lambda_i (S(b_i)y) \\ =& \sum_{i=1}^{n} x_i^T \Lambda_i x_i, \quad x_i = S(b_i)y \\ \geq& x_1^T\Lambda_1 x_1 + x_2^T\Lambda_2 x_2 + ... + x_n^T\Lambda_n x_n > 0. \end{aligned}$$

since every $\Lambda_i \in \mathbb{R}^{3\times 3}$ is a symmetric positive-definite matrix, this means that each term in the above sum is greater than zero. This proves item i).

Proof ii. Let's start by multiplying the matrix K by ω, we have that:

$$\begin{aligned} K\omega &= \sum_{i=1}^{n} S^T(b_i)\Lambda_i S(b_i)\omega, \quad \text{by definition of K, 3.6} \\ &= \sum_{i=1}^{n} S^T(b_i)\Lambda_i \dot{b}_i \quad \text{using that } \dot{b}_i = S(\omega)b_i \text{ 3.4} \end{aligned}$$

which ends the proof.

Proof iii. Similarly, we now multiply the matrix K_f by ω, this yields:

$$\begin{aligned} K_f\omega &= \sum_{i=1}^{n} S^T(b_{if})\Lambda_i S(b_i)\omega, \quad \text{by definition of } K_f, \text{ 3.7} \\ &= \sum_{i=1}^{n} S^T(b_{if})\Lambda_i \dot{b}_i \quad \text{using that } \dot{b}_i = S(\omega)b_i \text{ 3.4} \end{aligned}$$

and statement *iii* holds.

Proof iv. Starting by definition of the K_f matrix, we have that:

$$\begin{aligned} K_f &= \sum_{i=1}^{n} S^T(b_{if})\Lambda_i S(b_i) \quad \text{by definition of } K_f \\ &= \sum_{i=1}^{n} S^T(b_{if} + b_i - b_i)\Lambda_i S(b_i) \quad \text{adding and subtracting } b_i \\ &= \sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i) + \underbrace{\sum_{i=1}^{n} S^T(b_i)\Lambda_i S(b_i)}_{K} \quad \text{by distributing terms} \\ &= K + \sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i) \end{aligned}$$

This is the end of the proof. Furthermore, since we just have proven that

$$K_f = K + \sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i)$$

this implies that:

$$K_f - K = \sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i)$$

Proof v. Differentiating with respect to time, we have:

$$\begin{aligned}
\frac{d}{dt}\sum_{i=1}^{n} S^T(b_{if})\Lambda_i b_i &= \sum_{i=1}^{n} S^T(\dot{b_{if}})\Lambda_i b_i + \sum_{i=1}^{n} S^T(b_{if})\frac{d}{dt}\left(\Lambda_i b_i\right) \\
&= \sum_{i=1}^{n} S^T(\dot{b_{if}})\Lambda_i b_i + \sum_{i=1}^{n} S^T(b_{if})\left(\Lambda_i \dot{b}_i + \cancelto{0}{\dot{\Lambda}_i b_i}\right) \quad \text{since } \Lambda \text{ is constant over time} \\
&= \sum_{i=1}^{n} S^T(\dot{b_{if}})\Lambda_i b_i + \sum_{i=1}^{n} S^T(b_{if})\Lambda_i \dot{b}_i \\
&= \sum_{i=1}^{n} S^T(\dot{b_{if}})\Lambda_i b_i + \underbrace{\sum_{i=1}^{n} S^T(b_{if})\Lambda_i \dot{b}_i}_{K_f\omega} \quad \text{by statement } iii \\
&= K_f\omega + \sum_{i=1}^{n} S^T(\dot{b_{if}})\Lambda_i b_i \\
&= K_f\omega + \gamma_f \sum_{i=1}^{n} S^T(bi - b_{if})\Lambda_i b_i \quad \text{using that } \dot{b}_{if} = \gamma_f(b_i - b_{if}) \ (3.5) \\
&= K_f\omega - \gamma_f \sum_{i=1}^{n} S^T(\Lambda_i b_i)(bi - b_{if}) \quad \text{using that } S(x)y = -S(y)x
\end{aligned}$$

this completes the proof.

3.3 Stability analysis

We prove in this section the convergence of the proposed angular velocity observer. The results are summarized in the following theorem.

Theorem 3.3.1 Global, exponential convergence of the angular velocity observer: *The observer defined by* (3.15) *and* (3.16) *achieves* $\hat{\omega} \to \omega$ *exponentially from any initial condition.*

Proof. From the analytical form of the observer (3.3) it follows that

$$\begin{aligned}
M\dot{\hat{\omega}} &= S(M\hat{\omega})\hat{\omega} + K_f(\omega - \hat{\omega}) + \tau \\
&= S(M\omega)\hat{\omega} + S(M(\hat{\omega} - \omega))\hat{\omega} + K_f(\omega - \hat{\omega}) + \tau \\
&= S(M\omega)\hat{\omega} - S(\hat{\omega})M(\hat{\omega} - \omega) + K_f(\omega - \hat{\omega}) + \tau \quad \text{using a2. from Lemma 2.2.1} \\
&= S(M\omega)\hat{\omega} + S(\hat{\omega})M(\omega - \hat{\omega}) + K_f(\omega - \hat{\omega}) + \tau \\
&= S(M\omega)\hat{\omega} + \Big(K_f + S(\hat{\omega})M\Big)(\omega - \hat{\omega}) + \tau.
\end{aligned}$$

Let the angular velocity error be defined as

$$\tilde{\omega} = \hat{\omega} - \omega \tag{3.8}$$

Thus, the error dynamics is expressed as

$$M\dot{\tilde{\omega}} = S(M\omega)\tilde{\omega} - (K_f + S(\hat{\omega})M)\tilde{\omega}. \tag{3.9}$$

Consider now the Lyapunov function candidate

$$V_\omega(\tilde{\omega}) = \frac{1}{2}\tilde{\omega}^T M\tilde{\omega} \tag{3.10}$$

which is positive definite with respect to $\tilde{\omega} = 0$ and radially unbounded. Differentiating $V_\omega(\tilde{\omega})$ with respect to time and evaluating it along the error dynamics (3.9) yield

$$\begin{aligned}
\dot{V}_\omega(\tilde{\omega}) &= \tilde{\omega}^T M\dot{\tilde{\omega}} \\
&= \tilde{\omega}^T\Big(S(M\omega)\tilde{\omega} - (K_f + S(\hat{\omega})M)\tilde{\omega}\Big) \\
&= -\tilde{\omega}^T\Big(K + (K_f - K)\Big)\tilde{\omega} - \tilde{\omega}^T S(\hat{\omega})M\tilde{\omega} \\
&= -\tilde{\omega}^T K\tilde{\omega} - \tilde{\omega}^T(K_f - K)\tilde{\omega} - \tilde{\omega}^T S(\hat{\omega})M\tilde{\omega} \\
&= -\tilde{\omega}^T K\tilde{\omega} - \tilde{\omega}^T(K_f - K)\tilde{\omega} - \tilde{\omega}^T S(\hat{\omega} - \omega + \omega)M\tilde{\omega} \\
&= -\tilde{\omega}^T K\tilde{\omega} - \tilde{\omega}^T(K_f - K)\tilde{\omega} - \tilde{\omega}^T S(\tilde{\omega} + \omega)M\tilde{\omega}
\end{aligned} \tag{3.11}$$

To get a bound on the second and the third r.h.t. of (3.11) consider

$$\begin{aligned}
\|K_f - K\| &= \|\sum_{i=1}^{n} S^T(b_{if} - b_i)\Lambda_i S(b_i)\| \\
&\leq \sum_{i=1}^{n}\|S^T(b_{if} - b_i)\|\|\Lambda_i\|\|S(b_i)\| \\
&\leq \sum_{i=1}^{n}\lambda_{max}(\Lambda_i)\|b_{if} - b_i\| \leq \epsilon_f\bar{\lambda}.
\end{aligned}$$

where $max_i\{\lambda_{max}(\Lambda_i)\} \leq \bar{\lambda}$, and

$$\begin{aligned}
\|\tilde{\omega}^T S(\tilde{\omega} + \omega)M\tilde{\omega}\| &= \|\tilde{\omega}^T S(\omega)M\tilde{\omega}\| \\
&\leq \lambda_{max}(\Lambda_i)\|\omega\|\|\tilde{\omega}\|^2 \\
&\leq \lambda_{max}(\Lambda_i)\omega_{max}\|\tilde{\omega}\|^2
\end{aligned}$$

where $\|\omega\| \leq \omega_{max}$. Therefore,

$$\begin{aligned}
\dot{V}_\omega(\tilde{\omega}) &\leq -\underbrace{\big(\lambda_{min}(K) - \epsilon_f\bar{\lambda} - \lambda_{max}(M)\omega_{max}\big)}_{\lambda_\omega}\|\tilde{\omega}\|^2 \\
&= 2\frac{\lambda_\omega}{\lambda_{max}(M)}\big(-\frac{1}{2}\lambda_{max}(M)\|\tilde{\omega}\|^2\big) \\
&\leq -2\frac{\lambda_\omega}{\lambda_{max}(M)}V_\omega,
\end{aligned} \tag{3.12}$$

where Assumption 2 and the facts in Lemma 3.2.2 are used, $\lambda_\omega := \lambda_{min}(K) - \epsilon_f\bar{\lambda} - \lambda_{max}(M)\omega_{max}$, and $\bar{\lambda} \geq max_i\{\lambda_{max}(\Lambda_i)\}$. Therefore, the equilibrium $\tilde{\omega} = 0$ is globally exponentially stable (Theorem 4.10, [24]). This proves the global exponential convergence of the observer.

3.4 Implementation of the angular velocity observer

To eliminate the unknown angular velocity from the analysis form of the observer (3.3), we rewrite it as

$$M\dot{\hat{\omega}} = S(M\hat{\omega})\hat{\omega} + K_f\omega - K_f\hat{\omega} + \tau \tag{3.13}$$

By Lemma 3.2.3 v, we obtain

$$K_f\omega = \frac{d}{dt}\sum_{i=1}^{n} S^T(b_{if})\Lambda_i b_i + \gamma_f \sum_{i=1}^{n} S^T(\Lambda_i b_i)(b_i - b_{if}) \tag{3.14}$$

Substituting $K_f\omega$ from (3.14) in (3.13), yields

$$M\dot{\hat{\omega}} = S(M\hat{\omega})\hat{\omega} + \frac{d}{dt}\sum_{i=1}^{n} S^T(b_{if})\Lambda_i b_i + \gamma_f \sum_{i=1}^{n} S^T(\Lambda_i b_i)(b_i - b_{if}) - K_f\hat{\omega} + \tau$$

This gives that

$$\frac{d}{dt}\left(\underbrace{M\hat{\omega} - \sum_{i=1}^{n} S^T(b_{if})\Lambda_i b_i}_{\bar{\omega}}\right) = S(M\hat{\omega})\hat{\omega} + \gamma_f \sum_{i=1}^{n} S^T(\Lambda_i b_i)(b_i - b_{if}) - K_f\hat{\omega} + \tau$$

Defining

$$\bar{\omega} = M\hat{\omega} - \sum_{i=1}^{n} S^T(b_{if})\Lambda_i b_i,$$

the angular velocity observer is then implemented in the following form

$$\dot{\bar{\omega}} = S(M\hat{\omega})\hat{\omega} + \gamma_f \sum_{i=1}^{n} S^T(\Lambda_i b_i)(b_i - b_{if}) - K_f\hat{\omega} + \tau \tag{3.15}$$

$$\hat{\omega} = M^{-1}\left(\bar{\omega} + \sum_{i=1}^{n} S^T(b_{if})\Lambda_i b_i\right), \tag{3.16}$$

together with the filter (3.5) and the initial condition $\bar{\omega}(0) = M\hat{\omega}(0) - \sum_{i=1}^{n} S^T(b_{if}(0))\Lambda_i b_i(0)$.

Remark 3.4.1 *In contrast to the angular velocity proposed in [13, 16, 40, 44], instead of the attitude measurements, the angular velocity observer (3.15)-(3.16) uses solely the vector*

measurements b_i, $i = 1, 2, ..., n$, similarly to [9, 46]. This feature provides several advantages than using attitude measurements such as it avoids employing an attitude representation and enables to achieve the global exponential convergence. On the other hand, since this observer is interdependent of the rigid body attitude, the estimated angular velocity together with the same vector measurements can be used to estimate the rigid body attitude.

Chapter 4

Attitude observer design

This chapter an attitude observer that uses the same vector measurements as the angular velocity observer in the previous section and the estimated angular velocity is designed. Both angular velocity and attitude observer constitute the rotational state observer.

4.1 Problem statement

Consider a rigid body whose kinematics and dynamics are described by (2.7) and (2.11), respectively. The problem addressed in this chapter is to design an observer to estimate the attitude in terms of a rotation matrix R using, as in the previous chapter, the vector measurements $b_i \in \mathbb{R}^3$, $i = 1, ..., n$, available from on-board sensors such as accelerometers, magnetometers, sun sensors, or CCD cameras expressed in the body frame B, given known constant reference vectors $r_i \in \mathbb{R}^3$ in the inertial frame I.

Recall that the relationship between vectors b_i and r_i is given by $b_i = R^T r_i$ (3.1) and without loss of generality, the reference vectors r_i as well as its measurement in the body frame b_i are normalized, i.e., $\|r_i\| = \|b_i\| = 1$. Also the assumption A1 in Section 3.1 is made.

4.2 Attitude observer design

We use the ECF for attitude estimation. Instead of using the angular velocity from the gyros, we use the angular velocity estimate performed by the angular velocity observer (3.15)-(3.16). The attitude observer is

$$\dot{\hat{R}} = \hat{R} S(\hat{\omega} + k_p \sigma_R) \tag{4.1}$$

$$\sigma_R = \sum_{i=1}^{n} k_i S(b_i) \hat{b}_i, \tag{4.2}$$

where $\hat{R}$ is the attitude estimate, $\hat{b}_i = \hat{R}^T r_i$ is the estimate of b_i, $k_p > 0$ is a scalar gain, k_i are positive constant gains that represent the reliability of the sensors. If each sensor is equally trusted, then their value is equal to 1 in each case; b_i, $i = 1, 2, ..., n$ are the sensor measurements in the body frame defined in (3.1) satisfying Assumption A1.

The proposed angular observer with the attitude ECF constitutes an overall nonlinear observer with cascaded structure, illustrated in Fig. 4.1.

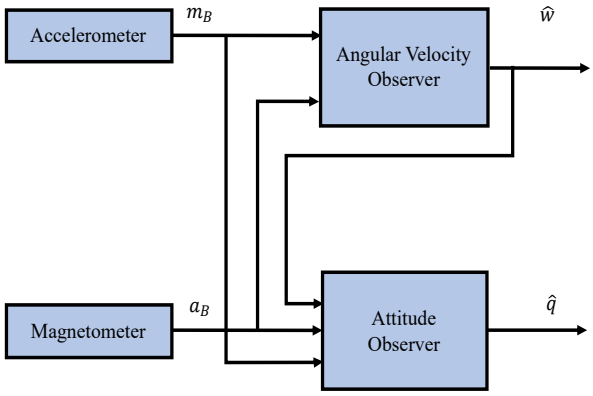


Figure 4.1: Illustration of the proposed cascaded structure of the observer when the vector measurements are obtained from an IMU.

4.2.1 Useful results for the convergence analysis of the attitude observer

The following results, summarized in lemmas, will be used for the convergence analysis of the attitude observer in cascade with the angular velocity observer proposed in the previous chapter.

Attitude estimation error.

Let the attitude estimation error be defined as $\tilde{R} = \hat{R}^T R$, and error e_R as

$$e_R = \frac{1}{2}\sum_{i=1}^{n} k_i \|b_i - \hat{b}_i\|^2. \tag{4.3}$$

Lemma 4.2.1 (Dynamics of the attitude estimation error.) *The dynamics of the attitude estimation error is*

$$\dot{\tilde{R}} = [\tilde{R}, S(\omega)] - S(\tilde{\omega})\tilde{R} - S(k_p \sigma_R)\tilde{R}. \tag{4.4}$$

Proof. Taking the time derivative of $\tilde{R}$ yields

$$\dot{\tilde{R}} = \dot{\hat{R}}^T R + \hat{R}^T \dot{R}$$

Substituting $\dot{R} = RS(\omega)$ in (2.7) and $\dot{\hat{R}} = \hat{R}S(\hat{\omega} + k_p\sigma_R)$ in (4.1) yield

$$\begin{aligned}
\dot{\tilde{R}} &= \dot{\hat{R}}^T R + \hat{R}^T \dot{R} \\
&= \left[\hat{R}S(\hat{\omega} + k_p\sigma_R)\right]^T R + \hat{R}^T RS(\omega) \\
&= S^T(\hat{\omega} + k_p\sigma_R)\hat{R}^T R + \hat{R}^T RS(\omega) \\
&= -S(\hat{\omega} + k_p\sigma_R)\hat{R}^T R + \hat{R}^T RS(\omega) \quad \text{using a1. Lemma 2.2.1 } S^T = -S \\
&= -S(\hat{\omega} + k_p\sigma_R)\tilde{R} + \tilde{R}S(\omega) \quad \text{by definition, } \tilde{R} = \hat{R}^T R \\
&= -S(\omega + \tilde{\omega} + k_p\sigma_R)\tilde{R} + \tilde{R}S(\omega) \quad \text{recalling, } \tilde{\omega} = \hat{\omega} - \omega \\
&= -S(\omega)\tilde{R} - S(\tilde{\omega})\tilde{R} - S(k_p\sigma_R)\tilde{R} + \tilde{R}S(\omega) \\
&= \tilde{R}S(\omega) - S(\omega)\tilde{R} - S(\tilde{\omega})\tilde{R} - S(k_p\sigma_R)\tilde{R} \\
&= [\tilde{R}, S(\omega)] - S(\tilde{\omega})\tilde{R} - S(k_p\sigma_R)\tilde{R}
\end{aligned}$$

where $[A, B] = AB - BA$ is Lie-bracket.

Correction term σ_R.

Some facts on the correction term σ_R are stated in the following lemma.

Lemma 4.2.2 (Facts on the correction term σ_R.)

- *The correction term (4.2) can be expressed as $\sigma_R = \sum_{i=1}^{n} k_i S(b_i)\hat{b}_i = P_a^{\vee}(\tilde{R}\Gamma_b)$, where the operation $(\cdot)^{\vee}$ is the inverse operation of $S(u), u \in \mathbb{R}^3$, i.e., $S^{\vee}(u) = u$, $\Gamma_b = R^T\Gamma_r R$ with $\Gamma_r := \sum_{i=1}^{n} k_i r_i r_i^T$. Both Γ_r and Γ_b are positive definite under Assumption A1.*
- *$\sigma_R = 0$ implies that $\tilde{R} = I_3$ or $tr(\tilde{R}) = -1$.*

Proof. i) By definition, the operator P_a acts on a matrix $\mathrm{H} \in \mathbb{R}^{3\times 3}$ $P_a(H) = \frac{1}{2}(H - H^T)$.

$$
\begin{aligned}
P_a(\tilde{R}\Gamma_b) &= \frac{1}{2}\left(\tilde{R}\Gamma_b - (\tilde{R}\Gamma_b)^T\right) \\
&= \frac{1}{2}\left(\tilde{R}\Gamma_b - \Gamma_b^T\tilde{R}^T\right) \\
&= \frac{1}{2}\left(\tilde{R}\Gamma_b - \Gamma_b\tilde{R}^T\right) \quad \text{since } \Gamma_b^T = \Gamma_b \\
&= \frac{1}{2}\left(\tilde{R}R^T\Gamma_r R - R^T\Gamma_r R\tilde{R}^T\right) \quad \text{using definition, } \Gamma_b = R^T\Gamma_r R \\
&= \frac{1}{2}\left(\hat{R}^T R R^T\Gamma_r R - R^T\Gamma_r R(\hat{R}^T R)^T\right) \quad \text{using, } \tilde{R} = \hat{R}^T R \\
&= \frac{1}{2}\left(\hat{R}^T\Gamma_r R - R^T\Gamma_r R R^T\hat{R}\right) \\
&= \frac{1}{2}\left(\hat{R}^T\Gamma_r R - R^T\Gamma_r\hat{R}\right) \\
&= \frac{1}{2}\left[\hat{R}^T\sum_{i=1}^{n} k_i r_i r_i^T R - R^T\sum_{i=1}^{n} k_i r_i r_i^T\hat{R}\right] \quad \text{using definition, } \Gamma_r := \sum_{i=1}^{n} k_i r_i r_i^T \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i\left[\hat{R}^T r_i r_i^T R - R^T r_i r_i^T\hat{R}\right] \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i\left[\hat{b}_i r_i^T R - R^T r_i\hat{b}_i^T\right] \quad \text{using, } \hat{b}_i = \hat{R}^T r_i \quad \text{and } \hat{b}_i^T = r_i^T\hat{R} \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i\left[\hat{b}_i b_i^T - b_i\hat{b}_i^T\right] \quad \text{using, } b_i^T = r_i^T R \quad \text{and } b_i = R^T r_i \\
&= \sum_{i=1}^{n}\frac{k_i}{2}\left[\hat{b}_i b_i^T - b_i\hat{b}_i^T\right]
\end{aligned}
$$

On the other hand, $\sigma_R = \sum_{i=1}^{n} k_i S(b_i)\hat{b}_i$. This implies that:

$$
\begin{aligned}
S(\sigma_R) &= S\left(\sum_{i=1}^{n} k_i S(b_i)\hat{b}_i\right) = \sum_{i=1}^{n} k_i S\left(S(b_i)\hat{b}_i\right) \\
&= \sum_{i=1}^{n} k_i S\left(b_i \times \hat{b}_i\right) \quad \text{using, } S(u)v = u \times v \\
&= \sum_{i=1}^{n}\frac{k_i}{2}\left(\hat{b}_i b_i^T - b_i\hat{b}_i^T\right) \quad \text{using property, } S(x\times y) = \frac{1}{2}(yx^T - xy^T)
\end{aligned}
$$

Hence,

$$
P_a(\tilde{R}\Gamma_b) = \sum_{i=1}^{n}\frac{k_i}{2}\left[\hat{b}_i b_i^T - b_i\hat{b}_i^T\right] = S(\sigma_R) \tag{4.5}
$$

by applying the operator $(\cdot)^\vee$ in both sides of equation (4.5), yields:

$$
P_a^\vee(\tilde{R}\Gamma_b) = S^\vee(\sigma_R) = \sigma_R = \sum_{i=1}^{n} k_i S(b_i)\hat{b}_i \tag{4.6}
$$

The proof of the item i) is now complete.

ii) Note that if $\sigma_R = 0$, then $S(\sigma_R) = 0$.

$$S(\sigma) = P_a(\tilde{R}\Gamma_b) = \frac{1}{2}\left(\tilde{R}\Gamma_b - (\tilde{R}\Gamma_b)^T\right) = \frac{1}{2}\left(\tilde{R}\Gamma_b - \Gamma_b^T\tilde{R}^T\right) = \frac{1}{2}\left(\tilde{R}\Gamma_b - \Gamma_b\tilde{R}^T\right) = 0.$$

$\Longrightarrow$

$$\tilde{R}\Gamma_b = \Gamma_b\tilde{R}^T \tag{4.7}$$

Given matrix $\tilde{R}$, the eigenvalues are the solutions to the characteristic equation,

$$det(R - \lambda I_3) = 0. \tag{4.8}$$

The Euler's theorem states that: For any $R \in SO(3)$, there is a non-zero vector $v \in \mathbb{R}^3$ satisfying $Rv = \lambda v$. This theorem implies that the attitude of a body can be specified in terms of a rotation by some angle about some fixed axis. Since $\tilde{R}$ is a real orthogonal matrix, the eigenvalue equation for $\tilde{R}$ and its complex conjugate transpose are given by:

$$\tilde{R}v = \lambda v, \quad v^{*T}\tilde{R}^T = \lambda^* v^{*T}$$

Hence, multiplying these two equations together yields,

$$\lambda^*\lambda v^{*T}v = v^{*T}\tilde{R}^T\tilde{R}v = v^{*T}v \tag{4.9}$$

since an orthogonal matrix satisfies $\tilde{R}R = I_3$. Since eigenvectors must be nonzero, it follows that $v^{*T}v \neq 0$. Hence, equation (4.9) yields $\|\lambda\| = 1$. Thus, the eigenvalues of a real orthogonal matrix must be complex numbers of unit modulus. That is, $\lambda = e^{i\alpha}$ for some α in the interval $0 \leq \alpha < 2\pi$. Note that,

$$\tilde{R}^T(I_3 - \tilde{R}) = \tilde{R}^T - I_3 = -(I_3 - \tilde{R})^T$$

Taking the determinant of both sides of this equation, it follows that:

$$det(\tilde{R})det(I_3 - \tilde{R}) = (-1)^3 det(I_3 - \tilde{R}), \tag{4.10}$$

since for the 3×3 identity matrix, $det(-I_3) = (-1)^3$. We have $det(\tilde{R}) = 1$ and $(-1)^3 = -1$. Hence, eq. (4.10) yields

$$det(I_3 - \tilde{R}) = 0,$$

Comparing with eq. (4.8), we can conclude that $\lambda = 1$ is an eigenvalue of $\tilde{R}$. Since $det(\tilde{R})$ is the product of its three eigenvalues and each eigenvalue of $\tilde{R}$ is a complex number of unit modulus, it follows that the eigenvalues of any proper 3×3 orthogonal matrix must be 1, $e^{i\theta}$ and $e^{-i\theta}$ for some value of θ that lies in the interval $0 \leq \theta \leq \pi$. All three eigenvalues of $\tilde{R}$ are real since complex eigenvalues must come in complex conjugate pairs. Hence, it follows that:

$$eigen(\tilde{R}) = (1, e^{i\theta}, e^{-i\theta}) = (1, cos(\theta) + isin\theta, cos(\theta) - isin\theta)$$

or $\theta = 0$ or $\theta = \pm\pi$. When $\theta = 0$, the eigenvalues of $\tilde{R}$ are: $eigen(\tilde{R}) = (1, 1, 1)$, using the angle- axis coordinates representation of $\tilde{R} \in SO(3)$:

$$cos(\theta) = \frac{1}{2}(tr(\tilde{R} - 1)) \tag{4.11}$$

in $\theta = 0$,

$$cos(0) = 1 = \frac{1}{2}(tr(\tilde{R} - 1))$$

$\implies tr(\tilde{R} - 1) = 2$ or equivalently $3 = tr(\tilde{R})$, This means that $\tilde{R} = I_3$. Now, evaluating eq. (4.11) in $\theta = \pm\pi$:

$$cos(\pm\pi) = -1 = \frac{1}{2}(tr(\tilde{R} - 1))$$

$\implies tr(\tilde{R}-1) = -2$ or equivalently $tr(\tilde{R}) = -2+1 = -1$. This ends the proof of the item ii).

More properties of the attitude estimation error $\tilde{R}$ aand e_R.

The attitude estimation error $\tilde{R}$ and e_R defined in (4.3) have the following properties:

Lemma 4.2.3 (More properties of the error e_R.) *The error e_R defined in (4.3) can be expressed as $e_R = \frac{1}{2}\sum_{i=1}^{n} k_i \|b_i - \hat{b}_i\|^2 = \sum_{i=1}^{n} k_i - tr(\tilde{R}\Gamma_b)$.*

Proof.

$$\begin{aligned}
e_R &= \frac{1}{2}\sum_{i=1}^{n} k_i \|b_i - \hat{b}_i\|^2 \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i (b_i - \hat{b}_i)^T (b_i - \hat{b}_i) \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i (b_i^T - \hat{b}_i^T)(b_i - \hat{b}_i) \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i (b_i^T b_i - b_i^T \hat{b}_i - \hat{b}_i^T b_i + \hat{b}_i^T \hat{b}_i) \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i (\|b_i\|^2 + \|\hat{b}_i\|^2 - b_i^T \hat{b}_i - \hat{b}_i^T b_i) \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i (1+1) - \frac{1}{2}\sum_{i=1}^{n} k_i (b_i^T \hat{b}_i + \hat{b}_i^T b_i) \\
&= \frac{1}{2}\sum_{i=1}^{n} k_i (2) - \sum_{i=1}^{n} \frac{k_i}{2} (b_i^T \hat{b}_i + \hat{b}_i^T b_i) \\
&= \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} \frac{k_i}{2} (b_i^T \hat{b}_i) - \sum_{i=1}^{n} \frac{k_i}{2} (\hat{b}_i^T b_i) \\
&= \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} \frac{k_i}{2} (tr(b_i \hat{b}_i^T)) - \sum_{i=1}^{n} \frac{k_i}{2} (tr(\hat{b}_i b_i^T))
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} \frac{k_i}{2}(tr(b_i\hat{b}_i^T)) - \sum_{i=1}^{n} \frac{k_i}{2}(tr(b_i^T\hat{b}_i)) \\
&= \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} \frac{k_i}{2}(tr(b_i\hat{b}_i^T)) - \sum_{i=1}^{n} \frac{k_i}{2}(tr(b_i\hat{b}_i^T)) \\
&= \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} k_i(tr(b_i\hat{b}_i^T))
\end{aligned}
$$

Therefore,

$$
e_R = \sum_{i=1}^{n} k_i - \sum_{i=1}^{n} k_i(tr(b_i\hat{b}_i^T)) = \sum_{i=1}^{n} k_i - tr(\tilde{R}\Gamma_b).
$$

where $tr(\tilde{R}\Gamma_b) = tr(\tilde{R}R^T\Gamma_r R) = tr(\hat{R}^T R R^T \Gamma_r R) = tr(\hat{R}^T\Gamma_r R) = tr\left(\hat{R}^T \sum_{i=1}^{n} k_i r_i r_i^T R\right) = \sum_{i=1}^{n} k_i tr(\hat{R}^T r_i r_i^T R) = \sum_{i=1}^{n} k_i tr(\hat{b_i} b_i^T) = \sum_{i=1}^{n} k_i \hat{b_i}^T b_i$. This ends the proof.

Lemma 4.2.4 (More properties of the error $\tilde{R}$.) $\frac{d}{dt}\tilde{R}\Gamma_b = [\tilde{R}\Gamma_b, S(\omega)] - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b$.

Proof.

$$
\begin{aligned}
\frac{d}{dt}\tilde{R}\Gamma_b &= \dot{\tilde{R}}\Gamma_b + \tilde{R}\dot{\Gamma}_b \\
&= [\tilde{R}, S(\omega)]\Gamma_b - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b + \tilde{R}\dot{\Gamma}_b
\end{aligned}
$$

but

$$
\begin{aligned}
\dot{\Gamma}_b &= R^T\Gamma_r\dot{R} + \dot{R}^T\Gamma_r R \\
&= R^T\Gamma_r(RS(\omega)) + (RS(\omega))^T\Gamma_r R \\
&= R^T\Gamma_r RS(\omega) + S^T(\omega)R^T\Gamma_r R \\
&= R^T\Gamma_r RS(\omega) - S(\omega)R^T\Gamma_r R \\
&= \Gamma_b S(\omega) - S(\omega)\Gamma_b = [\Gamma_b, S(\omega)]
\end{aligned}
$$

by substituting $\dot{\Gamma}_b$ in the above expression, we have:

$$
\begin{aligned}
\frac{d}{dt}\tilde{R}\Gamma_b &= [\tilde{R}, S(\omega)]\Gamma_b - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b + \tilde{R}[\Gamma_b, S(\omega)] \\
&= (\tilde{R}S(\omega) - S(\omega)\tilde{R})\Gamma_b + \tilde{R}(\Gamma_b S(\omega) - S(\omega)\Gamma_b) - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b \\
&= \tilde{R}S(\omega)\Gamma_b - S(\omega)\tilde{R}\Gamma_b + \tilde{R}\Gamma_b S(\omega) - \tilde{R}S(\omega)\Gamma_b - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b \\
&= \cancel{\tilde{R}S(\omega)\Gamma_b} - S(\omega)\tilde{R}\Gamma_b + \tilde{R}\Gamma_b S(\omega) - \cancel{\tilde{R}S(\omega)\Gamma_b} - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b \\
&= \tilde{R}\Gamma_b S(\omega) - S(\omega)\tilde{R}\Gamma_b - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b \\
&= [\tilde{R}\Gamma_b, S(\omega)] - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b
\end{aligned}
$$

which completes the proof of the Lemma.

Lemma 4.2.5 $\dot{e}_R = -tr\Big(\frac{d}{dt}\tilde{R}\Gamma_b\Big) = -2\tilde{\omega}^T\sigma_R - 2k_p\|\sigma_R\|^2$.

Proof. First, we can recall the definition of $\dot{e}_R$ Lemma 4.2.3.

$$e_R = \sum_{i=1}^{n} k_i - tr(\tilde{R}\Gamma_b)$$

Differentiating error with respect to time,

$$\begin{aligned}
\dot{e_R} &= -tr\left(\frac{d}{dt}\tilde{R}\Gamma_b\right) \\
&= -tr\left([\tilde{R}\Gamma_b, S(\omega)] - S(\tilde{\omega})\tilde{R}\Gamma_b - S(k_p\sigma_R)\tilde{R}\Gamma_b\right) \quad \text{by Lemma 4.2.4} \\
&= -tr\left([\tilde{R}\Gamma_b, S(\omega)]\right) + tr\left(S(\tilde{\omega})\tilde{R}\Gamma_b\right) + tr\left(S(k_p\sigma_R)\tilde{R}\Gamma_b\right) \\
&= tr\left(S(\tilde{\omega})\tilde{R}\Gamma_b\right) + tr\left(S(k_p\sigma_R)\tilde{R}\Gamma_b\right) \quad \text{since } tr([\tilde{R}\Gamma_b, S(\omega)]) = 0 \\
&= tr\left(\tilde{R}\Gamma_b S(\tilde{\omega})\right) + tr\left(\tilde{R}\Gamma_b S(k_p\sigma_R)\right) \quad \text{since } tr(AB) = tr(BA) \\
&= tr\left(P_a(\tilde{R}\Gamma_b)S(\tilde{\omega})\right) + tr\left(P_a(\tilde{R}\Gamma_b)S(k_p\sigma_R)\right) \quad \text{property } tr(AS(x)) = tr(P_a(A)S(x)) \\
&= -\langle S(\tilde{\omega}), P_a(\tilde{R}\Gamma_b)\rangle - \langle S(k_p\sigma_R), P_a(\tilde{R}\Gamma_b)\rangle \quad \text{using } tr(S(u)A) = -\langle S(u), P_a(A)\rangle \\
&= -2\langle \omega, P_a^\vee(\tilde{R}\Gamma_b)\rangle - 2\langle k_p\sigma_R, P_a^\vee(\tilde{R}\Gamma_b)\rangle \quad \text{using } -\langle S(u), P_a(A)\rangle = -2\langle u, P_a^\vee(A)\rangle \\
&= -2\tilde{\omega}^T P_a^\vee(\tilde{R}\Gamma_b) - 2(k_p\sigma_R)^T P_a^\vee(\tilde{R}\Gamma_b) \quad \text{using } \langle u, P_a^\vee(A)\rangle = u^T P_a^\vee(A) \\
&= -2\tilde{\omega}^T\sigma_R - 2k_p\sigma_R^T\sigma_R \quad \text{using } P_a^\vee(\tilde{R}\Gamma_b) = \sigma_R \\
&= -2\tilde{\omega}^T\sigma_R - 2k_p\|\sigma_R\|^2
\end{aligned}$$

This concludes the proof of the Lemma.

4.3 Stability analysis

The convergence of the attitude observer (4.1) in cascade with the angular velocity observer (3.15) is stated in the following theorem.

Theorem 4.3.1 (Almost global, asymptotic converge of the cascaded observer.) *Consider the kinematics (2.7) and the dynamics (2.11) of a rigid body. The observer defined by (3.15)-(3.16) and (4.1) achieves $\hat{\omega} \to \omega$ and $\hat{R} \to R$ asymptotically for almost all initial conditions $(\hat{\omega}(0), \hat{R}(0)) \in \mathbb{R}^3 \times SO(3)$ except a unstable set on $SO(3)$ of zero measure, provided the observer gain k_p satisfies $k_p\lambda_\omega > 1/2$.*

Proof. Consider the error dynamics of the angular velocity observer (3.9), the attitude observer (4.4), and the following Lyapunov-like function

$$V = V_\omega + e_R = \frac{1}{2}\tilde{\omega}^T M \tilde{\omega} + \frac{1}{2}\sum_{i=1}^{n} \|b_i - \hat{b}_i\|^2. \tag{4.12}$$

By (3.11) and the facts stated in Lemma 3.2.2, the time derivative of (4.12) evaluated along the error dynamics of (3.9) and (4.4) is

$$\begin{aligned} \dot{V} &= \dot{V}_\omega + \dot{e}_R \\ &\leq -\lambda_\omega \|\tilde{\omega}\|^2 - 2\tilde{\omega}^T \sigma_R - 2k_p \|\sigma_R\|^2 \\ &= -[\|\tilde{\omega}\| \ \|\sigma_R\|] \begin{bmatrix} \lambda_\omega & 1 \\ 1 & 2k_p \end{bmatrix} \begin{bmatrix} \|\tilde{\omega}\| \\ \|\sigma_R\| \end{bmatrix} \leq 0, \end{aligned} \tag{4.13}$$

provided $2\lambda_\omega k_p > 1$. By Barbalat lemma, it follows that $\tilde{\omega} \to 0$ and $\sigma \to 0$ asymptotically. This gives that $\hat{\omega} \to \omega$ and $\hat{R} \to R$ or $tr(\tilde{R}) = -1$ asymptotically. This condition $tr(\tilde{R}) = -1$ corresponds to a set in $SO(3)$ with measure zero, where e_R reaches its local maximum. By Chateav theorem [24], this set is unstable.

Remark 1. The observer system (3.15) and (3.16) in cascade with the attitude observer (4.1) provides both angular velocity and attitude estimation using at least no-coplanar vector measurements. While the convergence of the angular velocity observer is global and exponential, due to the topological constraints in $SO(3)$ the attitude observer is only almost global asymptotic [28]. Several improvements on its convergence including the gain selection (a matrix gain instead of the scalar gain), linear acceleration estimation with a velocity sensor to relax the assumption on small linear acceleration, and using a switching strategy to achieve global exponential convergence are available.

Remark 2. The proposed observer has a cascaded structure, making the angular velocity decoupled from the attitude estimation. This facilitates the convergence of the convergence analysis of both the angular velocity and attitude observer. Moreover, the proposed observer, by relying on only vector measurements, provides a backup solution in the eventual failure of gyros.

Chapter 5

Linear velocity and position observer

In this chapter a novel nonlinear observer for estimating the translational state of a rigid body is presented. This observer was created assuming that vector measurements are available along with the GPS data information. This chapter provides a detailed development of the theoretical framework on which the observer was created, this includes an overview of previous work related to full state estimation, the theoretical proposal of the observer and its stability analysis; as well as the discussion about its main characteristics, advantages and disadvantages with respect to other proposals.

5.1 Problem statement

Consider the translational motion of a rigid body described by the equations (2.12)-(2.13). The problem addressed in this section is to design a navigation observer to estimate the translational state (position p and linear velocity v) using vector measurements, available from onboard sensors, and a GPS sensor.

From the translational dynamics of a rigid body:

$$\sum_p : \quad \dot{v} = ge_3 - \frac{f}{m}Re_3. \tag{5.1}$$

The accelerometer measurements in body frame can be rewritten as follows: $a_B = -(f/m)e_3$. Therefore, the model of the plant can be written as follows:

$$\sum_p : \quad \dot{v} = ge_3 + Ra_B \tag{5.2}$$

We propose the following linear velocity observer in its analytical form

$$\sum_o : \quad \dot{\hat{v}} = ge_3 + \hat{R}a_B - \kappa_2(\hat{p} - p) - \kappa_3(\hat{v} - v) \tag{5.3}$$

where $\hat{v}$ represents the linear velocity estimation and v the true linear velocity. Note that the observer (5.3) consists of the translational dynamics (5.2) plus a correction term. Also, note that the theoretical proposal (5.3) is not directly implementable because it depends implicitly on v. The implementable version will be written later.

The objectives in the design of the translational observer are:

1. Designing the observer such that the estimated linear velocity $\hat{v}$ converges to true linear velocity v exponentially, i.e, $\hat{v} - v \to 0$ as $t \to \infty$. Also, that the estimated position $\hat{p}$ converges to true position p exponentially, i.e, $\hat{p} - p \to 0$ as $t \to \infty$.

2. To eliminate v so that the proposed observer (5.3), can be implementable using only the accelerometer measurements v_i, for $i = 1,\ 2,\ ...,\ n$ and a GPS sensor.

5.2 Linear velocity and position observer design

Starting from the analytical form of the observer (5.3), we have:

$$\dot{\hat{v}} = ge_3 + \hat{R}a_B - \kappa_2(\hat{p} - p) - \kappa_3(\hat{v} - v)$$

We can rearrange the above equation as:

$$\dot{\hat{v}} - \kappa_3 v = ge_3 + \hat{R}a_B - \kappa_2(\hat{p} - p) - \kappa_3\hat{v}$$

by substituting (2.12), $\dot{p} = v$ into the equation above, gives:

$$\begin{aligned} ge_3 + \hat{R}a_B - \kappa_2(\hat{p} - p) - \kappa_3\hat{v} &= \dot{\hat{v}} - \kappa_3 v \\ &= \dot{\hat{v}} - \kappa_3\dot{p} \\ &= \frac{d}{dt}(\underbrace{\hat{v} - \kappa_3 p}) \\ &= \frac{d}{dt}\bar{v} \end{aligned}$$

This implies that the implementable version of the translational observer can be written as:

$$\dot{\bar{v}} = ge_3 + \hat{R}a_B - \kappa_2(\hat{p} - p) - \kappa_3\hat{v}$$

and also, that:

$$\dot{\bar{v}} = \hat{v} - \kappa_3 p$$

This establishes the theoretical basis for generating the new proposal of the observer.

Let $\hat{p},\ \hat{v} \in \mathbb{R}^3$ be the position and linear velocity estimate, respectively. The following translational state observer is proposed.

$$\dot{\hat{p}} = \hat{v} - \kappa_1(\hat{p} - p) \tag{5.4}$$

$$\dot{\bar{v}} = ge_3 + \hat{R}a_B - \kappa_2(\hat{p} - p) - \kappa_3\hat{v}, \tag{5.5}$$

$$\hat{v} = \bar{v} + \kappa_3 p, \tag{5.6}$$

with initial conditions $\hat{p}(0) = p(0)$ and $\bar{v}(0) = \hat{v}(0) - \kappa_3 p(0)$ where $\hat{v}(0) \in \mathbb{R}^3$, κ_i, $i = 1, 2, 3$, are scalar gains, and $\hat{R}$ the attitude estimate provided by the attitude observer (1.1). Assume that:

A4: There exists a scalar $c_R > 0$ such that $\|\hat{R}R^T - I_3\| \leq c_R\|[\tilde{p}^T(t)\ \tilde{v}^T(t)]\|$, $\forall t \geq t_0 \geq 0$.

5.3 Stability analysis

The convergence of the translational observer (5.4)- (5.5) is stated in the following theorem.

Theorem 5.3.1 (Global exponential stability of the translational state observer.) *Consider the translational motion of a quadrotor described by (2.12)-(2.13). Define the translational state estimation error as* $\tilde{p} = \hat{p} - p$, $\tilde{v} = \hat{v} - v$.

The observer defined by (5.4) - (5.6) guarantees the exponential stability of $\tilde{p} = 0$ *and* $\tilde{v} = 0$ *for all initial conditions* $[\tilde{p}^T(0)\ \tilde{v}^T(0)] \in \mathbb{R}^3 \times \mathbb{R}^3$ *with the observer gains* $\kappa_i > 0$, $i = 1, 2, 3$.

Proof. Starting by the definition of $\tilde{v} = \hat{v} - v$, the time derivative of $\tilde{v}$, yields:

$$
\begin{aligned}
\dot{\tilde{v}} &= \dot{\hat{v}} - \dot{v} \\
&= \dot{\tilde{v}} + \kappa_3 \dot{p} - \dot{v} \quad \text{using equation (5.6)} \\
&= g e_3 + \hat{R} a_B - \kappa_2(\hat{p} - p) - \kappa_3 \hat{v} + \kappa_3 \dot{p} - \dot{v} \quad \text{by substituting (5.5)} \\
&= g e_3 + \hat{R} a_B - \kappa_2(\hat{p} - p) - \kappa_3 \hat{v} + \kappa_3 v - g e_3 - R a_B \quad \text{using that } \dot{p} = v \text{ and } \dot{v} = g e_3 + R a_B \\
&= (\hat{R} - R) a_B - \kappa_2(\hat{p} - p) - \kappa_3(\hat{v} - v) \quad \text{rearranging the terms} \\
&= (\hat{R} - R) a_B - \kappa_2 \tilde{p} - \kappa_3 \tilde{v} \\
&= (\hat{R} R^T - I_3) R a_B - \kappa_2 \tilde{p} - \kappa_3 \tilde{v}.
\end{aligned}
$$

Also, note that:

$$
\begin{aligned}
\dot{\tilde{p}} &= \dot{\hat{p}} - \dot{p} \\
&= \hat{v} - \kappa_1(\hat{p} - p) - v \quad \text{using equation (5.4) and (2.12)} \\
&= (\hat{v} - v) - \kappa_1(\hat{p} - p) \quad \text{rearranging the terms} \\
&= \tilde{v} - \kappa_1 \tilde{p}.
\end{aligned}
$$

The time derivative of $\hat{v}$ in Eq. (5.6) is

$$\dot{\hat{v}} = g e_3 + \hat{R}^T a_B - \kappa_2 \tilde{p} - \kappa_3 \tilde{v}. \tag{5.7}$$

Therefore, the dynamics of the translational state estimation error $[\tilde{p}^T\ \tilde{v}^T]$ is

$$
\begin{bmatrix} \dot{\tilde{p}} \\ \dot{\tilde{v}} \end{bmatrix} = \begin{bmatrix} -\kappa_1 I_3 & I_3 \\ -\kappa_2 I_3 & -\kappa_3 I_3 \end{bmatrix} \begin{bmatrix} \tilde{p} \\ \tilde{v} \end{bmatrix} + \begin{bmatrix} 0 \\ (\hat{R} R^T - I_3) R a_B \end{bmatrix}. \tag{5.8}
$$

Note that the matrix describing the dynamics of the translational state estimation error (5.8) is Hurwitz for $\kappa_i > 0$, $i = 1, 2, 3$, giving an exponentially stable LTI system with a bounded perturbation given by $\|(\hat{R}R^T - I_3)Ra_B\| \leq \|\hat{R}R^T - I_3\|\|Ra_B\| \leq \bar{c}_R\|[\tilde{p}^T(t)\ \tilde{v}^T(t)]\|$ under Assumption A3, where $\bar{c}_R = c_R\|Ra_B\|$. By Lemma 9.1 of [24], it follows that the equilibrium $[\tilde{p}^T\ \tilde{v}^T] = 0$ is globally exponentially stable with an appropriate choice of gains κ_i, $i = 1, 2, 3$.

Remark 5.3.1 *Assumption A4 may be fulfilled by either tuning the attitude observer gains to converge sufficiently fast or waiting a short time t_0 to start the translational state observer. Note that this assumption is not critical for the convergence of the translational state observer, because the inertial acceleration Ra_B of a quadrotor is much smaller than the gravity acceleration, and the attitude estimation error $(\hat{R}R^T - I_3)$ is always bounded. If A3 is not satisfied, by Theorem 4.18 of [24] that the translational estimation error tends to a neighborhood of the origin with size bounded by $\|\hat{R}R^T - I_3\|\|Ra_B\|$.*

The overall proposed navigation observer has a modular structure. It consists of three cascade modules working together as shown in the diagram Fig. 5.1 to estimate the angular velocity, the attitude, and the translational state, each of them being designed and run independently, ensuring their functionality and robustness to failure. This modular feature also facilitates the convergence analysis and implementation of the navigation observer in practice.

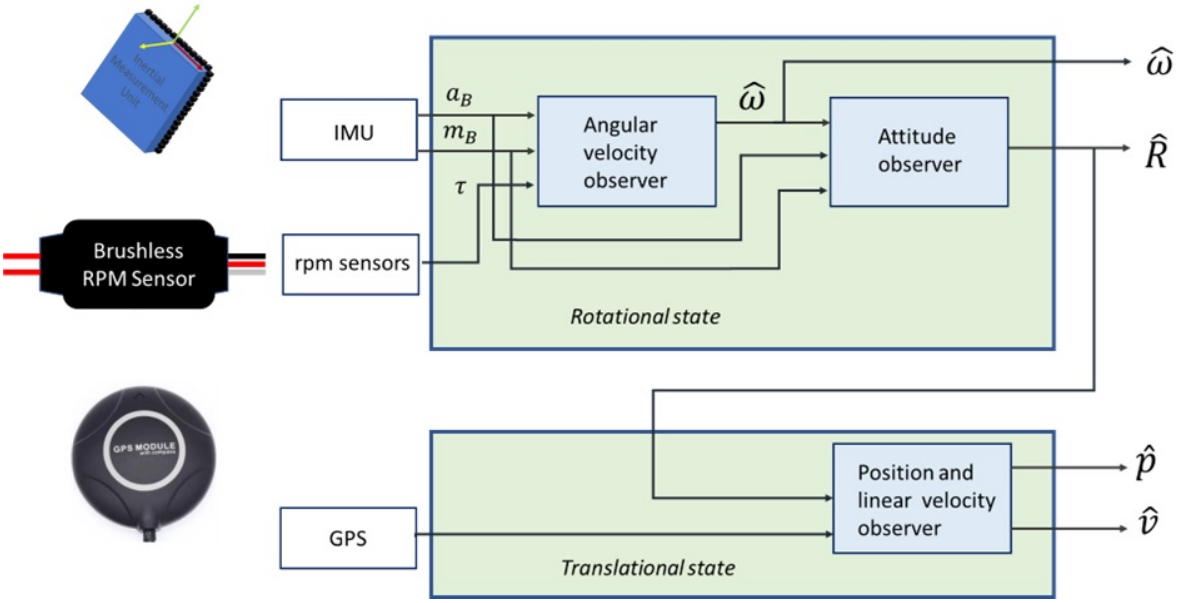


Figure 5.1: Modular structure of the navigation observer.

Chapter 6

Validation by numerical simulations

In this chapter, a series of numerical simulations are presented to validate the proposed observer (3.15), (3.16), (3.5), (4.1), (4.2), (5.4)- (5.6) developed in the previous chapters.

Three scenarios were considered in the simulations. The first scenario illustrates the performance of the observer in an ideal situation where the two reference vectors in the inertial frame are perpendicular and the on-board vector measurements are noise-free. The second scenario simulates a more realistic situation where the angle between the reference vectors is small and the vector measurements are contaminated by a Gaussian noise. Since the angular velocity estimate (3.15) and (3.16) depends on the inertia matrix, the third scenario illustrates the robustness of the observer under uncertainty in the inertia matrix.

We have selected an X-shaped quadrotor for simulation and experimental validation purposes. We want to show the performance of the navigation observer in a simulation environment and see how it performs in real flights, and compare the results in both situations. What we expect is that the simulation results will be highly similar to what happens in reality, which represents an interesting challenge in the implementation.

The quadrotor is a four rotor helicopter. Each motor with propeller mounted over a symmetrical cross rigid structure. Since quadrotors have been very well studied, there is extensive research on them; in addition, they are low-cost compared with other experimental platforms. These reasons make them a suitable technology for academic research.

The quadrotor was simulated to fly in a circular trajectory of radius $15[m]$ given by $r(t) = [15sin(\alpha t)\ 15cos(\alpha t)\ 5]^T$, with $\alpha = \sqrt{1/15}$ using Matlab and Simulink as shown in Fig. 6.1. Note that the magnitude of the linear acceleration is equal to one. Since the linear acceleration is much smaller than the gravity acceleration, the reference vectors of the normalized gravity acceleration $r_1 = [0, 0, -1]^T$ and the local magnetic field $r_2 = [0.6626, 0.0544, 0.7469]^T$ were used. The vector measurements are assumed to be obtained from a low-cost IMU sensor, from which only measurements of the accelerometer and the magnetometer are used and modeled as $b_1 = \frac{\hat{R}^T r_1}{\|\hat{R}^T r_1\|}$, $b_2 = \frac{\hat{R}^T r_2}{\|\hat{R}^T r_2\|}$. To simplify the implementation, the attitude observer was implemented in its quaternion representation by

$$\dot{\hat{q}} = \frac{1}{2}\hat{q} \otimes [0 \quad \hat{\omega}^T + k_p \sigma_R^T]^T \tag{6.1}$$

where $\hat{q} = [\hat{q}_0,\ \hat{q}_v]^T$ is the estimate of the attitude in terms of the unit quaternion $q \in \mathbb{S}^3$, with $\mathcal{S}^3 = \{q \in \mathbb{R}^4 | \|q\| = 1\}$ the unit sphere of dimension 3, and $\otimes$ denotes the product of

quaternions. The estimate $\hat{b}_i$ can be obtained by $\hat{b}_i = \hat{q} \otimes [0\ r_i^T]^T \otimes \hat{q}^*$, where $\hat{q}^* = [\hat{q}_0,\ -\hat{q}_v]^T$ is the conjugate of q. The resultant quaternion estimate from (6.1) was normalized to ensure numerical correctness. The quaternion error is calculated as: $\tilde{q} = \hat{q}^{-1} \otimes q$.

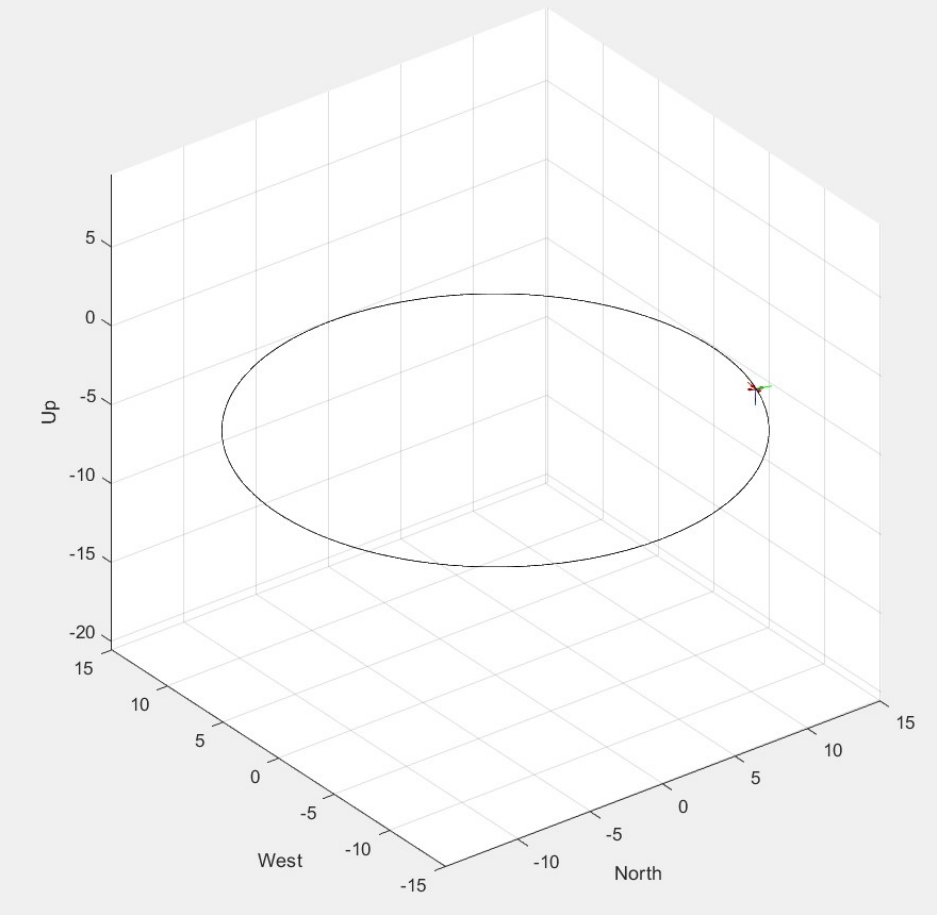
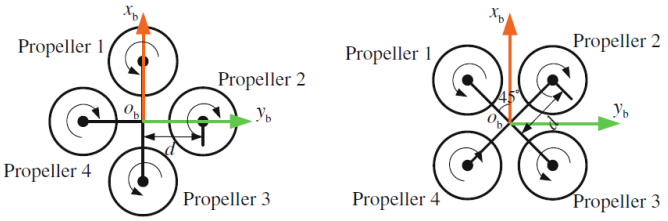


Figure 6.1: Circular path flight executed by the quadrotor in Simulink.

6.1 Quadrotor model

The propeller angular speeds, ω_i, $i = 1, 2, 3, 4$ will determine the total thrust f and moments τ [36].



(a) Plus-configuration (b) X-configuration

Figure 6.2: Plus and X configuration of a quadrotor. *Image credit* [36].

For a plus-configuration quadrotor, the total thrust that acts on the quadrotor and the moments produced by propellers are (6.2):

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ 0 & -dc_T & 0 & dc_T \\ dc_T & 0 & -dc_T & 0 \\ c_M & -c_M & c_M & -c_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{6.2}$$

The total thrust and moments produced by the propellers in an X-configuration quadrotor are (6.3):

$$\begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ dc_T\frac{\sqrt{2}}{2} & -dc_T\frac{\sqrt{2}}{2} & -dc_T\frac{\sqrt{2}}{2} & dc_T\frac{\sqrt{2}}{2} \\ dc_T\frac{\sqrt{2}}{2} & dc_T\frac{\sqrt{2}}{2} & -dc_T\frac{\sqrt{2}}{2} & -dc_T\frac{\sqrt{2}}{2} \\ c_M & -c_M & c_M & -c_M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{6.3}$$

where f represents the total thrust that acts on the quadrotor, τ_x, τ_y and τ_y are the moments, d is the distance between the body center and any motor, c_T, the thrust coefficient, c_M, the moment coefficient and ω_i $i = 1, .., 4$ are the angular speeds of each motor.

Assuming that the speeds of the motors are known at all times, the applied torque τ and thrust f can be calculated through (6.3). The physical data of the real quadrotor were used in the simulation and summarized in the next section in Tab. 7.2.

Given the speed of the motors and the physical parameters of the quadrotor: c_T, c_M and d, the components for τ in [Nm] units, can be modeled as the following functions of time:

$$\tau_x = (0.000019919) + (0.00020641) * sin(0.60803 * t - 1.6324) \tag{6.4}$$
$$\tau_y = (-6.0042e - 007) + (0.000092638) * sin(0.60746 * t + 3.1044) \tag{6.5}$$
$$\tau_z = (1.3528e - 008) + (0.00026022) * sin(0.60816 * t - 0.068381) \tag{6.6}$$

By generating τ, we can find the true value of the angular velocity ω by integrating equation (2.11) and the true value of attitude R by integrating equation (2.7) and compared these results with the estimated values $\hat{\omega}$, $\hat{R}$ output by the observer (3.15), (3.16), (3.5), (4.1), (4.2). Now, for the translational motion, we know the trajectory followed by the quadrotor, this means that we know the real values of the position p, the linear velocity v and the linear acceleration $\dot{v}$. We also know the model of the accelerometer and magnetometer sensors. The true values are compared to estimates given by the observer (5.4)- (5.6).

The initial conditions and parameters are shown in Table 6.1.

Table 6.1: Initial conditions and parameters used in simulations.

Initial state	Value	Units
$q(0)$	$[1\ 0\ 0\ 0]^T$	
$\hat{q}(0)$	$[0.7874\ 0.2\ -0.5\ -0.3]^T$	
$\omega(0)$	$[0.01\ 0.01\ 0.2]^T$	[rad/s]
$\bar{\omega}(0)$	$[1\ 1\ 1]^T$	[rad/s]
$b_{1f}(0)$	$[0\ 0\ -1]^T$	
$b_{2f}(0)$	$[0\ 1\ 0]^T$	
$\bar{v}(0)$	$[0\ 0\ 0]^T$	[m/s]
$\hat{p}(0)$	$[0.2\ 0.2\ 0.2]^T$	[m]
Parameters		
γ_f	5.0	
k_p	1.0	

6.2 Error Analysis

In order to estimate the reliability between the true values compared with the estimated values, we wish to quantify the error between these two measurements. For error analysis, *Root Mean Square Error* (RMSE) is a frequently used measure of the differences between the values (sample or population values) predicted by a model or estimator and the observed values.

Definition 6.2.1 *Root mean squared error (RMSE) is defined as*

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \|\hat{y}_i - y_i\|^2}$$

where $y_i \in \mathbb{R}^3$ is the observation, $\hat{y}_i \in \mathbb{R}^3$ is the real value of a variable, and N the number of observations available for analysis.

The equation above is useful to evaluate the errors in the observers for angular velocity, position and linear velocity. In case of quaternions, we have previously established that the error between the true quaternion and the estimated quaternion is calculated as: $\hat{q}^{-1} \otimes q$, which will be compared with the identity quaternion $[1; 0; 0; 0]$.

6.3 Scenario I. Noise-free case

This scenario represents the case where vector measurements are noise-free. Note that the parameters used were selected to assign the same confidence in the magnetometer than in the accelerometer because of the very small linear acceleration. Note that as the gravity and the magnetic field of Earth are not perpendicular to each other, there is a small angle between them, this represent the situation as would be a location near the above the Equator of the Earth. Scalar gains were selected as $k_1 = 5.0$, $k_2 = 5.0$, $\kappa_1 = 1.0$, $\kappa_2 = 1.0$, $\kappa_3 = 5.0$, $\Lambda_1 = diag[0.15,\ 0.15,\ 0.15]$, $\Lambda_2 = diag[0.15,\ 0.15,\ 0.15]$. Fig. 6.3 shows the performance of the observer.

We observe from the graphs Fig. 6.3 that the variables show an exponential convergence to zero within the first five seconds approximately, under ideal conditions. In the following set of graphs 6.4, an error analysis in terms of Root Mean Square Error is shown. We can see that the settling time for the angular velocity error remains stable after $10s$ and within a time interval of $\tilde{\omega}_{RMSE} \leq 1 \times 10^{-4} rad/s$, for the quaternion estimation error within $(\hat{q}^{-1} * q)_{RMSE} \leq 1 \times 10^{-6}$, and for the position and linear velocity error within $\tilde{p}_{RMSE} \leq 1 \times 10^{-6} m$ and $\tilde{v}_{RMSE} \leq 1 \times 10^{-6} m/s$, respectively.

6.4 Scenario II. Noisy case

In this scenario, the vector measurements were contaminated by a Gaussian noise of $n_{b,i} \in \mathbf{N}(0, 0.1^2)$. In this case, the scalar gains were selected as $k_1 = 5.0$, $k_2 = 5.0$, $\kappa_1 = 1.0$, $\kappa_2 = 1.0$, $\kappa_3 = 5.0$. $\Lambda_1 = diag[0.0025,\ 0.0025,\ 0.0025]$, $\Lambda_2 = diag[0.0025,\ 0.0025,\ 0.0025]$. Fig. 6.5 illustrates the results.

From the plots shown in Fig. 6.5, we can see that under noisy conditions the convergence to zero is achieved at approximately a settling time of $20s$. Furthermore, in the following plots Fig. 6.6 we can see the Root Mean Square Error for each variable, which shows it remains within an interval of 1×10^{-2}.

6.5 Scenario III. Uncertain inertia matrix

This scenario shows the performance of the observer in case of uncertainty in the inertia matrix. The inertia matrix in the angular velocity observer (3.15), (3.16), (3.5), was perturbed by ΔM, with $M_{pert} = M + \Delta M_i$, i=1,2. The values of ΔM_1 and ΔM_2 were selected as:

$$\Delta M_1 = \begin{bmatrix} 0.0003 & 0 & 0 \\ 0 & 0.0006 & 0 \\ 0 & 0 & -0.0021 \end{bmatrix}, \ \Delta M_2 = \begin{bmatrix} -0.0015 & 0 & 0 \\ 0 & -0.005 & 0 \\ 0 & 0 & 0.002 \end{bmatrix}$$

such that M_{pert} is positive definite with $\|\Delta M_1\| \approx 0.1\|M\|$, $\|\Delta M_2\| \approx 0.2\|M\|$. Scalar gains were selected as: $k_1 = 5.0$, $k_2 = 5.0$, $\kappa_1 = 1.0$, $\kappa_2 = 0.2$, $\kappa_3 = 0.5$, $\Lambda_1 = diag[0.25,\ 0.25,\ 0.25]$, $\Lambda_2 = diag[0.25,\ 0.25,\ 0.25]$. The behavior of the observer under these circumstances is illustrated in Fig. 6.7-Fig. 6.8.

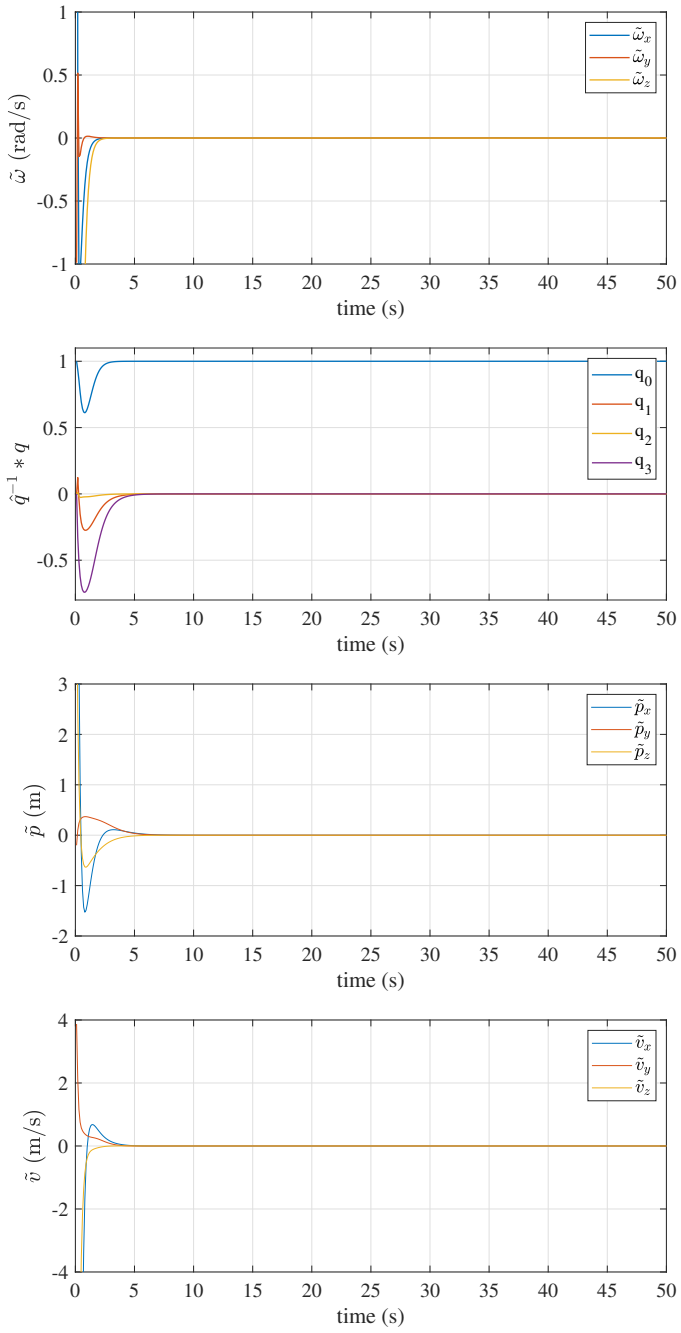


Figure 6.3: Scenario I (Noise-free case): (Top-down) angular velocity error, attitude error $\hat{q}^{-1} * q$, position error $p - \hat{p}$, and linear velocity error $v - \hat{v}$.

It is interesting to note that the convergence rate of the estimation errors occurs in less time with respect to the noise-free case. However, the errors increased after a short period, as shown in Fig.6.7. A similar behavior occurs with the observer of [3]. This indicates that

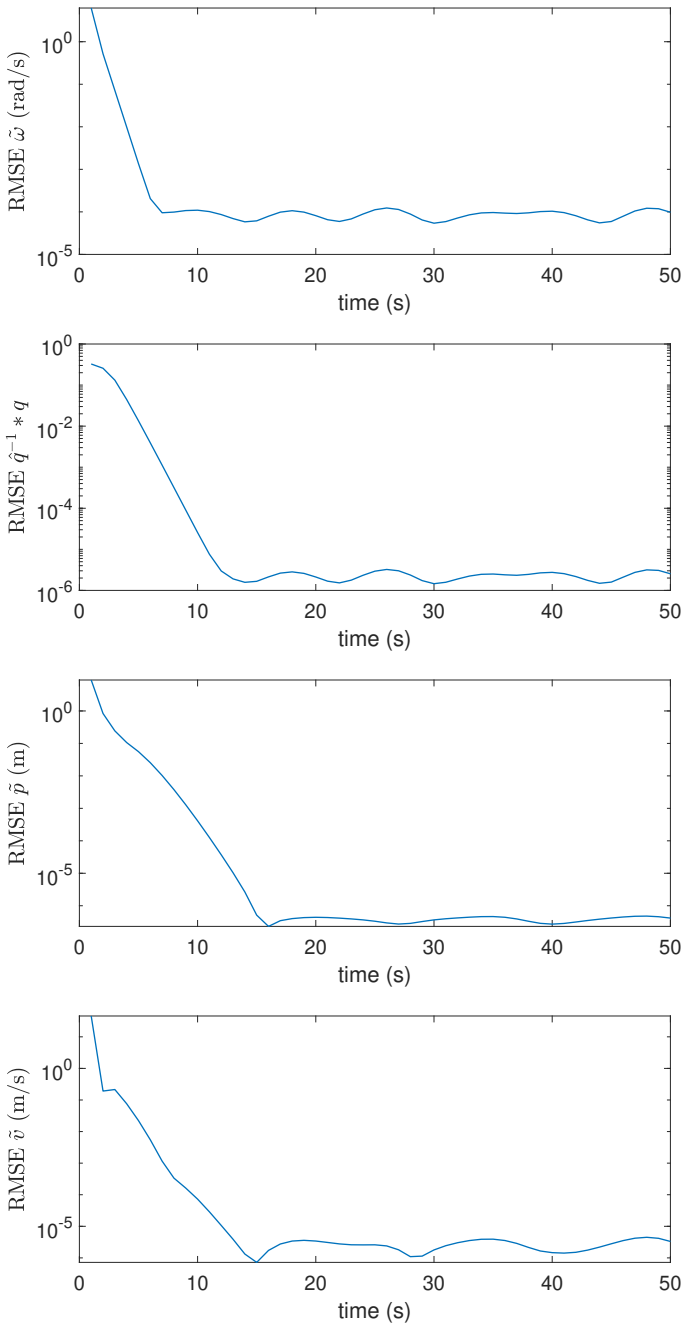


Figure 6.4: Scenario I (Noise-free case): (Top-down) Root Mean Square estimate for angular velocity error $\tilde{\omega}$, attitude error $\hat{q}^{-1} * q$, position error $\tilde{p}$ and linear velocity error $\tilde{v}$.

the proposed observer is sensitive to uncertainty in the inertia matrix and may be relevant when the uncertainty is greater than or equal to 20%.

These results encourage us to establish that the proposed observer is a good candidate

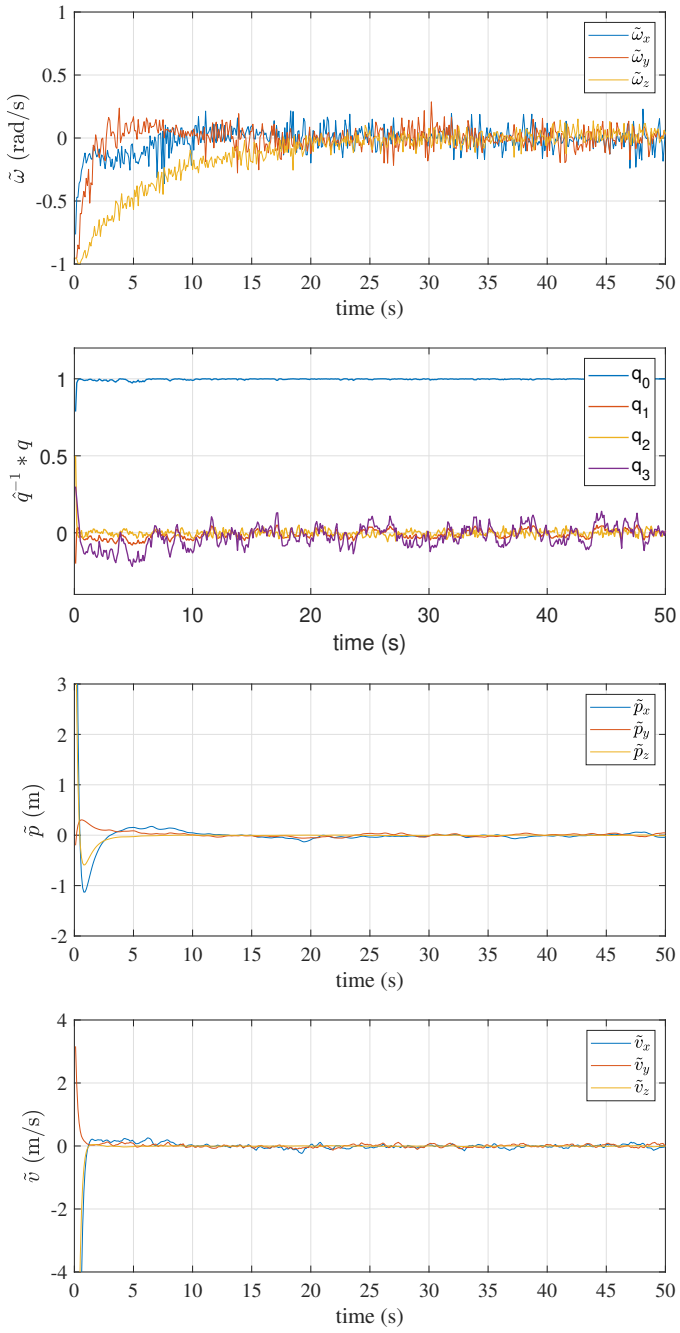


Figure 6.5: Scenario II (Noisy case): Top-down: Estimate for angular velocity error $\tilde{\omega}$, attitude error $\hat{q}^{-1} * q$, position error $\tilde{p}$ and linear velocity error $\tilde{v}$.

for implementation in autonomous aerial vehicles.

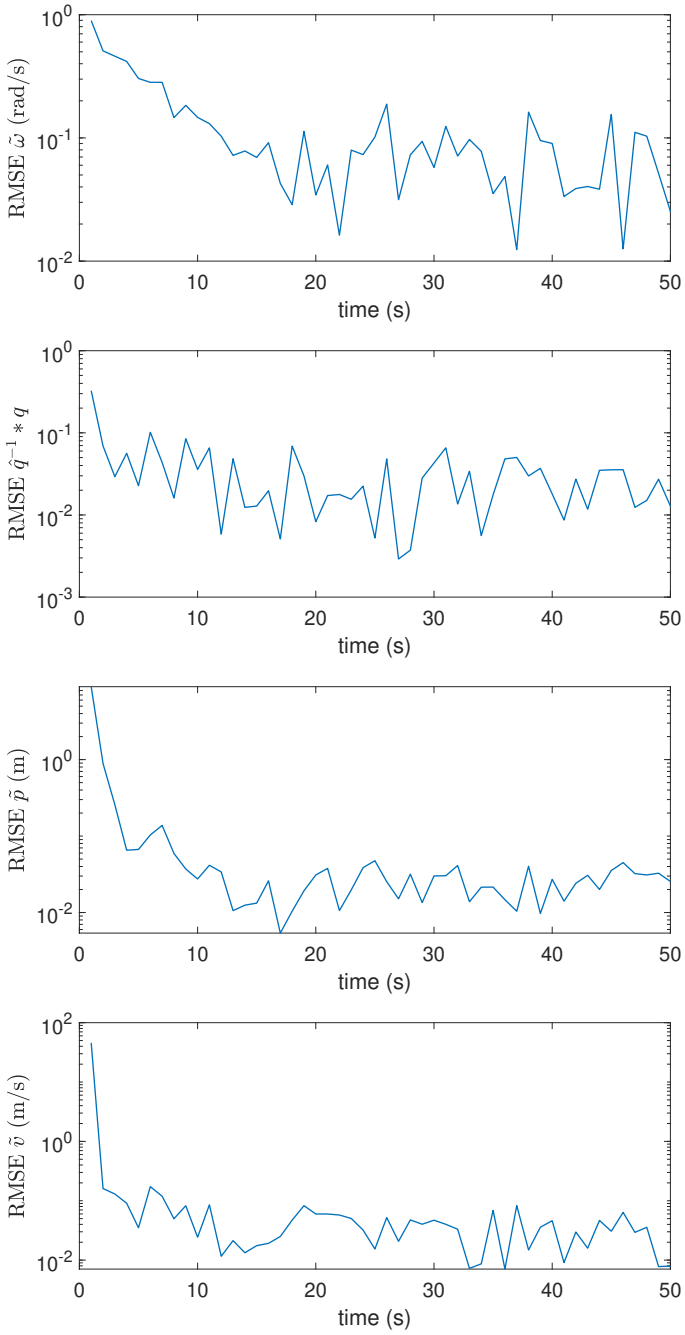


Figure 6.6: Scenario II (Noisy case): (Top-down) Root Mean Square estimate for angular velocity error $\tilde{\omega}$, attitude error $\hat{q}^{-1} * q$, position error $\tilde{p}$ and linear velocity error $\tilde{v}$.

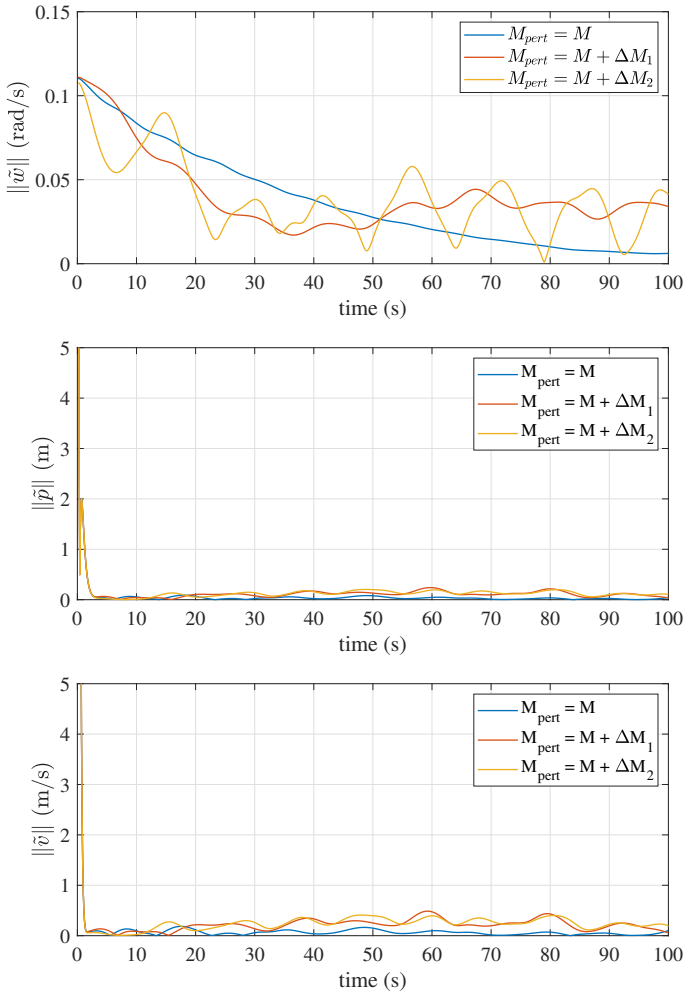


Figure 6.7: Scenario III (Uncertain inertia matrix): norm of (top-down) the angular velocity estimation error, position, and linear velocity estimation error.

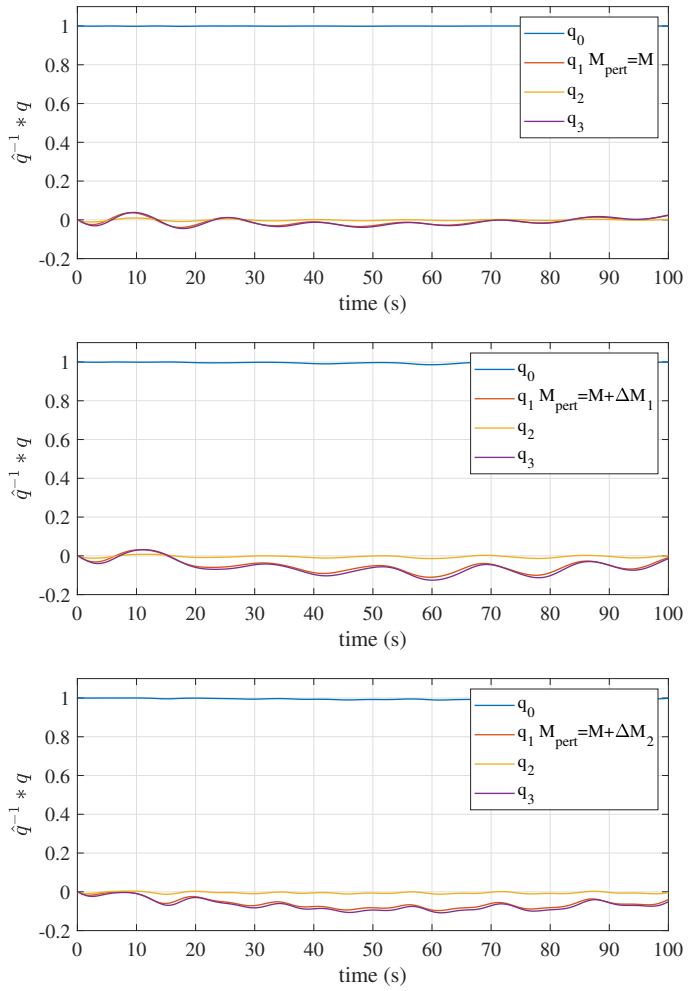


Figure 6.8: Scenario III (Uncertain inertia matrix): Attitude error $\hat{q}^{-1} * q$.

Chapter 7

Experimental validation

In this chapter, we present validation results of the angular velocity, attitude observer and translational state observer. The experiments were carried out in our laboratory. A detailed description of the construction of an experimental platform based on a quadrotor is given aimed at testing the proposed observer in real-world applications. This platform was developed from scratch in the Mechatronics Laboratory at *Unidad de Alta Tecnología* campus Juriquilla, UNAM Querétaro. The main objective of this platform is to close the gap between theoretical results on the design of our estimation and control algorithms for autonomous vehicles.

An overview of the hardware and firmware architecture is presented and the experimental results are discussed.

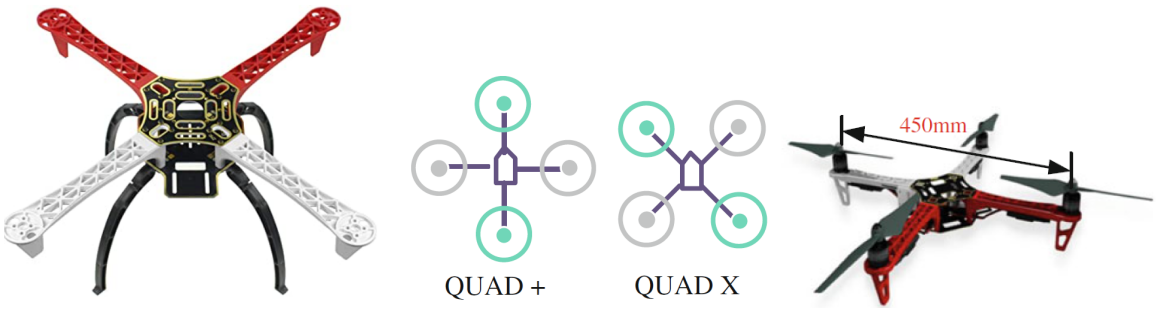
7.1 Experimental platform

The experimental platform consists of an open architecture quadrotor, with a *Pixhawk* autopilot on board, which contains an IMU with accelerometer, magnetometer, and gyroscope. The embedded algorithm in the IMU issues as the output the bias-corrected angular velocity and the attitude in Euler angles, which are taken as the "true" angular velocity and the attitude of the quadcopeter to compared with.

The firmware consists of *Ardupilot*, which was modified to acquire the signals from the motor speed sensors of the brushless DC motor of the quadcopeter to determine the total torque acting on the quadrotor. The experiment involves programming an autonomous mission on the quadrotor to execute flights in an outdoor environment. The flight data are processed to perform the analysis offline. The algorithm uses data from the accelerometer and the magnetometer to estimate the angular velocity of the quadcopeter. The estimated angular velocity is forwarded to the Explicit Complementary Filter to estimate the attitude of the quadcopeter quaternions. For an intuitive comparison, Euler angles are also plotted. Therefore, the output of the algorithm is the estimated angular velocity $\hat{\omega}$ and the estimated attitude $\hat{q}$, $\hat{\theta}$, $\hat{\phi}$, $\hat{\psi}$. These data are compared with the angular velocity ω and the attitude in terms of Euler angles θ, ϕ, ψ, provided by the microcontroller, which uses a *Kalman Filter* to output the attitude.

7.1.1 Hardware architecture

The platform that carries all the equipment of the quadrotor is illustrated in Fig. 7.1(a). The most common configuration is the shape of plus + and cross X shown in Fig. 7.1(b). The diameter (in mm) (diagonal size) is the circumcircle determined by the motor axes. In general, it is the distance between the motor axes in the diagonal line to indicate the size of an airframe (Fig. 7.1(c)). The diagonal size restricts the propeller size, which in turn determines the maximum thrust and thus the payload capacity.



(a) ***F450 Frame*** **(b)** ***Quadrotor main configurations*** **(c)** ***Diagonal size***

Figure 7.1: Quadrotor frame

Some of the main components for the platform construction are illustrated in Fig. 7.2 and are described below.

Figure 7.2: quadrotor components.

Frame

Flight Controller

A flight controller is an integrated circuit capable of executing commands stored in its memory, which monitors and controls the UAV. Basically, is the brain of the vehicle. It is composed of a central processing unit, memory and control unit. The flight controller is connected to a set of sensors. These sensors give information about its height, attitude, and speed. The vehicle filters this information and fuses some to calculate the desired speed for each of the four motors. The flight controller sends this desired speed to the Electronic Speed Controllers (ESC's), which translates this desired speed into a signal that the motors can read. Another task performed by the controller board is to maintain communication between the vehicle and the ground station, for example, to communicate flight status, battery level, and to send and receive information. A wide variety of flight controllers are available in the market. An overview about advanced autopilot, open source projects can be found in [25, 48].

Figure 7.3: *Pixhawk* Flight Controller.

We selected a *Pixhawk* (Fig. 7.3) as the flight controller due to its low cost and high performance. These features make it very popular for aerial robots [4, 17, 32]. Some of its main characteristics are shown below [35].

- Processor
  - 32-bit ARM Cortex M4 core with FPU
  - 168 Mhz/256 KB RAM/2 MB Flash
  - 32-bit failsafe co-processor
- Sensors
  - MPU6000 as the main accelerometer and gyroscope
  - ST Micro 16-bit gyroscope
  - ST Micro 14-bit accelerometer/compass
- Power
  - Ideal diode controller with automatic failover
  - Servo rail high-power (7 V) and high-current ready

  - All peripheral outputs over-current protected, all inputs ESD protected

- Interfaces
  - 5x UART serial ports, 1 high-power capable, 2 with HW flow control
  - PPM sum signal
  - PWM or voltage imput
  - I2C, SPI, 2x CAN, USB
  - 3.3V and 6.6V ADC inputs

- Dimensions
  - Weight 38 g, Width 50 mm, Height 15.5 mm , Length 81.5 mm

Brushless Motor.

The function of brushless DC motors is to convert electrical energy, stored in the battery, into mechanical energy for the propeller. The size of the motor is generally represented by its stator size with a four-digit number, such as motor 2212 , among which the first two indicate its stator diameter (mm) and the latter two indicate its stator height (mm).

The value KV for brushless DC motors is the number of RPM that the motor will revolve when $1(Volt)$ is applied without a load attached to the motor. For example, $1000KV$ means that when $1V$ is applied, the no-load motor speed will be $1000RPM$. The maximum current or power the motor can handle is denoted by two numbers. For example, the maximum continuous current $25A/30s$ represents that the motor can work safely with continuous current up to 25A, beyond which the motor may be burnt out for more than $30s$. The quadrotor of the platform uses four brushless motors $A2212/10T$ $1000KV$ with Hobbyking ESC $30A$ as shown in Fig. 7.4.

Figure 7.4: Motor Brushless and ESC used in our Quad.

Electronic Speed Controller (ESC).

An ESC is an electronic circuit device that controls and adjusts the speed of the electric motor. A signal from the flight controller causes the ESC to raise or lower the voltage to the

motor as required, thus changing the speed of the propeller. ESC can also handle active or regenerative braking, a process by which mechanical energy from a motor is converted into electrical energy that can be used to recharge the battery. During periods when the motor is decelerating, the motor can act as a generator and the ESC handles excess current that can be fed back into the battery.

An electronic speed control has three sets of wires. One wire plugs into the main battery, the second wire has a standard servo wire that plugs into the throttle channel of the receiver. And finally, the third wire whose function is to power the motor Fig. 7.5.

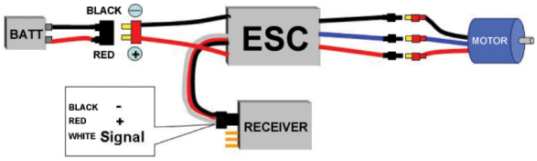


Figure 7.5: ESC Connection Diagram

Propeller.

The propeller is used to produce the thrust and torque to control a quadrotor. The motor efficiency varies with the output torque which depends on the type, size, speed, and other factors of a propeller. Therefore, a good match should ensure that the motor works in a high efficiency condition, which will guarantee less power consumed for the same thrust and then extend the time of endurance of the quadrotor. Therefore, choosing the appropriate propellers is a very direct way to improve the performance and efficiency of a quadrotor.

Generally, the propeller model is described by a four-digit number, such as a 1045 (or 10×45) propeller, among which the first two represent the diameter of the propeller (in *inch*), and the latter two represent the propeller pitch (also referred to as screw pitch, blade pitch, or simplified as pitch in *inch*. The quadrotor of the platform uses a 1045 propeller (Fig. 7.6).

Figure 7.6: Propeller 1045.

Power Module.

The power module (Fig. 7.7) is the bridge between the main power supply and the platform. It has two $XT60$ connectors (female-male), one for the battery connection and the

other where the motors will be connected through a power distribution board. It also has a current/voltage sensor associated with the battery to report the remaining charge, which is useful to stop the flight before the battery discharge.

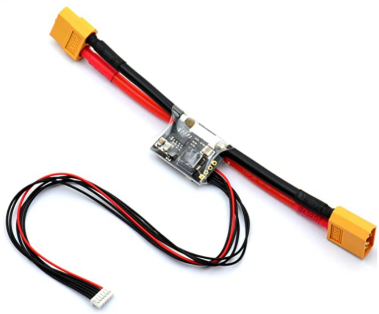
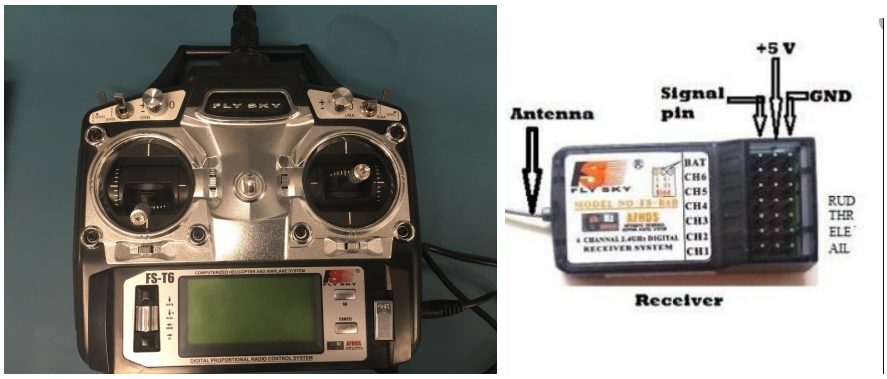


Figure 7.7: Power Module.

Radio Control.

An RC transmitter (Fig. 7.8(a)) is used to transmit commands from remote pilots to the corresponding receiver. The receiver in Fig. 7.8(b) passes the commands to the autopilot after decoding them. Finally, the quadrotor flies according to the commands.



(a) ***RC Transmitter*** **(b)** ***Receiver***

Figure 7.8: Fly Sky RC transmitter and receiver.

Battery.

An electric battery is a device with one or more electrochemical cells that convert stored chemical energy into electrical energy to power all the components of the system. Batteries are rechargeable, and they can be found in many types such as Lithium Polymer (LiPo), Nickel Metal Hydride (NiMH), Lithium ion (Li-ion) and Nickel Cadmium.

Battery cells consist of a positive anode electrode, a negative cathode electrode, and electrolytes generating a potential difference or voltage that allows ions to move between the electrodes and the electric current to flow. Lipo batteries are the most common for aerial

vehicles.

In lithium polymer batteries, each cell has a nominal voltage of 3.7 volts with a maximum voltage of 4.2 volts and a minimum of 3.0 volts. These voltages must be strictly observed to avoid irreparable damage. The number of cells is specified with the nomenclature 1s- 6s, where 1s means one cell, $2s$ means two cells, and so on. We use a Turnigy Lipo battery of 3 cells, discharged $2200mAh$ and $30 - 40$, as can be seen in Fig. 7.9.

Figure 7.9: Turnigy Lipo Battery

GPS sensor.

The Global Positioning System (GPS) is a U.S.-owned utility that provides users with positioning, navigation, and timing (PNT) services. This system consists of three segments: the space segment, the control segment, and the user segment. The U.S. Space Force develops, maintains, and operates the space and control segments. The space segment consists of a nominal constellation of 24 operating satellites that transmit one-way signals that give the current GPS satellite position and time. The control segment consists of worldwide monitor and control stations that maintain the satellites in their proper orbits through occasional command maneuvers and adjust the satellite clocks. It tracks the GPS satellites, uploads updated navigational data, and maintains health and status of the satellite constellation. The user segment consists of the GPS receiver equipment, which receiver the signals from the GPS satellites and uses the transmitted to calculate the vehicle's position and time.

The GPS receiver mounted in the quadrotor is a UBlox GPS + Compass module (Fig. 7.10). The recommended orientation is to mount the module with the arrow facing toward the front of the vehicle and in the same direction as the arrow on the autopilot.

Figure 7.10: UBlox GPS plus Compass module.

RPM Sensors.

It detects voltage changes on the wires of the brushless motors and then outputs the RPM signal to measure the main rotor speed and motor/engine RPM. This RPM sensor can work with some speed control systems. Common types of RPM sensor that can be used in ArduPilot are: Hall effect, ESC telemetry, electrical commutation, and optical [5]. We selected four *Hobbywing* RPM sensors (Fig. 7.11) to measure the speed of each motor [21].

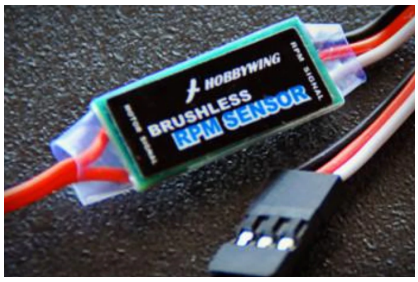


Figure 7.11: *Hobbywing* Brushless RPM Sensor

The quadrotor developed in Mechatronics Laboratory is shown in Fig. 7.12.

Figure 7.12: quadrotor developed in our laboratory

7.1.2 Software architecture

Pixhawk supports Software Development Kit (SDKs) called *ArduPilot* and *PX4*. *PX4* is an open source flight control software for drones and other unmanned vehicles. The project provides a flexible set of tools for drone developers to share technologies to create solutions for drone applications. *PX4* provides a standard to deliver the hardware support and software stack, allowing an ecosystem to build and maintain hardware and software in a scalable way. *PX4* is part of *Dronecode*, a non-profit organization administered by Linux Foundation to foster the use of open source software on flying vehicles. *Dronecode* also hosts *QGroundControl*, *MAVLink* and the *SDK*.

The *ArduPilot* software suite consists of navigation software (typically referred to as firmware when it is compiled to binary form for microcontroller hardware targets) running on the vehicle, along with ground station controlling software including *Mission Planner*, *APM Planner*, *QGroundControl*, it MavProxy, *Tower* and others.

7.1.2.1 Ardupilot

Ardupilot is an advanced open source autopilot available for autonomous unmanned vehicle systems. It has been under development since 2010 by a world wide team of professional engineers, computer scientists, and community contributors. *ArduPilot* firmware works on a wide variety of different hardware to control unmanned vehicles of all types: multicopters, airplanes, helicopters, rovers, antenna trackers and submarines. Coupled with ground control software, unmanned vehicles running ArduPilot can have advanced functionality including real-time communication with operators. It is continually being expanded to provide support for new emerging vehicle types.

The basic structure of ArduPilot is broken up into 5 main parts:

- Vehicle Code: The vehicle directories are the top level directories that define the firmware for each vehicle type. Currently there are 6 vehicle types: Plane, Copter, Rover, Sub, Blimp and AntennaTracker. There are a lot of common elements between different vehicle types. Along with the cpp files, each vehicle directory contains a wscript file which lists library dependencies.
- Shared libraries: These libraries include sensor drivers, attitude and position estimation (aka EKF) and control code (i.e. PID controllers)
- Hardware abstraction layer (AP_HAL): Is how the ArduPilot is portable to lots of different platforms, for example AVR based boards, Pixhawk boards or Linux based boards.
- Tools directories: The tools directories are miscellaneous support directories.
- External Support code: Some platforms need external support code to provide additional features or board support. Currently the external trees are: PX4NuttX, PX4Firmware, uavcan and mavlink.

Our project needs to customize the firmware of *Ardupilot* because we need to know the torque produced by the propellers that act on the drone at all times. This can be done by measuring the speeds of the four motors. Currently, the latest version of the firmware only considers two ports for RPM sensors, and in our case, we need 4. So in the appendix we present the code modified and implemented in the Pixhawk in order to have the ability to read, write, and save (log) the readings of the 4 sensors.

7.1.2.2 Mission planner

Mission Planner (Fig. 7.13) [34] is a ground control station application for the *Ardupilot* open source autopilot project for Plane, Copter and Rover, compatible with Windows only. Mission Planner can be used as a configuration utility or as a dynamic control supplement for autonomous vehicles. It is a board-user interface. This program contains a wide range of options and information available on the different aspects of the *Ardupilot* hardware and firmware. Some functions are as follows:

- Load the firmware (the software) into the autopilot board (i.e. Pixhawk series) that controls the vehicle.
- Setup, configure, and tune your vehicle for optimum performance.
- Plan, save and load autonomous missions into you autopilot with simple waypoints entry on Google or other maps.
- Download and analyze mission logs created by your autopilot.
- Interface with a PC flight simulator to create a full hardware-in-the-loop UAV simulator.
- Monitor the vehicle's status while in operation with appropriate telemetry hardware.

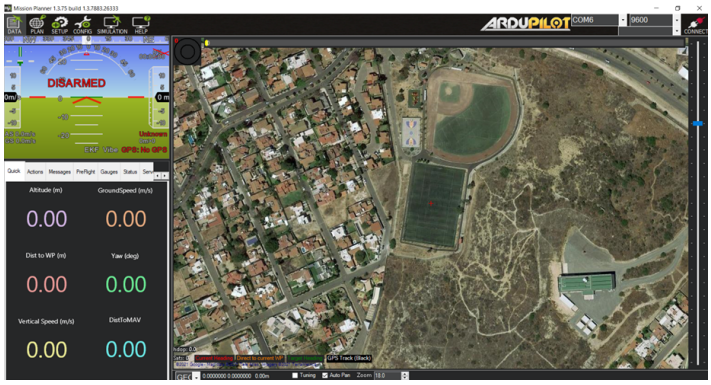


Figure 7.13: Mission Planner Main Menu

The main features used in this project were:

- Installing Mission Planner
- Loading custom Firmware
- Connect Mission Planner to AutoPilot
- Sensor calibration
- Programming an autonomous mission

These tasks are briefly summarized below.

Installing Mission Planner

Mission Planner was designed for native Windows installation. Download the latest Mission Planner installer [33] and double click on the downloaded .msi file to run the installer. Follow the instructions to complete the setup process. The installation utility will automatically install any necessary software drivers.

Loading custom Firmware

After customizing the firmware, an *apj* an executable file is generated. This file was loaded into Pixhawk hardware. First, open Mission Planner, then select *Initial Setup* and choose the option *Load custom firmware* as shown in Fig. 7.14. Select the path were the apj is located. If all goes well, you will see a status appear on the bottom right including the words: "erase...", "program...", "verify..", and "Upload Done". The firmware has been successfully uploaded to the board. Wait to press CONNECT until this occurs.

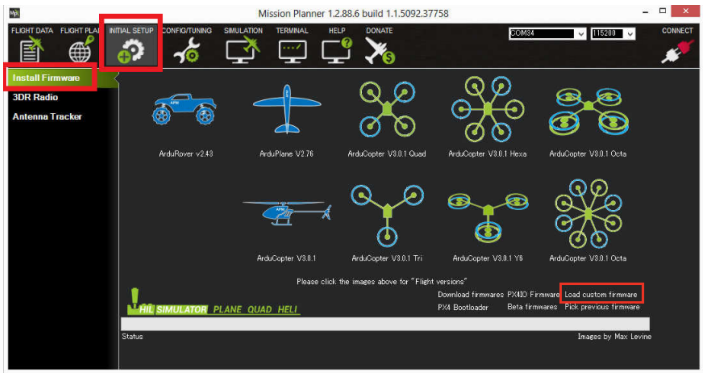


Figure 7.14: Load custom firmware

Connect Mission Planner to Autopilot

Connect the autopilot Pixhawk using the micro USB cable as shown and use a direct USB port on the computer, Fig. 7.15. Select the COM port drop-down in the upper-right corner of the window near the Connect button. Select AUTO or the specific port for your board. Set the Baud rate to 115200 and press Connect.

Figure 7.15: Autopilot to computer connection

Sensor calibration

The first time to setup, some required hardware components are needed to be configured. This process starts by selecting frame orientation and configuring the RC transmitter/receiver, compass, accelerometer and ESCs using Mission Planner. These sensors require a one-time calibration during the Mandatory Hardware setup step.

Radio Control Calibration

RC transmitters allow the pilot to set the flight mode, control the vehicle's movement and orientation and also turn on/off auxiliary functions. This involves capturing each RC input channel's minimum, maximum and "trim" values so that ArduPilot can correctly interpret the input. Some green bars should appear showing the ArduPilot is receiving input from the Transmitter/Receiver Fig. 7.16. Then move the transmitter's roll, pitch, throttle and yaw

sticks and ensure the green bars move in the correct direction and click when done. Mission Planner will show a summary of the calibration data.

Figure 7.16: RC calibration

Accelerometer Calibration

To perform basic accelerometer calibration using Mission Planner is a mandatory procedure. Mission Planner will prompt you to place the vehicle each calibration position as shown in Fig. 7.17. Press any key to indicate that the autopilot is in position and then proceed to the next orientation. It is important that the vehicle is kept still immediately after pressing the key for each step. Click when Done button once each position is reached and held still. When you have completed the calibration process, Mission Planner will display "Calibration Successful!".

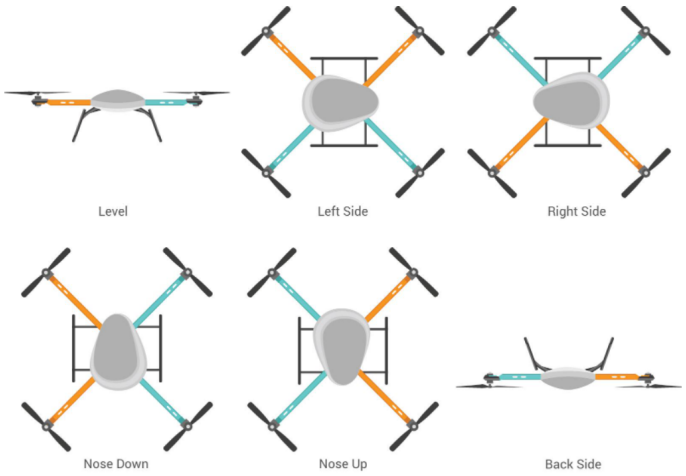


Figure 7.17: Accelerometer calibration. Diagram credit: [1].

Compass calibration

For basic compass calibration, click the "Onboard Mag Calibration" and "Start" button Fig. 7.18. Hold the vehicle in the air and rotate it so that each side (front, back, left, right, top and bottom) points down towards the earth for a few seconds in turn. Consider

a full 360-degree turn with each turn pointing a different direction of the vehicle to the ground. It will result in 6 full turns plus possibly some additional time and turns to confirm the calibration. As the vehicle is rotated the green bars should increase in length until the calibration completes. Finally, reboot the autopilot and the calibration will be completed.

It is not necessary to recalibrate the compass when the vehicle is flown at a new location because ArduPilot includes a "world magnetic model" which allows converting the location's magnetic North to true North without recalibrating. It is important that when compass calibration is done, the vehicle have a good 3D gps lock, in order to assure the best setup. If necessary, move outdoors in order to get a good 3D gps lock before doing the compass calibration.

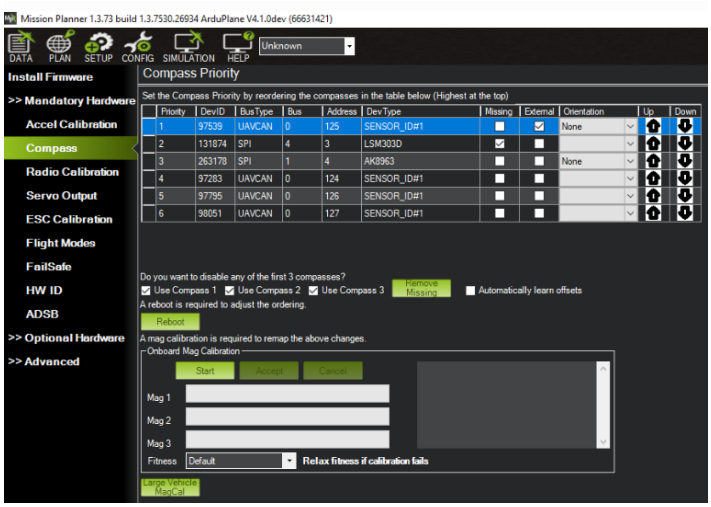


Figure 7.18: Compass calibration

ESC Calibration

Electronic speed controllers are responsible for spinning the motors at the speed requested by the autopilot. Most ESCs need to be calibrated so that they know the minimum and maximum pwm values that the flight controller will send. The recommended procedures are: "All at once" calibration, "Manual ESC-by-ESC" method or the Semi Automatic ESC-by-ESC Calibration routine in Mission Planner. We followed the Manual ESC by ESC method:

1. Plug one of your ESC three-wire cables into the throttle channel of the RC receiver.
2. Turn on the transmitter and set throttle stick to maximum.
3. Connect the LiPo battery. You will hear a musical tone then two beeps.
4. After the two beeps, lower the throttle stick to full down.
5. You will then hear a number of beeps (one for each battery cell you're using) and finally a single long beep indicating the end points have been set and the ESC is calibrated.
6. Disconnect battery. Repeat these steps for all ESCs.

7. If it appears that the ESC's did not calibrate then the throttle channel on the transmitter might need to be reversed.

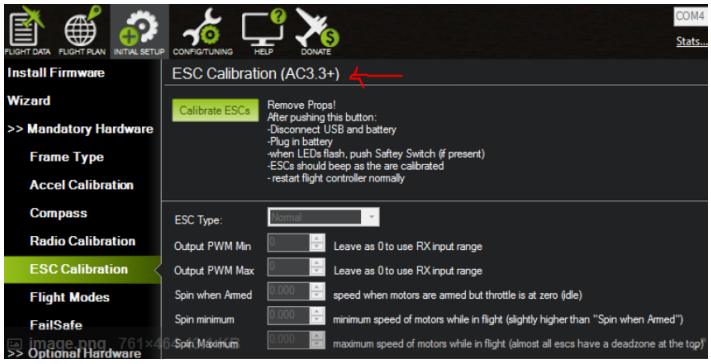


Figure 7.19: ESC's calibration

RPM sensor set up

In order to configure the RPM sensors, go to Configure Optional hardware. First the board needs to be configured to allow PWM pins to be set for General Purpose Input/Outputs (GPIOs). This is done using the parameter BRD_PWM_COUNT. Reduce the PWM count to free up a pin to be used for GPIO. On non-Pixhawk boards the PWM count will include all PWM outputs. On Pixhawk boards this parameter only affects AUX pins. Write the parameter and reboot the autopilot. As the Pixhawk board has 6 auxiliary ports, 4 of them are used as GPIOs, one for each motor, $6 - 4 = 2$ is the number of free pins in the BRD_PWM_COUNT parameter.

RPM2_TYPE. Set up to 2 if you are using the AUX ports on pixhawk. If using a PWM port on a pixhawk or a different board set RPM_TYPE to 1.

RP2_SCALING parameter. The scaling value is a function of the number of poles in the motor and should be the reciprocal of the number of poles. E.g. A 14 pole motor will need a scaling value of 0.071428.

RPM_PIN. Is the pin number assigned to be used as GPIO. For our project, the next parameters were configured as shown in the table 7.1.

For further explanation, an extensive documentation can be found from the Ardupilot Community [39].

Programming an Autonomous Mission

Mission Planner allows planning the path of the vehicle by selecting points or "waypoints", and other mission-related options. Mission Planner provides a list of the commands appropriate for the current vehicle type, and adds column headings for the parameters that needed to be set up by the user. These include navigation commands to travel to waypoints and loiter in the vicinity, DO commands to execute specific actions (for example taking pictures), and condition commands that can control when DO commands are able to run. Latitude and Longitude are entered by clicking on the map. Altitude is relative to the launch altitude/home position, so if it is set to 100m, for example, it will fly 100m above the pilot.

Default Alt is the default altitude when entering new waypoints.

Table 7.1: RPM sensor Parameters Set up

Parameter	Value
BRD_PWM_COUNT	2
RPM2_TYPE	1
RPM2SCALING	0.071428
RPM_PIN	55
RPM2_PIN	54
RPM3_PIN	53
RPM4_PIN	52

Verify height means that the Mission Planner will use Google Earth topology data to adjust your desired altitude at each waypoint to reflect the height of the ground beneath. Create a mission by clicking on the map the locations of the way points, see for example Fig. 7.20, then select Write and it will be sent to board and saved in EEPROM. To confirm that the mission was saved, select Read. You can save multiple mission files to your local hard drive by selecting Save WP File or read in files with Load WP File.

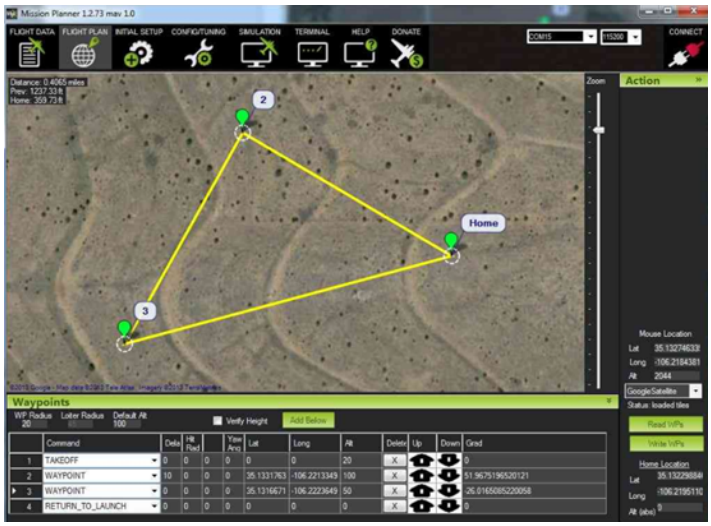


Figure 7.20: Autonomous Mission executing a triangle path.

For our experimental validation, we have programmed an autonomous mission of a circular flight 7.20 and an 8-shape flight.

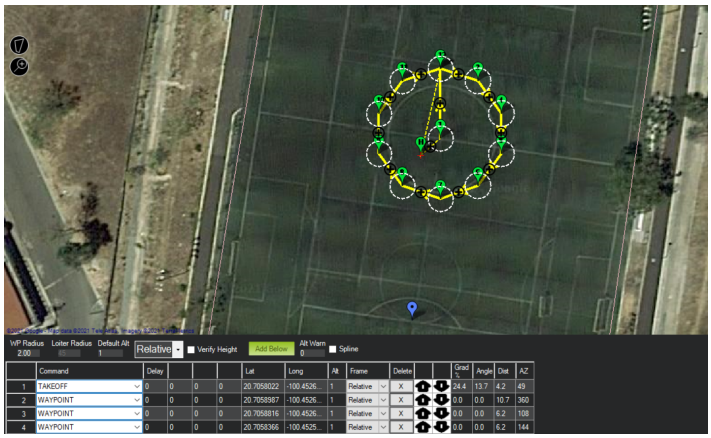


Figure 7.21: Autonomous Mission executing a circular trajectory.

7.2 Parameter identification

Before the experimental validation, we first need to know the physical parameters of our quadrotor. These parameters are: the mass of the quadrotor m, the arm length d, its inertia matrix I, the thrust coefficient c_T and the torque coefficient c_M. The physical parameters of the UAV were determined experimentally. The mass of the body quadrotor m was measured using a digital balance, the distance between the motor and the center of the quadrotor d was measured using a ruler. The principal moment of inertia was estimated using a CAD model, a theoretical model and through an algorithm. The thrust coefficient and the torque coefficient were determined by static tests in the laboratory. A brief description of the experiments carried out in the laboratory to identify the physical parameters of the system is presented.

7.2.1 Moment of inertia

The moment of inertia of a rigid body is expressed as:

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

Here $I_{xy} = I_{yx}$, $I_{xz} = I_{yz}$ and $I_{yz} = I_{zy}$. Assuming that the quadrotor has a symmetric mass distribution, then $I_{xy} = I_{xz} = I_{yz} = 0$ and the inertia matrix is simplified:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{7.1}$$

where I_{xx}, I_{yy} and I_{zz} are the principal moments of inertia. Principal moments of inertia were determined by two methods: computer-aided design (CAD) software and analytical method using the parallel axis theorem for homogeneous solids.

7.2.1.1 Quadrotor CAD model

A CAD model design of the quadrotor was used to determine the moment of inertia using software tools, where all parts of the quadrotor were modeled and the masses and dimensions of each component were known.

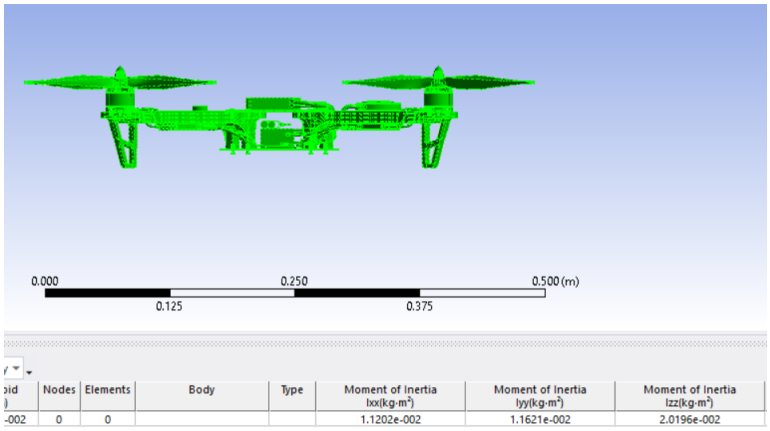


Figure 7.22: Quadrotor CAD model

$$I = \begin{bmatrix} 0.011202 & 0 & 0 \\ 0 & 0.011621 & 0 \\ 0 & 0 & 0.020196 \end{bmatrix} kgm^2$$

7.2.1.2 Analytical method

To determine the moment of inertia, the quad rotor vehicle was separated into different components. The model of each component was represented by a simplified geometric shape of constant internal density. The weight of each component was then measured. The moment of inertia of each component about the axes of the vehicle is determined using the parallel axis theorem. The total moment of inertia matrix of the vehicle is the sum of the inertias for every component about each axis. The components that need to be modeled are the motor, the ESC, arms, and the central hub. The parallel axis theorem.

$$I = \begin{bmatrix} 0.0115 & 0 & 0 \\ 0 & 0.0119 & 0 \\ 0 & 0 & 0.0126 \end{bmatrix} kgm^2$$

7.2.2 Determination of the motor-propeller thrust coefficient

The steady-state thrust generated by a hovering rotor (i.e., a rotor that is not translating horizontally or vertically) in free air may be modeled using momentum theory as:

$$T_i = C_T \rho A_{r_i} r_i^2 \omega_i^2$$

where for the rotor i, A_{r_i} is the area of the rotor disk, r_i is the radius, ω_i^2 is the angular velocity of the motor, and C_T is the thrust coefficient that depends on the geometry and profile of the rotor and ρ is the density of the air. A simple lumped-parameter model is

$$T_i = c_T \omega_i^2 \tag{7.2}$$

where $c_T = C_T \rho A_{r_i} r_i^2 > 0$ is modeled as a constant that can be determined by static thrust test. Identifying the thrust constant experimentally has the advantage that it will also naturally incorporate the effect of drag on the airframe induced by the rotor flow [29]. our project, we used a Racerstar Brushless Motor Thrust Stand V3 for static thrust test, it is a High precision thrust sensor with precise digital monitoring for speed, voltage and power parameters. The set up is shown in Fig. 7.23. According with (7.2), we can plot the thrust force vs. motor-propeller speed in order to determine constant c_T. The result of this experiment is shown in Fig. 7.24 .

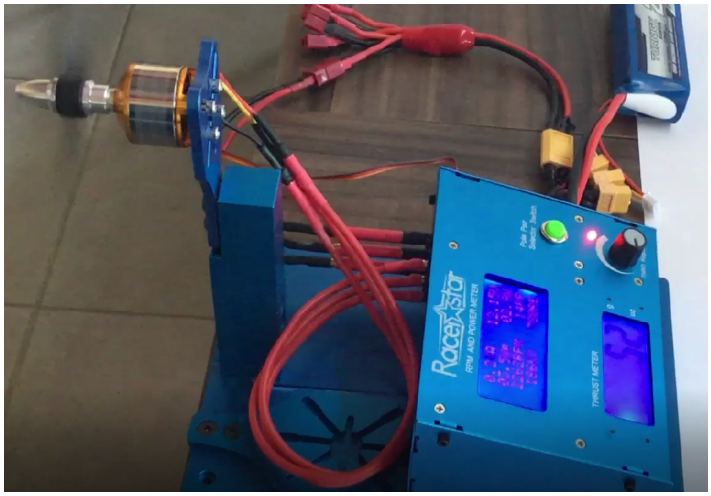


Figure 7.23: Test bench for motor-propeller characterization to measure thrust force and motor speeds.

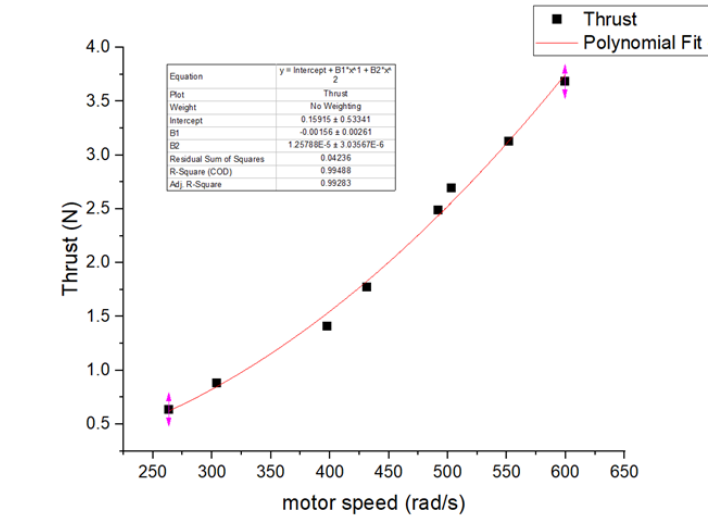


Figure 7.24: Thrust force vs. motor-propeller speed.

7.2.3 Determination of the motor-propeller torque coefficient

The reaction torque (due to rotor drag) acting on the airframe generated by a hovering rotor in free air may be modeled as

$$M_i = c_M \omega_i^2 \tag{7.3}$$

where M_i is the reaction torque of the ith propeller acting of the fuselage, $c_M = (1/4\pi^2) * \rho D_p^5 C_M$, D_p denotes the diameter of the propeller, ρ is the air density and C_M is the torque coefficient and ω_i is the motor speed. Similarly, as in the previous case, we have from equation (7.3) that if reaction torque and motor speed are measured, then c_M can be determined, see Fig. 7.25.

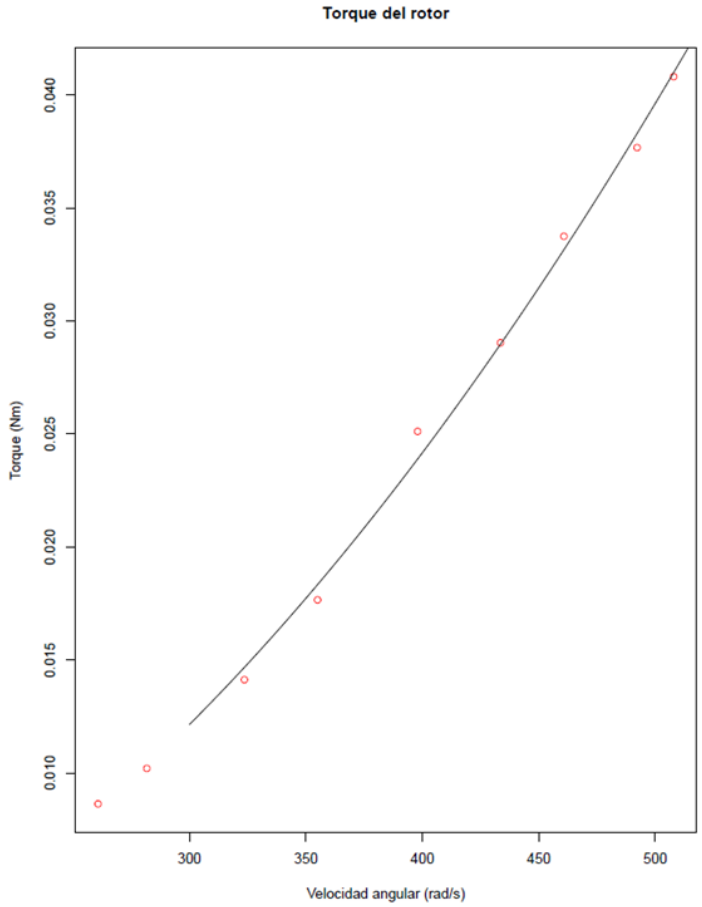


Figure 7.25: Torque vs. motor-propeller speed.

The physical parameters of the quadrotor are summarized in Tab. 7.2 above.

Table 7.2: Quadrotor physical parameters

Symbol	Physical property	Value	Units
m	quadrotor mass	1.07	kg
d	distance	0.23	m
I_{xx}	Moment of inertia x-axis	0.0112	kgm^2
I_{yy}	Moment of inertia y-axis	0.0116	kgm^2
I_{zz}	Moment of inertia z-axis	0.0201	kgm^2
c_T	Thrust coefficient	1.257×10^{-05}	$N/(rad/s)^2$
c_M	Moment coefficient	1.717×10^{-07}	$Nm/(rad/s)^2$

7.3 Experimental results

In this section, the experimental results regarding the performance of the angular velocity (3.15), (3.16), (3.5), attitude observer (4.1), (4.2) and translational observer (5.4)- (5.6), are presented and discussed.

The experiment was carried out by programming an autonomous mission so that the quadrotor followed a regular path. Data from the IMU sensor were collected from the flight and processed in an outdoor environment for experimental verification. The reference vector for acceleration is taken as $[0, 0, -9.8]^T\ m/s^2$, or as a unit vector $[0, 0, -1]^T$. The normalized vector of the Earth's magnetic field was taken as: $[0.6626, 0.0544, 0.7469]^T$, which corresponds to the value in Juriquilla, Querétaro, Mexico, place where the experiments were carried out, according to [26] which uses the World Magnetic Model to calculate the Magnetic Field in a certain region of space knowing its latitude, longitude and altitude. The initial states were taken as: $q(0) = [0.9071, 0.0165, -0.0075, 0.4205]^T$, $\hat{q}(0) = [1, 0, 0, 0]^T$, $\bar{\omega}(0) = [0.05, 0.05, 0.05]^T$ and the parameters used for implementation are summarized in the Tab. 7.3.

Table 7.3: Data used in the implementation of the complete algorithm in the Quad

	Value 1	Value 2
Reference vectors	$r_1 = [0, 0, -1]^T$	$r_2 = [0.6626, 0.0544, 0.7469]^T$
Parameters		
Gain matrix	$\Lambda_1 = diag[0.25, 0.25, 0.25]$	$\Lambda_2 = diag[0.25, 0.25, 0.25]$
sensor's weights	$k_1 = 5.0$	$k_2 = 5.0$
Filter's gain	$\gamma_f = 5.0$	
Gain	$\kappa_p = 1.0$	

The results of the two flights were analyzed and are presented below.

7.3.1 Circular path flight

A circular trajectory of radius of 15 m during approximately 80 with take-off and landing s was executed by the quadrotor. It started its motion very close to the center of the circle. It takes off and heads towards the first point of the circle. Then, the flight followed a circular trajectory Fig. 7.26. Because the drone rotates around the center of the circle, this experiment is suitable to analyze mainly the z component of the angular velocity and the evolution of yaw along the trajectory, because they are the significant components for the type of motion so that roll and pitch are kept almost constant and small.

The GPS sensor provides the position measurement of the quadrotor in terms of latitude (degrees), longitude (degrees) and altitude (meters) in the Earth Centered Earth Fixed frame, which were transformed to the north-east-down (NED) coordinates into meters. The following

figure 7.26 shows the trajectory executed by the quadrotor in the real flight.

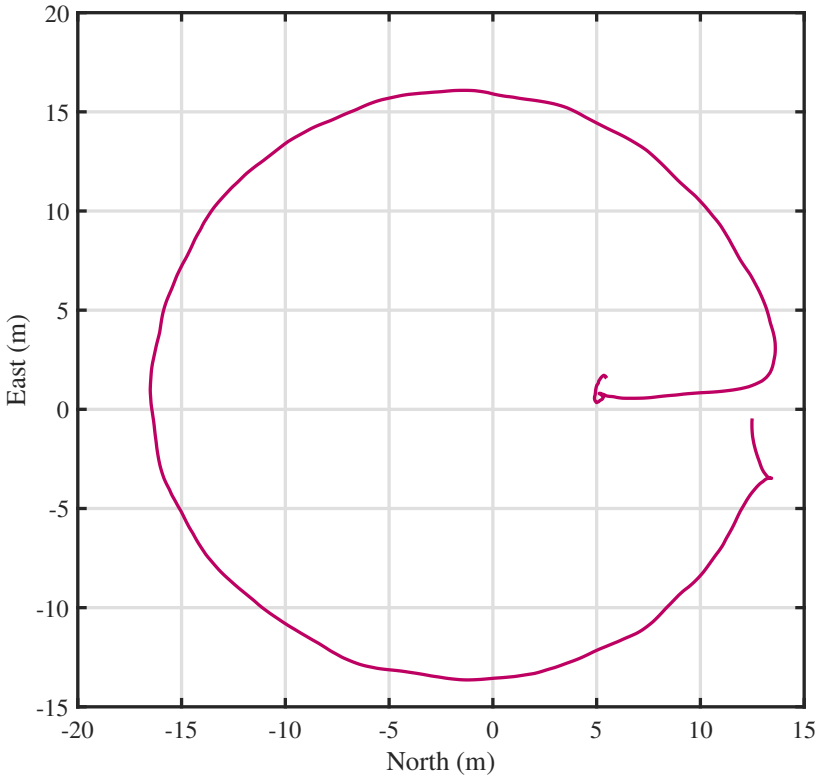


Figure 7.26: Circular flight of the quadrotor in NED frame.

The origin of the NED frame was chosen in the center of the circular trajectory. The true linear velocity was obtained by time differentiation of the GPS signal and then compared with the position and velocity estimated by the observer (5.4)- (5.6).

As discussed previously, the choice of gains k_1 and k_2 should be in accordance with the reliability of each sensor. In the experiments where the flying vehicles are operated in aggressive maneuvers susceptible to high accelerations, it is advisable to rely on the magnetometer rather than the accelerometer; therefore, give a higher gain value to the magnetometer or use a third sensor such as GPS or INS. On the other hand, if the experiment can now be affected by magnetic disturbances or the sensors are near magnets, then it is advisable to rely on the accelerometer rather than the magnetometer. In our experiment, the quadrotor flies smoothly during the circular motion; therefore, it is not subject to abrupt changes in linear acceleration during this period. For this reason, the same reliability was given to the accelerometer as for the magnetometer.

The calibrated accelerometer and magnetometer measurements from the IMU onboard the quadrotor are shown in Fig. 7.27 and Fig. 7.28, respectively. This set of data are the input for angular velocity - attitude observer algorithm.

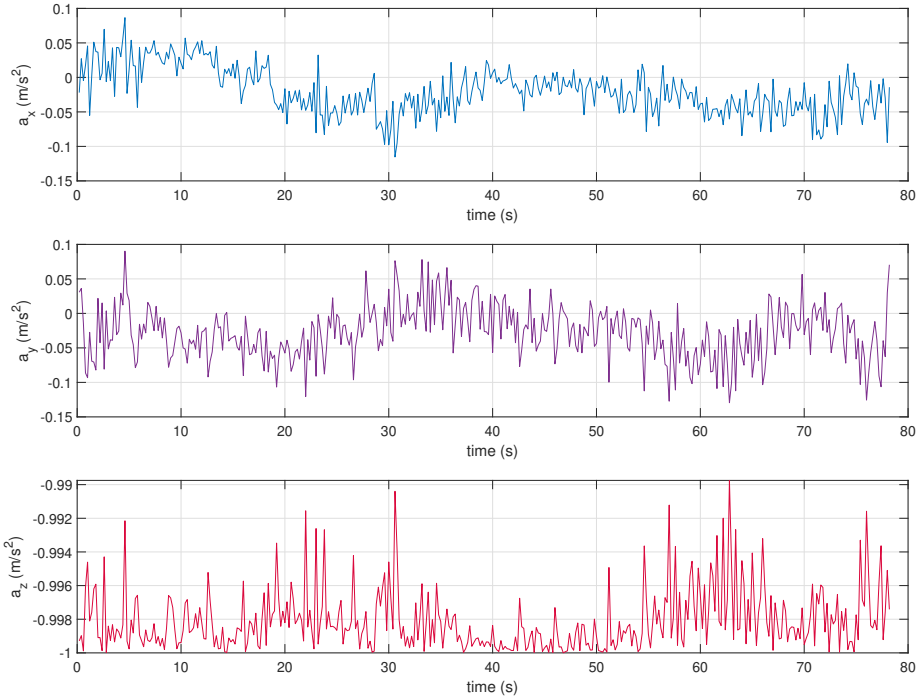


Figure 7.27: Normalized accelerometer measurements (Circular flight).

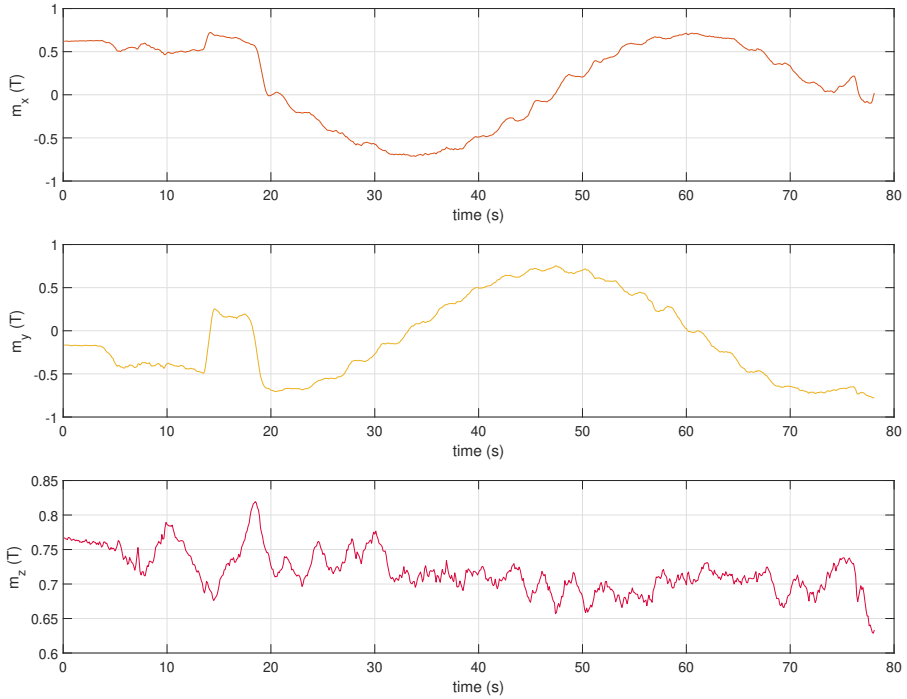


Figure 7.28: Magnetometer measurements (Circular flight).

As can be seen from (3.15), the implementable version of the observer, the angular velocity can be estimated using at least two body vector measurements and the total torque. To calculate the torque that acts on the quadrotor, the speed of each motor is required. Data from the RPM sensors are shown in Fig. 7.29.

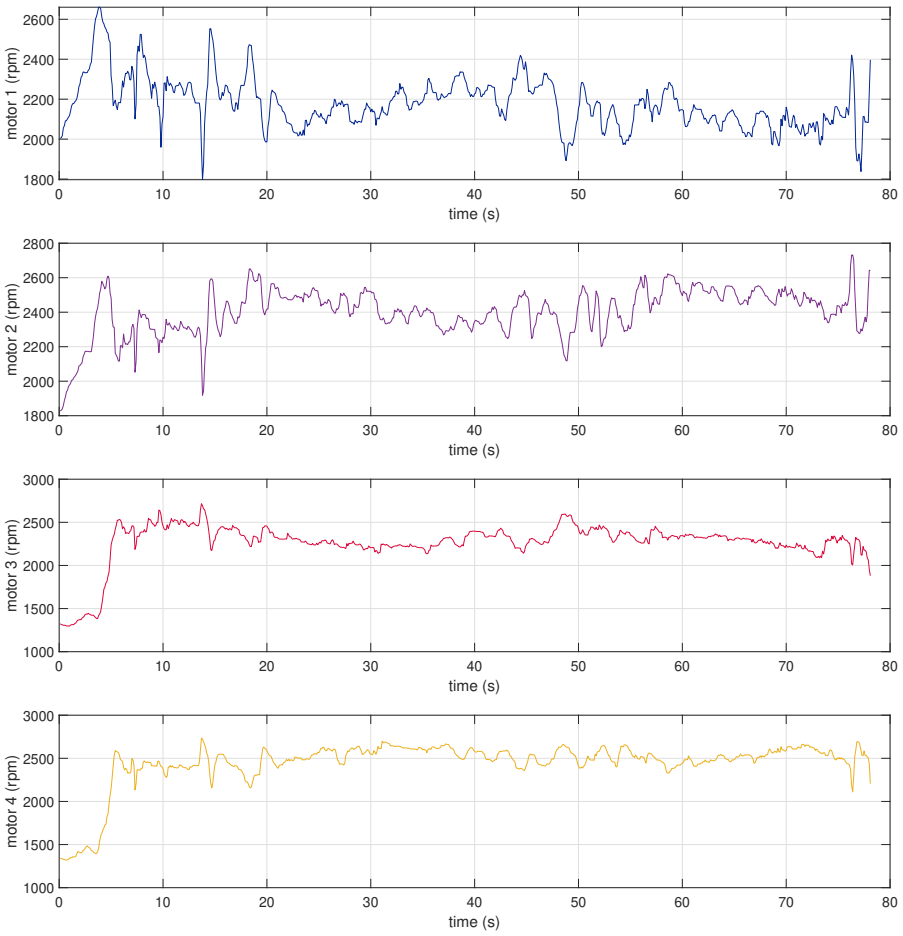


Figure 7.29: Motor angular speeds (Circular flight).

The performance of the angular velocity observer is illustrated in Fig. 7.30. The results of the experiment show that there are a good resemblance between the real angular velocity and the estimated angular velocity. In order to show the performance of the attitude observer, we represent the orientation of the quadrotor in Euler angles, and compared the true values ϕ , θ, ψ with the estimated values $\hat{\phi}$, $\hat{\theta}$ $\hat{\psi}$ in Fig. 7.31.

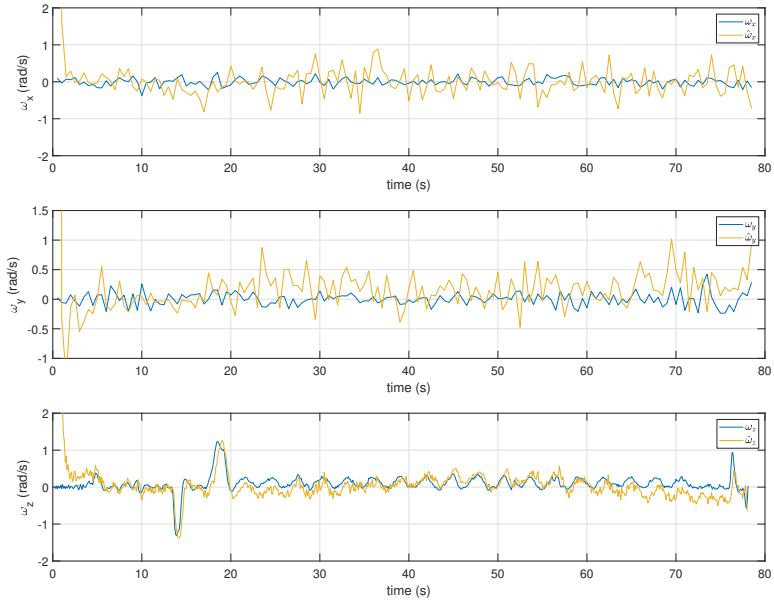


Figure 7.30: True and estimated angular velocity of the quadrotor (Circular flight).

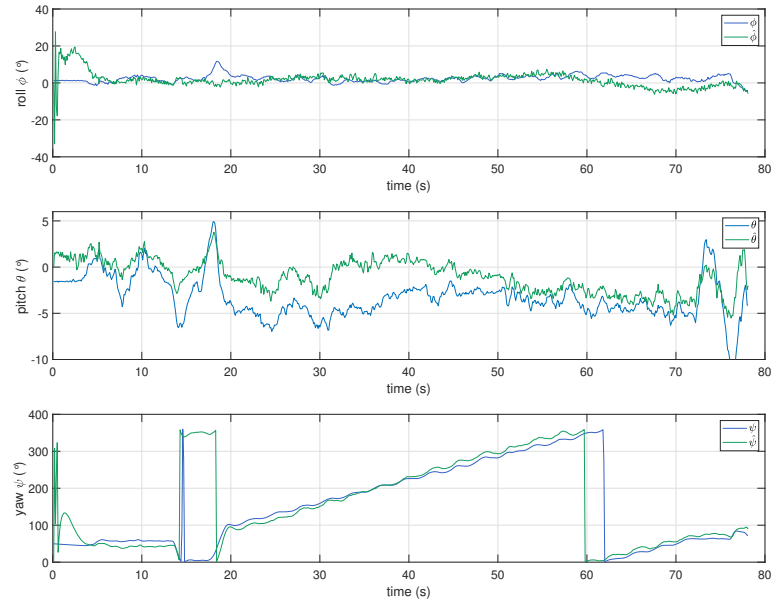


Figure 7.31: True and estimated attitude in Euler angles representation of the quadrotor (Circular flight).

Regarding the translation state observer, we show in Fig. 7.32 the comparison between the real linear velocity v with the estimated linear velocity $\hat{v}$ and in Fig.7.33 the comparison between the real position p with the estimated position $\hat{p}$.

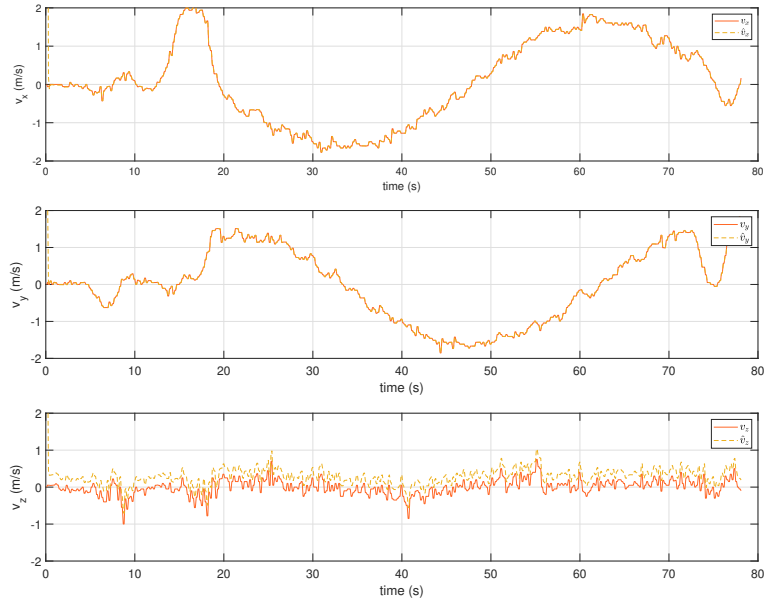


Figure 7.32: True and estimated linear velocity (Circular flight).

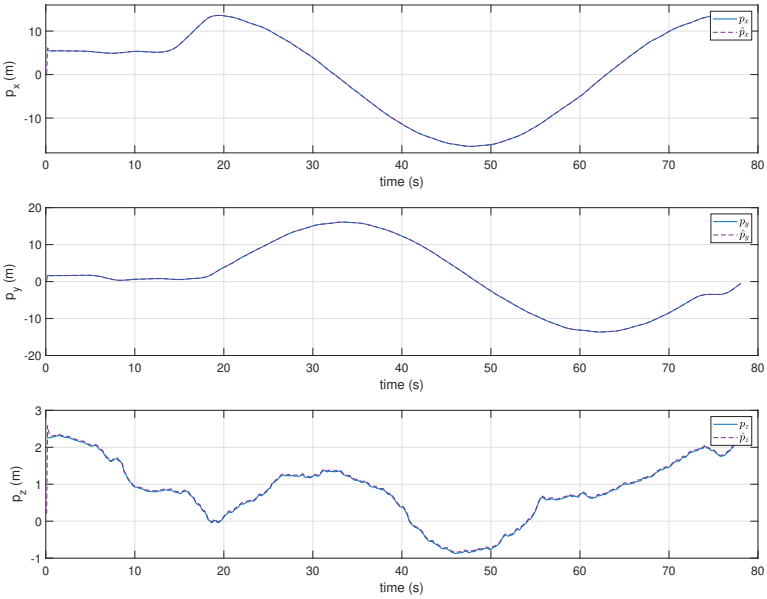


Figure 7.33: True and estimated position (Circular flight).

We show the error for the overall observer in Fig. 7.34. Note from the figure Fig.6.3 that simulation results coincide with the results obtained in the experiments Fig. 7.34.

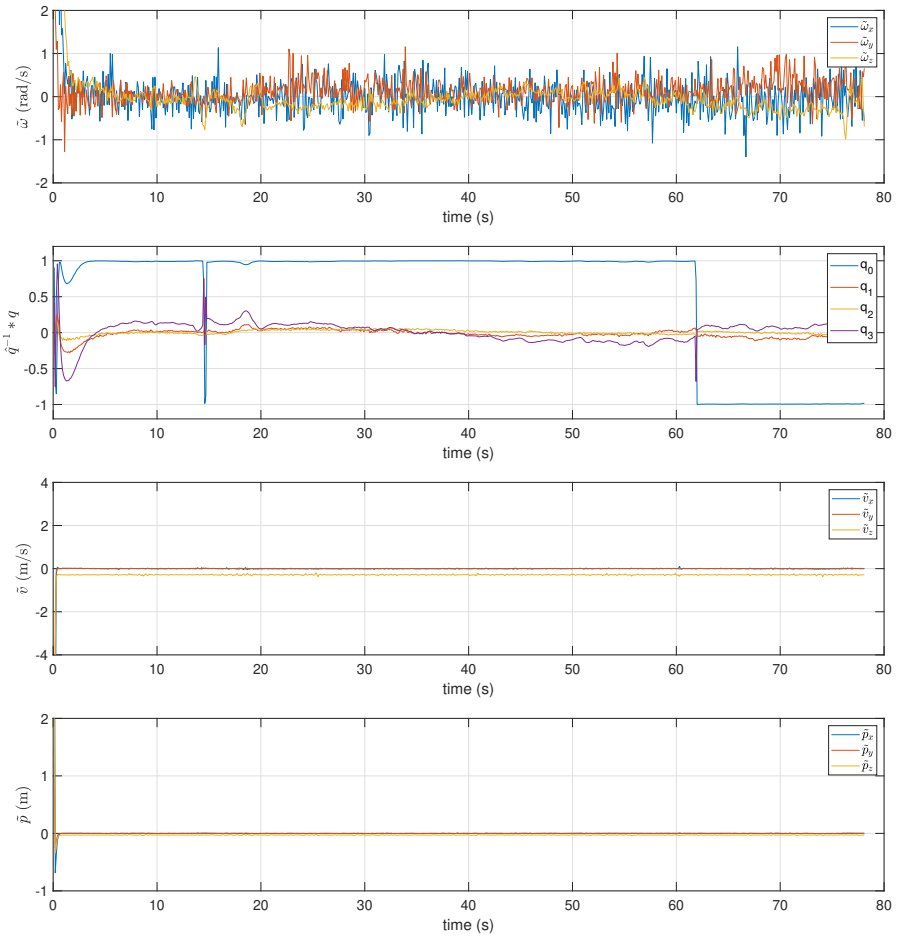


Figure 7.34: Real flight: (Top-down) angular velocity error $\tilde{\omega}$, attitude error $\hat{q}^{-1} \otimes q$, linear velocity error $\tilde{v}$ and position error $\tilde{p}$ (Circular flight).

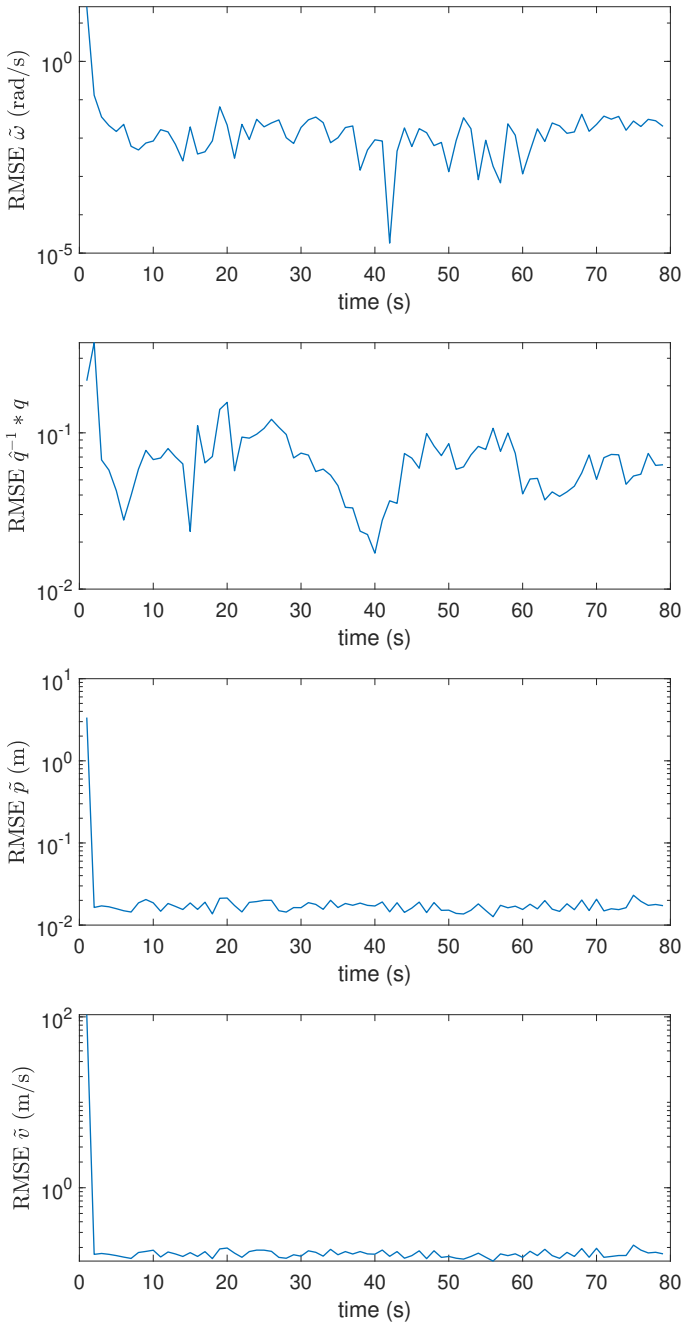


Figure 7.35: Real flight: (Top-down) Root Mean Square estimate for angular velocity error $\tilde{\omega}$, attitude error $\hat{q}^{-1} * q$, position error $\tilde{p}$ and linear velocity error $\tilde{v}$.

Chapter 8

Conclusions

In this thesis, the problem of design a navigation observer for autonomous vehicles such as quadrotors has been addressed. A new angular velocity-attitude observer has been proposed first, within a framework where neither the angular velocity from gyros nor the attitude is needed to perform the estimation, only requiring vector measurements available from a sensor, such as accelerometer and magnetometer. The proposed angular velocity observer is then combined with Explicit Complementary Filter (ECF) for attitude estimation. The almost global asymptotic convergence of the proposed observer to the true state was established using the Lyapunov analysis. The resulting cascaded structure of the observer decouples the design of the angular velocity observer from the attitude observer, facilitates the convergence of the proposed observer and its implementation. Moreover, since the proposed observer uses only vector measurements, it may represent a backup solution for autonomous vehicles in the event of gyroscope failures. A translational state observer is developed at last which is globally exponentially stable if a mild condition on the attitude estimation is fulfilled.

Through simulation, the behavior of the navigation observer was validated in ideal and practical situations, where the measurements noise and the parameter uncertainty of the inertia matrix were present. The results were in good agreement with the theoretical analysis.

In addition, the proposed observer was validated on an experimental platform built in the Mechatronics Laboratory based on a quadrotor. The design and implementation of the platform were described in detail. The quadrotor ran a circular flight path and an eight-shape flight. The experimental results showed that there is a good correspondence between the experimental results and the simulation results.

Future works may include designing an adaptive observer to estimate the inertia matrix of the autonomous vehicle and incorporate the proposed navigation observer into a control loop.

Appendix A

A.1 AP_RPM Library

Ardupilot firmware supports the use of numerous types of RPM sensors. In principle, any rpm sensor that outputs a PWM signal or a step-change in voltage as a function of RPM can be used by ArduPilot. All RPM sensors use the AP_RPM library because the rpm sensor is plugged into an AUX port on a Pixhawk, so we need to ensure that the AUX port is configured as a GPIO pin to receive a signal input. Also, Ardupilot needs to know which pin the RPM sensor is connected to. The RPM library in ArduPilot monitors the voltage of the designated signal pin. With the latest version firmware of Ardupilot a maximum of two RPM sensors can be used. For our project, we need to active four rpm sensors, one for each motor. This is the reason why we require to customize the firmware, particularly the RPM library.

The following lines of programming code are those that were added to the firmware and implemented on the Pixhawk board. Basically, the tasks the code performs are:

- To active four pins, AUX pins as GPIO (Instead of two).
- To read and store the RPM sensor signals.
- To send the messages to the Full Parameters list, so that the user can configure the parameters related with RPM sensors through Mission Planner.
- To log RPM sensors readings into a microsd card.

libraries/AP_RPM/AP_RPM.h

```
/*
   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */
#pragma once

#include <AP_Common/AP_Common.h>
#include <AP_HAL/AP_HAL.h>
#include <AP_Param/AP_Param.h>
#include <AP_Math/AP_Math.h>

// Maximum number of RPM measurement instances available on this platform
#define RPM_MAX_INSTANCES 4

class AP_RPM_Backend;

class AP_RPM
{
    friend class AP_RPM_Backend;

public:
    AP_RPM();

    /* Do not allow copies */
    AP_RPM(const AP_RPM &other) = delete;
    AP_RPM &operator=(const AP_RPM&) = delete;

    // RPM driver types
```

```
    enum RPM_Type {
        RPM_TYPE_NONE    = 0,
        RPM_TYPE_PWM     = 1,
        RPM_TYPE_PIN     = 2,
        RPM_TYPE_EFI     = 3,
        RPM_TYPE_HNTCH   = 4,
#if CONFIG_HAL_BOARD == HAL_BOARD_SITL
        RPM_TYPE_SITL    = 10,
#endif
    };

    // The RPM_State structure is filled in by the backend driver
    struct RPM_State {
        uint8_t                 instance;        // the instance number of
    this RPM
        float                   rate_rpm;        // measured rate in revs
    per minute
        uint32_t                last_reading_ms; // time of last reading
        float                   signal_quality;  // synthetic quality
    metric
    };

    // parameters for each instance
    AP_Int8  _type[RPM_MAX_INSTANCES];
    AP_Int8  _pin[RPM_MAX_INSTANCES];
    AP_Float _scaling[RPM_MAX_INSTANCES];
    AP_Float _maximum;
    AP_Float _minimum;
    AP_Float _quality_min;

    static const struct AP_Param::GroupInfo var_info[];

    // Return the number of rpm sensor instances
    uint8_t num_sensors(void) const {
        return num_instances;
    }
```

```
    // detect and initialise any available rpm sensors
    void init(void);

    // update state of all rpm sensors. Should be called from main loop
    void update(void);

    /*
      return RPM for a sensor. Return -1 if not healthy
     */
    bool get_rpm(uint8_t instance, float &rpm_value) const;

    /*
      return signal quality for a sensor.
     */
    float get_signal_quality(uint8_t instance) const {
        return state[instance].signal_quality;
    }

    bool healthy(uint8_t instance) const;

    bool enabled(uint8_t instance) const;

    static AP_RPM *get_singleton() { return _singleton; }

private:
    static AP_RPM *_singleton;

    RPM_State state[RPM_MAX_INSTANCES];
    AP_RPM_Backend *drivers[RPM_MAX_INSTANCES];
    uint8_t num_instances:3;  // This is a bit field not =

    void detect_instance(uint8_t instance);
};

namespace AP {
    AP_RPM *rpm();
```

```
};
```

libraries/AP_RPM/AP_RPM.cpp

```
/*
   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
*/
#include "AP_RPM.h"
#include "RPM_Pin.h"
#include "RPM_SITL.h"
#include "RPM_EFI.h"
#include "RPM_HarmonicNotch.h"

extern const AP_HAL::HAL& hal;

// table of user settable parameters
const AP_Param::GroupInfo AP_RPM::var_info[] = {
    // Param: _TYPE
    // DisplayName: RPM type
    // Description: What type of RPM sensor is connected
    // Values: 0:None,1:PWM,2:AUXPIN,3:EFI,4:Harmonic Notch
    // User: Standard
    AP_GROUPINFO("_TYPE",    0, AP_RPM, _type[0], 0),

    // Param: _SCALING
    // DisplayName: RPM scaling
    // Description: Scaling factor between sensor reading and RPM.
    // Increment: 0.001
```

```
    // User: Standard
    AP_GROUPINFO("_SCALING", 1, AP_RPM, _scaling[0], 1.0f),

    // Param: _MAX
    // DisplayName: Maximum RPM
    // Description: Maximum RPM to report
    // Increment: 1
    // User: Standard
    AP_GROUPINFO("_MAX", 2, AP_RPM, _maximum, 100000),

    // Param: _MIN
    // DisplayName: Minimum RPM
    // Description: Minimum RPM to report
    // Increment: 1
    // User: Standard
    AP_GROUPINFO("_MIN", 3, AP_RPM, _minimum, 10),

    // Param: _MIN_QUAL
    // DisplayName: Minimum Quality
    // Description: Minimum data quality to be used
    // Increment: 0.1
    // User: Advanced
    AP_GROUPINFO("_MIN_QUAL", 4, AP_RPM, _quality_min, 0.5),

    // Param: _PIN
    // DisplayName: Input pin number
    // Description: Which pin to use
    // Values: -1:Disabled,50:PixhawkAUX1,51:PixhawkAUX2,52:PixhawkAUX3
    ,53:PixhawkAUX4,54:PixhawkAUX5,55:PixhawkAUX6
    // User: Standard
    AP_GROUPINFO("_PIN",     5, AP_RPM, _pin[0], 54),

#if RPM_MAX_INSTANCES > 1
    // Param: 2_TYPE
    // DisplayName: Second RPM type
    // Description: What type of RPM sensor is connected
    // Values: 0:None,1:PWM,2:AUXPIN,3:EFI,4:Harmonic Notch
```

```
    // User: Advanced
    AP_GROUPINFO("2_TYPE",    10, AP_RPM, _type[1], 0),

    // Param: 2_SCALING
    // DisplayName: RPM scaling
    // Description: Scaling factor between sensor reading and RPM.
    // Increment: 0.001
    // User: Advanced
    AP_GROUPINFO("2_SCALING", 11, AP_RPM, _scaling[1], 1.0f),

    // Param: 2_PIN
    // DisplayName: RPM2 input pin number
    // Description: Which pin to use
    // Values: -1:Disabled,50:PixhawkAUX1,51:PixhawkAUX2,52:PixhawkAUX3
    ,53:PixhawkAUX4,54:PixhawkAUX5,55:PixhawkAUX6
    // User: Standard
    AP_GROUPINFO("2_PIN", 12, AP_RPM, _pin[1], -1),

    // Param: 3_TYPE
    // DisplayName: Third RPM type
    // Description: What type of RPM sensor is connected
    // Values: 0:None,1:PWM,2:AUXPIN,3:EFI,4:Harmonic Notch
    // User: Advanced
    AP_GROUPINFO("3_TYPE", 13, AP_RPM, _type[2], 0),

    // Param: 3_SCALING
    // DisplayName: RPM scaling
    // Description: Scaling factor between sensor reading and RPM.
    // Increment: 0.001
    // User: Advanced
    AP_GROUPINFO("3_SCALING", 14, AP_RPM, _scaling[2], 1.0f),

    // Param: 3_PIN
    // DisplayName: RPM3 input pin number
    // Description: Which pin to use
    // Values: -1:Disabled,50:PixhawkAUX1,51:PixhawkAUX2,52:PixhawkAUX3
    ,53:PixhawkAUX4,54:PixhawkAUX5,55:PixhawkAUX6
```

```
    // User: Standard
    AP_GROUPINFO("3_PIN", 15, AP_RPM, _pin[2], -1),

    // Param: 4_TYPE
    // DisplayName: Fourth RPM type
    // Description: What type of RPM sensor is connected
    // Values: 0:None,1:PWM,2:AUXPIN,3:EFI,4:Harmonic Notch
    // User: Advanced
    AP_GROUPINFO("4_TYPE",    16, AP_RPM, _type[3], 0),

    // Param: 4_SCALING
    // DisplayName: RPM scaling
    // Description: Scaling factor between sensor reading and RPM.
    // Increment: 0.001
    // User: Advanced
    AP_GROUPINFO("4_SCALING", 17, AP_RPM, _scaling[3], 1.0f),

    // Param: 4_PIN
    // DisplayName: RPM4 input pin number
    // Description: Which pin to use
    // Values: -1:Disabled,50:PixhawkAUX1,51:PixhawkAUX2,52:PixhawkAUX3
    ,53:PixhawkAUX4,54:PixhawkAUX5,55:PixhawkAUX6
    // User: Standard
    AP_GROUPINFO("4_PIN",    18, AP_RPM, _pin[3], -1),
#endif

    AP_GROUPEND
};

AP_RPM::AP_RPM(void)
{
    AP_Param::setup_object_defaults(this, var_info);

    if (_singleton != nullptr) {
        AP_HAL::panic("AP_RPM must be singleton");
    }
    _singleton = this;
```

```
}


/*
  initialise the AP_RPM class.
 */
void AP_RPM::init(void)
{
    if (num_instances != 0) {
        // init called a 2nd time?
        return;
    }
    for (uint8_t i=0; i<RPM_MAX_INSTANCES; i++) {
        uint8_t type = _type[i];
#if CONFIG_HAL_BOARD != HAL_BOARD_SITL
        if (type == RPM_TYPE_PWM) {
            // PWM option same as PIN option, for upgrade
            type = RPM_TYPE_PIN;
        }
        if (type == RPM_TYPE_PIN) {
            drivers[i] = new AP_RPM_Pin(*this, i, state[i]);
        }
#endif
#if EFI_ENABLED
        if (type == RPM_TYPE_EFI) {
            drivers[i] = new AP_RPM_EFI(*this, i, state[i]);
        }
#endif
        // include harmonic notch last
        // this makes whatever process is driving the dynamic notch appear
    as an RPM value
        if (type == RPM_TYPE_HNTCH) {
            drivers[i] = new AP_RPM_HarmonicNotch(*this, i, state[i]);
        }
#if CONFIG_HAL_BOARD == HAL_BOARD_SITL
        if (type == RPM_TYPE_SITL) {
            drivers[i] = new AP_RPM_SITL(*this, i, state[i]);
        }
```

```
#endif
        if (drivers[i] != nullptr) {
            // we loaded a driver for this instance, so it must be
            // present (although it may not be healthy)
            num_instances = i+1; // num_instances is a high-water-mark
        }
    }
}

/*
  update RPM state for all instances. This should be called by main loop
 */
void AP_RPM::update(void)
{
    for (uint8_t i=0; i<num_instances; i++) {
        if (drivers[i] != nullptr) {
            if (_type[i] == RPM_TYPE_NONE) {
                // allow user to disable an RPM sensor at runtime and
    force it to re-learn the quality if re-enabled.
                state[i].signal_quality = 0;
                continue;
            }

            drivers[i]->update();
        }
    }
}

/*
  check if an instance is healthy
 */
bool AP_RPM::healthy(uint8_t instance) const
{
    if (instance >= num_instances || _type[instance] == RPM_TYPE_NONE) {
        return false;
    }
```

```
    // check that data quality is above minimum required
    if (state[instance].signal_quality < _quality_min.get()) {
        return false;
    }

    return true;
}

/*
  check if an instance is activated
 */
bool AP_RPM::enabled(uint8_t instance) const
{
    if (instance >= num_instances) {
        return false;
    }
    // if no sensor type is selected, the sensor is not activated.
    if (_type[instance] == RPM_TYPE_NONE) {
        return false;
    }
    return true;
}

/*
  get RPM value, return true on success
 */
bool AP_RPM::get_rpm(uint8_t instance, float &rpm_value) const
{
    if (!healthy(instance)) {
        return false;
    }
    rpm_value = state[instance].rate_rpm;
    return true;
}

// singleton instance
AP_RPM *AP_RPM::_singleton;
```

```
namespace AP {

AP_RPM *rpm()
{
    return AP_RPM::get_singleton();
}

}
```

libraries/AP_RPM/AP_Pin.cpp

```
/*
   This program is free software: you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation, either version 3 of the License, or
   (at your option) any later version.
   This program is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.
   You should have received a copy of the GNU General Public License
   along with this program.  If not, see <http://www.gnu.org/licenses/>.
 */

#include <AP_HAL/AP_HAL.h>
#include "RPM_Pin.h"

#include <AP_HAL/GPIO.h>
#include <GCS_MAVLink/GCS.h>

extern const AP_HAL::HAL& hal;
AP_RPM_Pin::IrqState AP_RPM_Pin::irq_state[RPM_MAX_INSTANCES];

/*
   open the sensor in constructor
```

```
*/
AP_RPM_Pin::AP_RPM_Pin(AP_RPM &_ap_rpm, uint8_t instance, AP_RPM::
    RPM_State &_state) :
  AP_RPM_Backend(_ap_rpm, instance, _state)
{
}

/*
  handle interrupt on an instance
 */
void AP_RPM_Pin::irq_handler(uint8_t pin, bool pin_state, uint32_t
    timestamp)
{
    const uint32_t dt = timestamp - irq_state[state.instance].
    last_pulse_us;
    irq_state[state.instance].last_pulse_us = timestamp;
    // we don't accept pulses less than 100us. Using an irq for such
    // high RPM is too inaccurate, and it is probably just bounce of
    // the signal which we should ignore
    if (dt > 100 && dt < 1000*1000) {
        irq_state[state.instance].dt_sum += dt;
        irq_state[state.instance].dt_count++;
    }
}

void AP_RPM_Pin::update(void)
{
    if (last_pin != get_pin()) {
        // detach from last pin
        if (last_pin != (uint8_t)-1 &&
            !hal.gpio->detach_interrupt(last_pin)) {
            gcs().send_text(MAV_SEVERITY_WARNING, "RPM: Failed to detach
    from pin %u", last_pin);
            // ignore this failure or the user may be stuck
        }
        irq_state[state.instance].dt_count = 0;
        irq_state[state.instance].dt_sum = 0;
```

```
        // attach to new pin
        last_pin = get_pin();
        if (last_pin) {
            hal.gpio->pinMode(last_pin, HAL_GPIO_INPUT);
            if (!hal.gpio->attach_interrupt(
                    last_pin,
                    FUNCTOR_BIND_MEMBER(&AP_RPM_Pin::irq_handler, void,
uint8_t, bool, uint32_t),
                    AP_HAL::GPIO::INTERRUPT_RISING)) {
                gcs().send_text(MAV_SEVERITY_WARNING, "RPM: Failed to
attach to pin %u", last_pin);
            }
        }
    }

    if (irq_state[state.instance].dt_count > 0) {
        float dt_avg;

        // disable interrupts to prevent race with irq_handler
        void *irqstate = hal.scheduler->disable_interrupts_save();
        dt_avg = irq_state[state.instance].dt_sum / irq_state[state.
instance].dt_count;
        irq_state[state.instance].dt_count = 0;
        irq_state[state.instance].dt_sum = 0;
        hal.scheduler->restore_interrupts(irqstate);

        const float scaling = ap_rpm._scaling[state.instance];
        float maximum = ap_rpm._maximum.get();
        float minimum = ap_rpm._minimum.get();
        float quality = 0;
        float rpm = scaling * (1.0e6 / dt_avg) * 60;
        float filter_value = signal_quality_filter.get();

        state.rate_rpm = signal_quality_filter.apply(rpm);

        if ((maximum <= 0 || rpm <= maximum) && (rpm >= minimum)) {
            if (is_zero(filter_value)){
```

```
                quality = 0;
            } else {
                quality = 1 - constrain_float((fabsf(rpm-filter_value))/
    filter_value, 0.0, 1.0);
                quality = powf(quality, 2.0);
            }
            state.last_reading_ms = AP_HAL::millis();
        } else {
            quality = 0;
        }
        state.signal_quality = (0.1 * quality) + (0.9 * state.
    signal_quality);
    }

    // assume we get readings at at least 1Hz, otherwise reset quality to
    zero
    if (AP_HAL::millis() - state.last_reading_ms > 1000) {
        state.signal_quality = 0;
        state.rate_rpm = 0;
    }
}
```

ardupilot/libraries/AP_Logger/LogFile.cpp

```
void AP_Logger::Write_RPM(const AP_RPM &rpm_sensor)
{
    float rpm1 = -1, rpm2 = -1, rpm3 = -1, rpm4 = -1;

    rpm_sensor.get_rpm(0, rpm1);
    rpm_sensor.get_rpm(1, rpm2);
    rpm_sensor.get_rpm(2, rpm3);
    rpm_sensor.get_rpm(3, rpm4);

    const struct log_RPM pkt{
        LOG_PACKET_HEADER_INIT(LOG_RPM_MSG),
        time_us     : AP_HAL::micros64(),
        rpm1        : rpm1,
```

```
        rpm2         : rpm2,
        rpm3         : rpm3,
        rpm4         : rpm4
    };
    WriteBlock(&pkt, sizeof(pkt));
}
```

ardupilot/libraries/AP_LogStructure.h

```
struct PACKED log_RPM {
    LOG_PACKET_HEADER;
    uint64_t time_us;
    float rpm1;
    float rpm2;
    float rpm3;
    float rpm4;
};
LOG_STRUCTURE_FROM_AHRS \
    { LOG_DF_FILE_STATS, sizeof(log_DSF), \
      "DSF", "QIHIIII", "TimeUS,Dp,Blk,Bytes,FMn,FMx,FAv", "s--b---", "F
    --0---" }, \
    { LOG_RPM_MSG, sizeof(log_RPM), \
      "RPM",  "Qffff", "TimeUS,rpm1,rpm2,rpm3,rpm4", "sqqqq", "F0000" }, \
    { LOG_RALLY_MSG, sizeof(log_Rally), \
      "RALY", "QBBLLh", "TimeUS,Tot,Seq,Lat,Lng,Alt", "s--DUm", "F--GGB"
    },  \
    { LOG_MAV_MSG, sizeof(log_MAV),   \
      "MAV", "QBHHHBHH",   "TimeUS,chan,txp,rxp,rxdp,flags,ss,tf", "s#----
    s-", "F-000-C-" },   \
```

Appendix B

Quadrotor Moment of inertia

To determine the moment of inertia of the quadrotor analytically, the rigid body can be divided into 6 parts: the motor, arm, battery, pixhawk, ESC and a hollow plate as shown in the picture B.1. The parallel axis theorem is used to calculate de moment of inertia of each component around the rotation axis x, y, z.

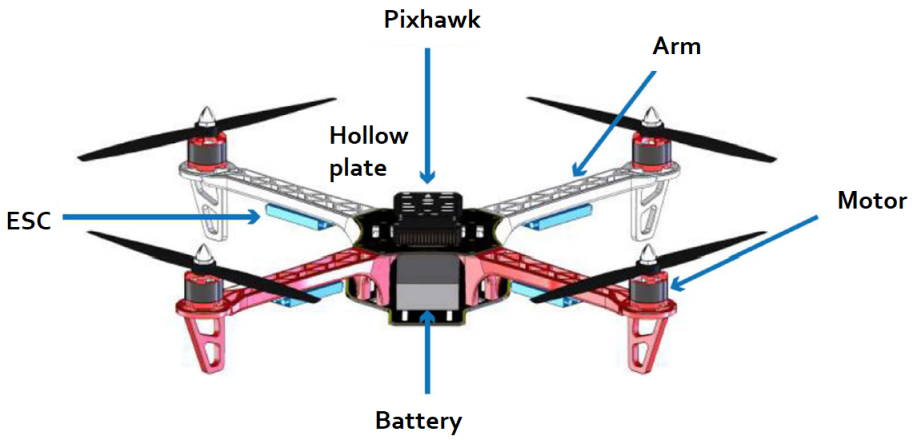


Figure B.1: Quadrotor divided in small parts. *Credit image:* [20]

Parallel axis Theorem S tates that the moment of inertia of an object around a particular axis is equal to the moment of inertia around a parallel axis that goes through the center of mass, plus the distance between the axes of rotation squared times the mass.

$$I = I_{cm} + md^2$$

where I is the moment of inertia of an object with respect to an axis from which the center of mass of the object is a distance d. I_{cm} is the moment of inertia of the object with respect to an axis that is parallel to the first axis and passes through the center of mass, m is the mass of the object, d is the distance between the two axes.

Principal moment of inertia around x axis

$$I_{xx} = 4I_{motor} + 4I_{arm} + I_{battery} + I_{pix} + 4I_{ESC} + 2I_{plate}$$

The motor were modelled as a solid cylinder. Motor mass $m_{motor} = 0.05kg$.

$$\begin{aligned} I_{motor} &= \frac{1}{4}mr^2 + \frac{1}{12}mL^2 + md^2 \\ &= \frac{1}{4}(0.05kg)(0.014m)^2 + \frac{1}{12}(0.05kg)(0.024m)^2 + (0.050kg)(0.16m)^2 \\ &= 1.2842 \times 10^{-3} kgm^2 \end{aligned}$$

The arm is modelled as a solid rectangular prism: $m_{arm} = 0.055kg$.

$$\begin{aligned} I_{arm} &= \frac{1}{12}m(b^2 + c^2) + md^2 = \frac{1}{12}(0.055kg)((0.01m)^2 + (0.03m)^2) + (0.055kg)(0.16m)^2 \\ &= 1.4126 \times 10^{-3} kgm^2 \end{aligned}$$

The battery is modelled as a solid rectangular prism: $m_{battery} = 0.185kg$.

$$I_{battery} = \frac{1}{12}m(b^2 + c^2) = \frac{1}{12}(0.185kg)((0.105m)^2 + (0.023m)^2) = 1.7812 \times 10^{-4} kgm^2$$

The Pixhawk board is modelled as a solid rectangular prism: $m_{arm} = 0.065kg$.

$$I_{pix} = \frac{1}{12}m(b^2 + c^2) = \frac{1}{12}(0.065kg)((0.08m)^2 + (0.015m)^2) = 3.588 \times 10^{-5} kgm^2$$

The ESC is modelled as a solid rectangular prism: $m_{ESC} = 0.025kg$.

$$\begin{aligned} I_{ESC} &= \frac{1}{12}m(b^2 + c^2) + md^2 = \frac{1}{12}(0.025kg)((0.05m)^2 + (0.025m)^2) + (0.025kg)(0.06m)^2 \\ &= 9.65104 \times 10^{-5} kgm^2 \end{aligned}$$

The plate: $m_{plate} = 0.065kg$.

$$I_{plate} = \frac{1}{12}m(b^2 + c^2) = \frac{1}{12}(0.065kg)((0.10m)^2 + (0.05m)^2) = 6.77 \times 10^{-5} kgm^2$$

The total moment of inertia around x axis is:

$$\begin{aligned} I_{xx} &= 4I_{motor} + 4I_{arm} + I_{battery} + I_{pix} + 4I_{ESC} + 2I_{plate} \\ &= 4(1.2842 \times 10^{-3} kgm^2) + 4(1.4126 \times 10^{-3} kgm^2) + 1.7812 \times 10^{-4} kgm^2 \\ &+ 3.588 \times 10^{-5} kgm^2 + 4(9.651 \times 10^{-5} kgm^2) + 2(6.77 \times 10^{-5} kgm^2) \\ &= 0.0115 kgm^2 \end{aligned}$$

Principal moment of inertia around y axis

$$I_{yy} = 4I_{motor} + 4I_{arm} + I_{battery} + I_{pix} + 4I_{ESC} + 2I_{plate}$$

The motor were modelled as a solid cylinder. Motor mass $m_{motor} = 0.05kg$.

$$\begin{aligned} I_{motor} &= \frac{1}{4}mr^2 + \frac{1}{12}mL^2 + md^2 \\ &= \frac{1}{4}(0.05kg)(0.014m)^2 + \frac{1}{12}(0.05kg)(0.024m)^2 + (0.050kg)(0.16m)^2 \\ &= 1.2848 \times 10^{-3} kgm^2 \end{aligned}$$

The arm is modelled as a solid rectangular prism: $m_{arm} = 0.055kg$.

$$\begin{aligned} I_{arm} &= \frac{1}{12}m(a^2 + c^2) + md^2 = \frac{1}{12}(0.055kg)((0.185m)^2 + (0.01m)^2) + (0.055kg)(0.16m)^2 \\ &= 1.5653 \times 10^{-3} kgm^2 \end{aligned}$$

The battery is modelled as a solid rectangular prism: $m_{battery} = 0.185kg$.

$$I_{battery} = \frac{1}{12}m(a^2 + c^2) = \frac{1}{12}(0.185kg)((0.032m)^2 + (0.023m)^2) = 2.394 \times 10^{-5} kgm^2$$

The Pixhawk board is modelled as a solid rectangular prism: $m_{arm} = 0.065kg$.

$$I_{pix} = \frac{1}{12}m(a^2 + c^2) = \frac{1}{12}(0.065kg)((0.05m)^2 + (0.015m)^2) = 1.476 \times 10^{-5} kgm^2$$

The ESC is modelled as a solid rectangular prism: $m_{ESC} = 0.025kg$.

$$\begin{aligned} I_{ESC} &= \frac{1}{12}m(a^2 + c^2) + md^2 = \frac{1}{12}(0.025kg)((0.05m)^2 + (0.007m)^2) + (0.025kg)(0.06m)^2 \\ &= 9.5310 \times 10^{-5} kgm^2 \end{aligned}$$

The plate: $m_{plate} = 0.065kg$.

$$I_{plate} = \frac{1}{12}m(a^2 + c^2) = \frac{1}{12}(0.065kg)((0.10m)^2 + (0.05m)^2) = 6.77 \times 10^{-5} kgm^2$$

The total moment of inertia around y axis is:

$$\begin{aligned} I_{yy} &= 4I_{motor} + 4I_{arm} + I_{battery} + I_{pix} + 4I_{ESC} + 2I_{plate} \\ &= 4(1.2848 \times 10^{-3} kgm^2) + 4(1.5653 \times 10^{-3} kgm^2) + 2.394 \times 10^{-5} kgm^2 \\ &+ 1.476 \times 10^{-5} kgm^2 + 4(9.5310 \times 10^{-5} kgm^2) + 2(6.77 \times 10^{-5} kgm^2) \\ &= 0.0119 kgm^2 \end{aligned}$$

Principal moment of inertia around z axis

$$I_{zz} = 4I_{motor} + 4I_{arm} + I_{battery} + I_{pix} + 4I_{ESC} + 2I_{plate}$$

The motor were modelled as a solid cylinder. Motor mass $m_{motor} = 0.05kg$.

$$I_{motor} = \frac{1}{2}mr^2 + md^2 = \frac{1}{2}(0.05kg)(0.0135m)^2 + (0.05kg)(0.22m)^2 = 2.42455 \times 10^{-3}kgm^2$$

The arm is modelled as a solid rectangular prism: $m_{arm} = 0.055kg$.

$$\begin{aligned} I_{arm} &= \frac{1}{12}m(a^2 + b^2) + md^2 = \frac{1}{12}(0.055kg)((0.185m)^2 + (0.03m)^2) + (0.055kg)(0.06m)^2 \\ &= 3.589 \times 10^{-4}kgm^2 \end{aligned}$$

The battery is modelled as a solid rectangular prism: $m_{battery} = 0.185kg$.

$$I_{battery} = \frac{1}{12}m(a^2 + b^2) = \frac{1}{12}(0.185kg)((0.105m)^2 + (0.05m)^2) = 2.085 \times 10^{-4}kgm^2$$

The Pixhawk board is modelled as a solid rectangular prism: $m_{arm} = 0.065kg$.

$$I_{pix} = \frac{1}{12}m(a^2 + b^2) = \frac{1}{12}(0.065kg)((0.08m)^2 + (0.05m)^2) = 4.8208 \times 10^{-5}kgm^2$$

The ESC is modelled as a solid rectangular prism: $m_{ESC} = 0.025kg$.

$$\begin{aligned} I_{ESC} &= \frac{1}{12}m(a^2 + b^2) + md^2 = \frac{1}{12}(0.025kg)((0.05m)^2 + (0.025m)^2) + (0.025kg)(0.10m)^2 \\ &= 2.5651 \times 10^{-4}kgm^2 \end{aligned}$$

The plate: $m_{plate} = 0.065kg$.

$$I_{plate} = \frac{1}{12}m(a^2 + b^2) = \frac{1}{12}(0.065kg)((0.10m)^2 + (0.10m)^2) = 1.083 \times 10^{-4}kgm^2$$

The total moment of inertia around z axis is:

$$\begin{aligned} I_{zz} &= 4I_{motor} + 4I_{arm} + I_{battery} + I_{pix} + 4I_{ESC} + 2I_{plate} \\ &= 4(2.4245 \times 10^{-3}kgm^2) + 4(3.589 \times 10^{-4}kgm^2) + 2.085 \times 10^{-4}kgm^2 \\ &+ 4.8208 \times 10^{-5}kgm^2 + 4(2.5651 \times 10^{-4}kgm^2) + 2(1.083 \times 10^{-4}kgm^2) \\ &= 0.0126kgm^2 \end{aligned}$$

Bibliography

[1] Accelerometer calibration (2022). https://diyodemag.com/projects/part_three_take_flight_with_h4wk. xii, 60

[2] Ahuatzín, G., Tang, Y., and Espíndola, E. (2022). Navigation observer design using vector measurements and a gps sensor. *IEEE Sensors Journal (Submitted)*. 8

[3] Akella, M. R., Thakur, D., and Mazenc, F. (2015). Partial lyapunov strictification: Smooth angular velocity observers for attitude tracking control. *Journal of Guidance, Control, and Dynamics*, 38(3):442–451. 42

[4] Ardupilot, A., Mendoza-Mendoza, J. A., Gonzalez-Villela, V., Sepulveda-Cervantes, G., Mendez-Martinez, M., and Sossa-Azuela, H. (2020). *Advanced Robotic Vehicles Programming*. Springer. 50

[5] Ardupilot Common rpm sensors (2022). https://ardupilot.org/copter/docs/common-rpm.html. 55

[6] Barczyk, M. and Lynch, A. F. (2012). Invariant observer design for a helicopter uav aided inertial navigation system. *IEEE Transactions on Control Systems Technology*, 21(3):791–806. 6

[7] Barczyk, M. and Lynch, A. F. (2013). Invariant observer design for a helicopter uav aided inertial navigation system. *IEEE Transactions on Control Systems Technology*, 21(3):791–806. 5

[8] Benziane, L., Benallegue, A., Chitour, Y., and Tayebi, A. (2016). Velocity-free attitude stabilization with inertial vector measurements. *International Journal of Robust and Nonlinear Control*, 26(11):2478–2493. 5

[9] Berkane, S., Abdessameud, A., and Tayebi, A. (2016). Global exponential angular velocity observer for rigid body systems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4154–4159. IEEE. 5, 23

[10] Berkane, S., Tayebi, A., and De Marco, S. (2021). A nonlinear navigation observer using imu and generic position information. *Automatica*, 127:109513. 6

[11] Black, H. D. (1964). A passive system for determining the attitude of a satellite. *AIAA journal*, 2(7):1350–1351. 2

[12] Bonnabel, S., Martin, P., and Rouchon, P. (2008). Symmetry-preserving observers. *IEEE Transactions on Automatic Control*, 53(11):2514–2526. 5

[13] Chavez-Moreno, R., Tang, Y., Hernandez, J.-C., and Ji, H. (2018). Contracting angular velocity observer for small satellites. *IEEE Transactions on Aerospace and Electronic Systems*, 54(6):2762–2775. 22

[14] Cirillo, A., Cirillo., P., Maria, G. D., Natale, C., and Pirozzi, S. (2017). *A Comparison of Multisensor Attitude Estimation Algorithms*, chapter 29. CRC Press Taylor and Francis Group. 4

[15] Crassidis, J., Markley, F., and Cheng, Y. (2007). Survey of nonlinear attitude estimation methods. *Journal of Guidance, Control and Dynamics*. 1, 5, 6

[16] Espíndola, E. and Tang, Y. (2022). A new angular velocity observer for attitude tracking of spacecraft. *ISA transactions*. 17, 22

[17] Feng, L. and Fangchao, Q. (2016). Research on the hardware structure characteristics and ekf filtering algorithm of the autopilot pixhawk. In *2016 Sixth International Conference on Instrumentation Measurement, Computer, Communication and Control (IMCCC)*, pages 228–231. 50

[18] Fourati, H. and Belkhiat, D. E. C. (2016). *Multisensor attitude estimation: fundamental concepts and applications*. CRC Press. 5

[19] Grip, H. F., Fossen, T. I., Johansen, T. A., and Saberi, A. (2015). Globally exponentially stable attitude and gyro bias estimation with application to gnss/ins integration. *Automatica*, 51:158–166. 6

[20] Grujin, I. and Nilsson, R. (2016). Model-based development and evaluation of control for complex multi-domain systems: Attitude control for a quadrotor uav. *Department of Engineering, Aarhus University*, Technical report ECE-TR-23:92. xii, 94

[21] Hobbywing Direct (2022). `https://www.hobbywingdirect.com/products/rpm-sensor`. 55

[22] Hua, M.-D. (2010). Attitude estimation for accelerated vehicles using gps/ins measurements. *Control Engineering Practice*, 18:723–732. 4

[23] Hua, M.-D., Ducard, G., Hamel, T., and Mahony, R. (2014). Introduction to nonlinear attitude estimation for aerial robotic systems. *Aerospace Lab*, pages AL08–04. 1, 2, 5

[24] Khalil, H. K. (2002). Nonlinear systems third edition. *Patience Hall*, 115. 22, 32, 36

[25] Lim, H., Park, J., Lee, D., and Kim, H. (2012). Build your own quadrotor: Open-source projects on unmanned aerial vehicles. *IEEE Robotics Automation Magazine*, 19(3):33–45. 50

[26] Magnetic Fields Calculators (2022). https://www.ngdc.noaa.gov/geomag/calculators/magcalc.shtml#igrfwmm. 69

[27] Magnis, L. and Petit, N. (2017). Angular velocity nonlinear observer from vector measurements. *Automatica*, 75:46–53. 5

[28] Mahony, R., Hamel, T., and Pflimlin, J.-M. (2008). Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on automatic control*, 53(5):1203–1217. 4, 6, 8, 32

[29] Mahony, R., Kumar, V., and Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3):20–32. 66

[30] Markley, F. L. and Crassidis, J. (2014). *Fundamentals of Spacecraft Attitude Determination and Control*. Springer. 2, 3, 12

[31] Martin, P. and Salaün, E. (2008). An invariant observer for earth-velocity-aided attitude heading reference systems. *IFAC Proceedings Volumes*, 41(2):9857–9864. 5

[32] Meier, L., Tanskanen, P., Fraundorfer, F., and Pollefeys, M. (2011). Pixhawk: A system for autonomous flight using onboard computer vision. In *2011 IEEE International Conference on Robotics and Automation*, pages 2992–2997. 50

[33] Mission Planner Installation (2022). https://ardupilot.org/planner/docs/mission-planner-installation.html. 58

[34] Mission Planner Overview (2022). https://ardupilot.org/planner/docs/mission-planner-overview.html. 57

[35] Pixhawk Overview (2022). https://ardupilot.org/copter/docs/common-pixhawk-overview.html. 50

[36] Quan, Q. (2017). *Introduction to multicopter design and control*. Springer. xi, 38, 39

[37] Rehbinder, H. and Ghosh, B. K. (2003). Pose estimation using line-based dynamic vision and inertial sensors. *IEEE Transactions on Automatic control*, 48(2):186–199. 1, 6

[38] Roberts, A. and Tayebi, A. (2011). On the attitude estimation of accelerating rigid-bodies using gps and imu measurements. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 8088–8093. 4

[39] RPM Sensor Wiki - Knowledge Share (2022). https://discuss.ardupilot.org/t/rpm-sensor-wiki-knowledge-share/45091. 62

[40] Salcudean, S. (1991). A globally convergent angular velocity observer for rigid body motion. *IEEE transactions on Automatic Control*, 36(12):1493–1497. 22

[41] Sarras, I. (2020). Global exponential estimation of rigid body angular velocity directly from multiple vector measurements. In *2020 European Control Conference (ECC)*, pages 979–984. 5

[42] Schaub, H. and Junkins, J. (2014). *Analytical Mechanics of Space Systems*. AIAA Education Series. 2

[43] Shuster, M. D. et al. (1993). A survey of attitude representations. *Navigation*, 8(9):439–517. 2

[44] Tayebi, A. (2008). Unit quaternion-based output feedback for the attitude tracking problem. *IEEE Transactions on Automatic Control*, 53(6):1516–1520. 22

[45] Tayebi, A., Roberts, A., and Benallegue, A. (2013a). Inertial vector measurements based velocity-free attitude stabilization. *IEEE Transactions on Automatic Control*, 58(11):2893–2898. 5

[46] Tayebi, A., Roberts, A., and Benallegue, A. (2013b). Inertial vector measurements based velocity-free attitude stabilization. *IEEE Transactions on Automatic Control*, 58(11):2893–2898. 23

[47] Wahba, G. (1965). A least squares estimate of satellite attitude. *SIAM Review*, 7(3):409–409. 2

[48] Yang, Z., Lin, F., and Chen, B. M. (2016). Survey of autopilot for multi-rotor unmanned aerial vehicles. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*, pages 6122–6127. 50