

Vinícius Freitas Rodrigues

**Development of a Model-in-the-Loop Environment
and Design of an LQI Attitude Controller for
Open-Source Quadcopter Autopilots**

Uberlândia

2026

Vinícius Freitas Rodrigues

Development of a Model-in-the-Loop Environment and Design of an LQI Attitude Controller for Open-Source Quadcopter Autopilots

Dissertation presented to the Faculty of Electrical Engineering at Universidade Federal de Uberlândia as a partial requirement for obtaining the **Master of Science** degree.

Área de Concentração: Sistemas de Energia Elétrica

Linha de Pesquisa: Eletrônica Industrial, Sistemas e Controles Eletrônicos

Universidade Federal de Uberlândia - UFU

Faculdade de Engenharia Elétrica - FEELT

Advisor: Prof. Dra. Gabriela Vieira Lima

Co-advisor: Prof. Dr. Thiago Vaz da Costa

Uberlândia

2026

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

R696
2026 Rodrigues, Vinicius Freitas, 1999-
Development of a Model-in-the-Loop Environment and Design of
an LQI Attitude Controller for Open-Source Quadcopter Autopilots
[recurso eletrônico] / Vinicius Freitas Rodrigues. - 2026.

Orientadora: Gabriela Vieira Lima.

Coorientador: Thiago Vaz da Costa.

Dissertação (Mestrado) - Universidade Federal de Uberlândia,
Pós-graduação em Engenharia Elétrica.

Modo de acesso: Internet.

DOI <http://doi.org/10.14393/ufu.di.2026.557>

Inclui bibliografia.

Inclui ilustrações.

1. Engenharia elétrica. I. Lima, Gabriela Vieira, 1990-, (Orient.).
II. Costa, Thiago Vaz da, 1982-, (Coorient.). III. Universidade
Federal de Uberlândia. Pós-graduação em Engenharia Elétrica. IV.
Título.

CDU: 621.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091

Nelson Marcos Ferreira - CRB6/3074



ATA DE DEFESA - PÓS-GRADUAÇÃO

Programa de Pós-Graduação em:	Engenharia Elétrica				
Defesa de:	Dissertação de Mestrado, 824, PPGEELT				
Data:	Dez de julho de dois mil e vinte e seis	Hora de início:	15:30	Hora de encerramento:	18:00
Matrícula do Discente:	12422EEL005				
Nome do Discente:	Vinícius Freitas Rodrigues				
Título do Trabalho:	Development of a Model-in-the-Loop Environment and Design of an LQI Attitude Controller for Open-Source Quadcopter Autopilots				
Área de concentração:	Sistemas de Energia Elétrica				
Linha de pesquisa:	Eletrônica Industrial, Sistemas e Controles Eletrônicos				
Projeto de Pesquisa de vinculação:	Coordenadora do projeto: Gabriela Vieira Lima. Título do Projeto: Investigação do uso de VANTs para manutenção preditiva e atuação em emergências no contexto da Indústria 4.0. Agência financiadora: FAPEMIG. Número do processo na agência financiadora: APQ-00594-22. Vigência do projeto: 07/07/2022 até 06/07/2025.				

Reuniu-se através de videoconferência, a Banca Examinadora designada pelo Colegiado do Programa de Pós-graduação em Engenharia Elétrica, assim composta:

Doutores: Éder Alves de Moura (UFU), Ícaro Bezerra Viana (ITA) e Gabriela Vieira Lima, orientadora do discente.

Iniciando os trabalhos a presidente da mesa, Prof^a. Dr^a. Gabriela Vieira Lima, apresentou a Comissão Examinadora e o candidato, agradeceu a presença do público, e concedeu ao discente a palavra para a exposição do seu trabalho. A duração da apresentação do discente e o tempo de arguição e resposta foram conforme as normas do Programa.

A seguir a senhora presidente concedeu a palavra, pela ordem sucessivamente, aos examinadores, que passaram a arguir o candidato. Ultimada a arguição, que se desenvolveu dentro dos termos regimentais, a Banca, em sessão secreta, atribuiu o resultado final, considerando o candidato:

APROVADO.

Esta defesa faz parte dos requisitos necessários à obtenção do título de Mestre. O competente diploma será expedido após cumprimento dos demais requisitos, conforme as normas do Programa, a legislação pertinente e a regulamentação interna da UFU.

Nada mais havendo a tratar foram encerrados os trabalhos. Foi lavrada a presente ata que após lida e achada conforme, foi assinada pela Banca Examinadora.



Documento assinado eletronicamente por **Gabriela Vieira Lima, Professor(a) do Magistério Superior**, em 15/07/2026, às 13:14, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **Eder Alves de Moura, Professor(a) do Magistério Superior**, em 15/07/2026, às 14:23, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



Documento assinado eletronicamente por **ICARO BEZERRA VIANA, Usuário Externo**, em 15/07/2026, às 14:58, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site https://www.sei.ufu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **7429528** e o código CRC **EB498DBB**.

Referência: Processo nº 23117.041770/2026-48

SEI nº 7429528

Este trabalho é dedicado a batata, zebra, morceguinho, jacaré e Diana.

Acknowledgements

Agradeço aos meus pais, Luciana e José Humberto que me deram todo apoio necessário durante toda minha formação. Sei que a carreira acadêmica é uma realidade impossível para maior parte das pessoas. Entendo que minha oportunidade e privilégio vêm de uma família que sempre confiou na minha capacidade e investiu no meu futuro. Portanto, sou eternamente grato.

Agradeço ao Nathan pelo carinho e pela motivação. Com certeza os problemas que enfrentei durante essa trajetória se tornam pequenos na sua presença.

Agradeço também aos meus orientadores Gabriela Vieira Lima e Thiago Vaz da Costa pela oportunidade de aprender sobre temas tão desafiadores e por confiarem no meu potencial de produzir esta pesquisa.

Agradeço ao meu amigo Gabriel pelos momentos compartilhando as batalhas de nossa caminhada pelo mestrado.

Finalmente, agradeço ao programa de Pós-Graduação da Faculdade de Engenharia Elétrica e a Universidade Federal de Uberlândia que sustentam, em um ato de resistência, a produção científica e o ensino superior de qualidade em nossa região. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

“[...]”
Knowledges are a deadly friend
If no one sets the rules
The fate of all mankind I see
Is in the hands of fools”
(Epitaph - King Crimson)

Abstract

Research on Unmanned Aerial Vehicles (UAVs) has grown significantly in recent years and is expected to continue expanding due to the wide variety of vehicle designs, applications, regulations, and performance requirements. These differences influence both aircraft dynamic modeling and control strategies. At the same time, the rapid growth of the commercial UAV market, ranging from consumer drones to military systems, has increased the demand for the integration of advanced sensors, actuators, processing units, and payloads, creating numerous opportunities and challenges for research and development in both academia and industry. Although modern control techniques for quadcopters are widely studied and have demonstrated successful implementations, open-source autopilots and commercial UAV manufacturers still predominantly rely on classical Proportional-Integral-Derivative (PID) control architectures. This persistence is largely driven by the need for backward compatibility, hardware limitations, support for multiple UAV platforms, and the prioritization of software reliability and extensibility over control-system redesign. Some academic studies have implemented advanced controllers, but these are often executed on companion computers rather than embedded directly in the flight controller. Consequently, there remains a gap in the literature regarding implementations of modern control approaches within open-source autopilot frameworks, motivating efforts to transfer successful academic and industrial control concepts to open-source flight systems. In this context, this work presents a dynamic model of a quadcopter-type UAV that attempts to satisfactorily represent the use of the aircraft in most use cases. Furthermore, before proposing a new control strategy, a study is also conducted to analyze the existing control structure in the largest open-source autopilot: ArduPilot. With a solid foundation simulated using a Model-in-the-Loop (MITL), the research achieves the potential to compare and analyze the practical feasibility of different control strategies with the already established standard. Therefore, based on the developed model, an Optimal Control architecture is designed to replace the original autopilot control structure. In order to expand the flight envelope, the linear optimal controller is associated with a gain-scheduling technique. This controller was initially designed only for the attitude control loop. The developed strategy allowed for trajectory tracking with performance similar to the existing architecture with minimal control effort. For standard gain matrix selections, using conservative maximum errors according to Bryson's Rule, the tuning of the optimal Linear-Quadratic-Integral (LQI) control increased the Integral Absolute Error (IAE) by 5.76% in comparison with the original PID. Nevertheless the optimal controller reduced the Integral Squared Input (ISU) by 0.1% and the Total Variation (TV) by 50.45%.

Keywords: Optimal Control; Quadcopter; ArduPilot; PID Control; Dynamic Modeling.

Resumo

A pesquisa sobre Veículos Aéreos Não Tripulados (VANTs) cresceu significativamente nos últimos anos e espera-se que continue a se expandir devido à grande variedade de projetos, aplicações, regulamentações e requisitos de desempenho dos veículos. Essas diferenças influenciam tanto a modelagem dinâmica das aeronaves quanto as estratégias de controle. Ao mesmo tempo, o rápido crescimento do mercado comercial de VANTs, que abrange desde drones de consumo até sistemas militares, aumentou a demanda pela integração de sensores, atuadores, unidades de processamento e cargas úteis avançadas, criando inúmeros desafios para pesquisa e desenvolvimento tanto na academia quanto na indústria. Embora as técnicas modernas de controle para quadricópteros sejam amplamente estudadas e tenham demonstrado implementações bem-sucedidas, os autopilotos de código aberto e os fabricantes comerciais de UAVs ainda dependem predominantemente de arquiteturas de controle Proporcional-Integral-Derivativo (PID). Essa persistência é impulsionada principalmente pela necessidade de retrocompatibilidade, limitações de hardware, suporte a múltiplas plataformas de UAVs e pela priorização da confiabilidade e extensibilidade do software em detrimento da reformulação do sistema de controle. Embora alguns estudos acadêmicos tenham implementado controladores avançados, estes são frequentemente executados em computadores auxiliares, em vez de serem incorporados diretamente à controladora de voo. Consequentemente, ainda existe uma lacuna na literatura em relação às implementações de abordagens de controle modernas em firmwares de controle de voo de código aberto, o que motiva esforços para transferir conceitos de controle acadêmicos para esses sistemas autônomos *open-source*. Este trabalho, nesse contexto, apresenta um modelo dinâmico de um UAV do tipo quadricóptero que busca representar satisfatoriamente o uso da aeronave na maioria dos casos de aplicação. Além disso, antes de propor uma nova estratégia de controle, realiza-se também um estudo para analisar a estrutura de controle existente no maior piloto automático de código aberto: o Ardupilot. Com uma base sólida simulada utilizando um *Model-in-the-Loop* (MITL), a pesquisa alcança o potencial de comparar e analisar a viabilidade prática de diferentes estratégias de controle com o padrão já estabelecido. Portanto, com base no modelo desenvolvido, uma arquitetura de Controle Ótimo foi projetada para substituir a estrutura de controle original do autopiloto. Para ampliar o envelope de voo, o controlador ótimo linear foi associado a uma técnica de escalonamento de ganhos. Inicialmente, esse controlador foi projetado apenas para o malha de controle de atitude. A estratégia desenvolvida permitiu o rastreamento de trajetória com desempenho semelhante à arquitetura existente, com esforço de controle mínimo. Para seleções padrão da matriz de ganho, utilizando erros máximos conservadores conforme a Regra de Bryson, o ajuste do controle ótimo Linear-Quadrático-Integral (LQI) aumentou o Erro Integral Absoluto (IAE) em 5,76% em comparação com o PID original. No entanto, o controlador ótimo reduziu a Integral do Quadrado da Entrada (ISU) em 0,1% e a Variação Total (TV) em 50,45%.

Key-words: Controle Ótimo; Quadricóptero; Ardupilot; Controle PID; Modelagem Dinâmica.

List of Figures

Figure 1 – Multirotor drones in the market with different size scales: (a) the toy nano quadcopter Cheerson CX-10, (b) the military nano drone Black Hornet Nano, (c) the passenger drone EHang EH216-S and (d) the agricultural quadcopter prototype Agrobbee 200.	15
Figure 2 – Block diagram of a PID in parallel form.	24
Figure 3 – Block diagram of a PID in ideal form.	25
Figure 4 – Block diagram of a PID in series form.	25
Figure 5 – Representation of a closed-loop control system highlighting the actuator non-linear dynamics.	26
Figure 6 – Block diagram of an anti-windup algorithm based on conditional integration.	27
Figure 7 – Mapping of the stability region of s-plane on the z-plane for the transformations in Equations 2.13, 2.14 and 2.15.	28
Figure 8 – Block diagram of typical cascade control structure.	29
Figure 9 – Common feedforward control approaches where (a) is the motion control approach and (b) is the 2-DoF control system.	30
Figure 10 – Block diagram of a linear system state-space representation with state feedback control.	32
Figure 11 – Block diagram of the state feedback tracking system.	33
Figure 12 – Overview of the multirotor UAV control loops: the outer position control loop, the intermediate attitude control loop, and the innermost motors control loop.	38
Figure 13 – Quadrotor UAV in (a) <i>plus</i> and (b) <i>X</i> configurations. The scheme shows the motors numbering accordingly with the Ardupilot standard.	39
Figure 14 – Coordinate frames of a quadrotor UAV.	40
Figure 15 – CAD model inspired by the experimental drone.	52
Figure 16 – Test benches for estimating propeller parameters, where: (a) is the thrust test stand and (b) is the drag test stand.	53
Figure 17 – Curve fitting for the thrust test, where: (a) is a linear fit relating the signal pulse width and the rotation speed and (b) is a second-order polynomial fit relating the thrust/drag forces and the rotation speed.	53
Figure 18 – Steps in data processing where: (a) is the raw data, (b) the modulus of the measured quantities, and (c) the data with filtered pitch measurements.	55
Figure 19 – Flight periods corresponding to navigation between waypoints, against and with the wind respectively, where: the graphs on the left (a) contain the pitch curves before correction for the deviation caused by the wind, the graphs on the right (b) contain the pitch curves after correction.	57

Figure 20 – Linear regression used to compute the induced aerodynamic drag coefficient k_d .	57
Figure 21 – Top view diagram of a generic ArduPilot use case.	60
Figure 22 – Ardupilot attitude controller block diagram. The structure is the same for the xy linear motion control, but with a traditional P controller for position. . . .	62
Figure 23 – Cascade controllers used in the z-axis motion.	63
Figure 24 – Ardupilot PID with all its available features.	64
Figure 25 – Optimal control strategy developed for application in quadcopters.	66
Figure 26 – Interpolation algorithm used to estimate gain matrices, steady-state states, and steady-state inputs at operating points intermediate to those obtained with linearization.	67
Figure 27 – Step responses used to verify attitude control performance.	69
Figure 28 – UAV prototype used for the project The vehicle is based on the F450 frame and the Pixhawk 2.4.8 controller.	70
Figure 29 – Communication interfaces between UAV and GCS and between UAV modules.	71
Figure 30 – AUTO flight comparison between the WPNav setpoints and both real and MITL trajectories, in which: (a) is the 3D position, (b) is the xy-plane speeds, (c) is the roll angle, (d) is the z-axis speed, (e) is the yaw angle and (f) the pitch angle.	74
Figure 31 – Visualization of the test trajectory presented in a SITL window.	76
Figure 32 – Comparison between the performance indices of the PID and optimal con- trollers.	77
Figure 33 – Comparison between position and attitude of the PID and LQI controlled systems in simulation.	79
Figure 34 – Comparison between linear velocities and angular rates of the PID and LQI controlled systems in simulation.	80
Figure 35 – Comparison between BLDC motor speeds of the PID and LQI controlled systems in simulation.	81
Figure 36 – Comparison between the performance indices of the PID and optimal con- trollers with feedforward gains in the linear speed control loops.	82
Figure 37 – Comparison between the performance indices of the optimal controller operating both at linearized operating points and at operating points with interpolated K , X_{ss} and U_{ss}	83
Figure 38 – Free-body diagram of the UAV representing its horizontal velocities and drag forces on the body frame.	98
Figure 39 – Simulink diagram of the LQI attitude control.	100
Figure 40 – Simulink diagram of the general cascade quadcopter control and dynamics. .	101
Figure 41 – Simulink diagram of the Ardupilot's position controllers.	102
Figure 42 – Simulink diagram of the Ardupilot's attitude control.	103

List of Tables

Table 1	– Table showing the maximum acceptable values for states and inputs used to calculate weights based on Bryson’s Rule.	68
Table 2	– Ardupilot controller parameters used in the simulations.	73
Table 3	– Quantitative analysis of deviations between the simulated model and a real flight.	75
Table 4	– Average values of the IAE, ISU, and TV indices for the PID and LQI attitude controllers tested for tracking the same trajectory.	78
Table 5	– Table with the experimental drone’s configuration and the weight of the parts.	94
Table 6	– Experimental drone parameters.	95

List of Abbreviations and Acronyms

ANN	<i>Artificial Neural Network</i>
ARE	<i>Algebraic Riccati Equation</i>
BLDC	<i>Brushless Direct-Current</i>
CAD	<i>Computer Aided Design</i>
DARE	<i>Discrete Algebraic Riccati Equation</i>
DOF	<i>Degrees-of-Freedom</i>
ESC	<i>Electronic Speed Controller</i>
GCS	<i>Ground Control Station</i>
GNSS	<i>Global Navigation Satellite System</i>
GPS	<i>Global Positioning System</i>
HAL	<i>Hardware Abstraction Layer</i>
IAE	<i>Integral Absolute Error</i>
IMU	<i>Inertial Measurement Unit</i>
ISU	<i>Integral Squared Input</i>
LQG	<i>Linear-Quadratic-Gaussian</i>
LQI	<i>Linear-Quadratic-Integral</i>
LQR	<i>Linear-Quadratic-Regulator</i>
MAVLink	<i>Micro Air Vehicle Link</i>
MITL	<i>Model-in-th-Loop</i>
MPC	<i>Model Predictive Control</i>
NRMSE	<i>Normalised Root Mean Squared Error</i>
PID	<i>Proportional-Intergral-Derivative</i>
PWM	<i>Pulse Width Modulation</i>
RC	<i>Remote Control</i>
RL	<i>Reinforcement Learning</i>
RMSE	<i>Root Mean Squared Error</i>
SITL	<i>Software-in-the-Loop</i>
TV	<i>Total Variation</i>
UAV	<i>Unmanned Aerial Vehicle</i>

Contents

1	INTRODUCTION	13
1.1	Motivation	13
1.2	Objectives	19
1.3	Thesis Organization	19
2	FUNDAMENTALS	21
2.1	PID and Classical Control Techniques	21
2.1.1	The Proportional Action	22
2.1.2	The Integral Action	23
2.1.3	The Derivative Action	23
2.1.4	PID Control Structures	24
2.1.5	Integrator Anti-Windup	26
2.1.6	PID Digital Implementation	27
2.1.7	Cascade Control	28
2.1.8	Feedforward Action in Motion Control	29
2.2	Classical Optimal Control Strategies	30
2.2.1	Linear State Feedback Control Laws	31
2.2.2	A General Formulation for the Optimal Control Problem	34
2.2.3	<i>Linear-Quadratic</i> (LQ) Techniques in Optimal Control	34
2.2.4	Digital Implementation of <i>Steady-State</i> LQR and LQI Controllers	36
2.3	Performance Assessment of Control Systems	36
3	MATHEMATICAL MODELING	38
3.1	The quadcopter from the perspective of Control Systems	38
3.2	Coordinate Frames	40
3.3	Newton-Euler Formalism	43
3.4	UAV Model Derivation	45
3.5	Determination of the System Parameters	51
3.5.1	Estimating the propeller coefficients with test stands	52
3.5.2	Aerodynamic drag estimation	54
3.6	Linear State-Space Representations	57
4	MATERIALS AND METHODS	59
4.1	Ardupilot Software Architecture	59
4.1.1	Firmware Organization	60
4.1.2	Autonomous Mission Development	61

4.1.3	Simulation and Testing	62
4.2	Ardupilot's Control Architecture Modeling	62
4.3	Optimal Control Architectures and Implementation	65
4.4	UAV Assembly and Experimental Approach	70
	5 RESULTS	72
5.1	Simulation Model Validation	72
5.2	Analysis of Simulated Control Systems	75
	6 CONCLUSION	85
6.1	Future Work	86
	BIBLIOGRAPHY	87
	A EXPERIMENTAL DRONE CONFIGURATION	94
	B ADDITIONAL MODELING	96
B.1	Attitude Planner	96
B.2	Aerodynamic Drag in the Earth Frame	98
	C SIMULINK MODEL DIAGRAMS	100
	D ATTITUDE CONTROLLER GAIN MATRICES	104

1

Introduction

Over the past decade, Artificial Intelligence (AI), 5G telecommunications, and the Internet of Things (IoT) have broken many technological paradigms and changed the general public's perception of science and technology.

Despite that, contrary to popular belief, these technologies alone do not shape the evolution of various engineering sectors, such as robotics and aerospace. There are areas of knowledge shaped by studies that have been developing slowly since the space race, such as modern Control Theory, which gradually develops new techniques for complex problems. Not only is theoretical development slower in these areas, but the adoption of new techniques by industries related to these technologies is also cautious, requiring rigorous scientific methodology and assertive implementations.

Therefore, the development of Control Systems applications remains fundamentally relevant in the AI age. Mobile robotics, whether with ground, aerial, surface, or underwater vehicles, still has much to be developed in terms of control algorithms. Even though AI can accelerate the development of these robots or provide cognitive intelligence for some tasks (such as vision and decision-making), artificial intelligence technology does not yet replace techniques based on numerical processing, signal processing, and optimization for the control of systems.

1.1 Motivation

In recent years, a massive amount of research on Unmanned Aerial Vehicles (UAVs) has been published. It is expected that this number of studies will remain high in the coming years, given that these vehicles differ in design and technology to meet different applications and regulations. Furthermore, performance requirements also vary according to legislation or the criticality of the application. These differences are reflected in the dynamic modeling of the aircraft, and consequently in their control strategy.

Beyond the academic field, the market is flooded with new drones. From toys to military applications, the variety of tasks that these mobile flying robots can perform is vast. The demand for new products creates challenges in integrating different technologies for sensors, actuators, processing, and payloads. Therefore, the private market also opens up a range of possibilities in terms of research and development of UAVs.

It is common to categorize UAVs using classifications based on size, design, or application. The most common approach; and used in many countries' legislations; is to categorize the drones by size based on their maximum takeoff weight. The Canadian Aviation Regulatory Advisory Council (CARAC), for example, defines 4 classes (CARAC, 2025):

- Microdrones: $m < 250g$;
- Small-sized: $250g \leq m < 25kg$;
- Medium-sized: $25kg \leq m < 150kg$;
- Large-sized: $m > 150kg$;

Some agencies adopt a regulation based on fewer categories, such as the National Civil Aviation Agency (Portuguese: Agência Nacional de Aviação Civil, ANAC) from Brazil, which defines only 3 classes (ANAC, 2023). Other agencies, like the Federal Aviation Administration (FAA) from the United States of America, create a more complex categorization divided into 5 groups based on maximum takeoff weight, normal operating altitude, and speed (FAA, 2019).

The design-based approach divides drones based on its structure and propulsion technologies. A survey of Karam et al. (2024) divides UAVs into 5 broader types: blimps, flapping-wing, parafoil-wing, fixed-wing and rotary-wing. The author also shows that the rotary-wing UAVs can be further classified into different types, such as helicopter, coaxial-coper, tricopter, quadcopter, hexacopter and octocopter. As in the article, this work focuses on the study of quadcopter-type rotary-wing drones. This type of vehicle, in itself, already presents a wide range of applications and limitations to be analyzed and improved. Figure 1 shows examples of rotary-wing drones in a variety of designs, sizes and applications.

In terms of applications, there is a huge and constantly expanding classification for UAVs. Some relevant roles of these vehicles include: inspection, agriculture, military use, audiovisual production, and mapping. Some future applications currently under research include product delivery, personal transportation, and high-risk maintenance tasks.

Until 2019, the most potentially valuable UAV-powered solutions in industry were that related to infrastructure and agriculture, which could correspond to 45.2% and 32.4% of the total market. By that time the addressable market value of UAV uses were already over US\$127 million (SHAKHATREH et al., 2019). The drone market itself is also relevant and promising.

Figure 1 – Multirotor drones in the market with different size scales: (a) the toy nano quadcopter Cheerson CX-10, (b) the military nano drone Black Hornet Nano, (c) the passenger drone EHang EH216-S and (d) the agricultural quadcopter prototype Agrobbee 200.



(a)



(b)



(c)



(d)

Source: (a) [Drone Rich \(2017\)](#), (c) [Nakola \(2025\)](#), (c) [Wiepen \(2017\)](#), (d) [Leite \(2024\)](#)

A report developed in 2022 indicates that the market is projected to grow from a value of 7.2 billion in that year to 19.8 billion in 2031 ([GERTLER; ZORETICH, 2022](#)).

In Brazil, according to [World Bank \(2025\)](#), the estimated market value in 2021 was approximately US\$67 million, with a projected value of US\$321.6 million for 2026. In the country, the dominant sector is agriculture, and it is predicted that this will remain the main application for the coming years. However, there are still emerging sectors that could occupy significant shares of this market, such as: pharmaceutical delivery, environmental and forestry monitoring, and mining inspection.

The current literature highlights the wide range of applications for agricultural drones and the current limitations of the technologies and techniques used. [Karam et al. \(2024\)](#) shows successful researches applying quadcopters for chemical spraying, irrigation inspection, crop mapping and monitoring, planting and seeding and plant health assessment. The survey also states that exists limitations some of them in terms of controls, obstacle avoidance and power consumption. For the authors advanced control techniques, real-time object avoidance and path-optimization algorithms (alongside with the design improvements) could improve these stated problems. [Amaral et al. \(2020\)](#), analyzing the use of drones in Agriculture 4.0, observed that for

phytosanitary spraying and the distribution of bioinputs, there are three directions for technical and scientific exploration. According to the authors, the solutions would be: the development of energy-efficient and high-payload-capacity spraying UAVs, the determination of ideal spraying conditions, and the development of peripherals (sensors and actuators) that improve spraying efficiency.

The application in infrastructure inspection, the most popular worldwide, faces similar limitations. As evidenced by [Shakhathreh et al. \(2019\)](#), one of the biggest challenges of this task concerns limited energy, short flight time, and limited processing capabilities. Some specific applications, such as the inspection of underground sites, also present additional difficulties as GPS-denied navigation which requires: advanced localization and mapping techniques, mesh communication, and swarm intelligence ([ZHANG et al., 2025](#)).

Nevertheless, the successful case studies in both areas are noteworthy. [Chen et al. \(2021\)](#) applied a UAV-based system to fight the infestation of *Tessaratoma papillosa* pest Taiwanese orchards. Using reconnaissance drones equipped with the Tiny-YOLOv3 neural network on the NVIDIA Jetson TX2 platform, the UAV-based system identifies the life stage and exact location of insects in real time. This georeferenced information allows a second agricultural drone to perform precise pesticide spraying, optimizing the flight path to save energy. [Chatzargyros et al. \(2024\)](#) showed the application of UAV for inspection of medium voltage power line infrastructure of Lesvos Island in Greece. It integrated drone provided images to a Geographic Information System (GIS) with AI on a web platform. These examples show that, regardless of the limitations, the use of UAVs already represents a mature technology in full adoption.

From large-capacity agricultural drones to small-sized drones for infrastructure inspection, all face a similar challenge: the search for efficiency to compensate for the bottleneck in the energy availability of current batteries. This optimization can be achieved both by improving the performance of the drone's task and by reducing the power consumption of the actuators.

When considering control strategies, many approaches have already been validated. Surveys ([LI; SUN; JIN, 2015](#); [KHALID et al., 2024](#)) show that among the techniques discussed in the literature, the most popular are: Proportional-Integral-Derivative (PID), Linear-Quadratic (LQ) optimal controllers, H-Infinity (H_∞), Model Predictive Control (MPC), Sliding Mode Control (SMC), Backstepping Control (BC), Feedback Linearization (FB), Fuzzy Logic Control (FLC), Artificial Neural Network (ANN), Iterative Learning Control (ILC), and Reinforcement Learning Control (RLC).

While some modern techniques are linear or model-based (such as LQ, H_∞ , MPC, FB and BC), others are learning-based (FLC, ANN, ILC, RLC). This differentiation defines the difference between controllers that seek optimality and robustness versus parameterization flexibility. Model-based techniques tend to be more transparent and analytically rigorous, providing formal mathematical guarantees of stability, performance, and robustness margins, but are highly dependent on the mathematical fidelity with which the system dynamics have been represented.

Meanwhile, learning-based techniques approach the plant by extracting behaviors directly from empirical data or interactions with the environment. They are capable of mapping complex nonlinearities and adapting to environments where the analytical model, or model parameters, are unknown or very difficult to identify.

The major problem with the flexibility offered by learning-based techniques is the loss of analytical transparency. It is notoriously complex (and often impossible) to derive formal proofs of global stability for ANNs or Reinforcement Learning (RL) operating in a closed loop. This results in behavior that can be unpredictable in scenarios (or states) that were not adequately covered during the training or parameterization phase.

Therefore, in critical systems, such as flight control, it is preferable not to delegate the entire controller algorithm to machine learning-based techniques. And the application of these modern techniques, focusing on linear or model-based strategies, is already standard practice in these leading sectors of industry:

- Boeing uses controllers based on gain-scheduled Linear-Quadratic Regulator (LQR) to control the lateral movement (yaw angle) of the 747 airplane (BLIGHT; DAILEY; GANGSAAS, 1994). By 2003, Boeing had already used LQR and Linear-Quadratic-Gaussian (LQG) control strategies for two decades (BALAS, 2003);
- The researches that originated the SpaceX Falcon 9 retrorocket used a control strategy based on online convex optimization similar to a MPC control strategy (BLACKMORE; AÇIKMEŞE; SCHARF, 2010);
- Scientific satellites, such as the CubeSat SNUGLITE developed by Seoul National University, use LQG in their attitude control system (KIM et al., 2020). The Brazilian Multimission Platform Satellite, developed by INPE and used in the Amazônia 1 satellite (INPE, 2024), also applies a control algorithm based on LQG (ARANTES; MARTINS-FILHO; SANTANA, 2009).

There are a considerable number of researches showing the success of these industry validated modern algorithms for multirotor drones. The work done by Okasha, Kralev e Islam (2022), for example, shows a detailed comparison, with experimental results, of linear and/or model-based techniques for the commercial mini-drone Parrot Mambo. The work establishes a comparison between PID, LQR, and MPC. The paper argues that the use of MPC is advantageous because it guarantees performance and robustness while explicitly allowing for handling constraints. However, it points out that in practical implementation, MPC's performance was surpassed by LQR, given the reduction in samples and prediction horizon required to embed the complex algorithm in the limited hardware of the Parrot Mambo. Meanwhile, PID's performance was average, with its only advantage being manual tuning.

Another success case demonstrating the practical application of modern control embedded in quadcopters is the TinyMPC. Addressing the problem of MPC with the computational complexity, [Nguyen et al. \(2024\)](#) developed the TinyMPC, a high speed MPC solver with low memory footprint for constrained microcontrollers. The work demonstrated the algorithm for trajectory tracking control of a nano-quadcopter powered by an STM32F4. TinyMPC was the only strategy, compared to two other algorithms, that managed to track the trajectory while respecting the actuator limits.

Despite the widespread availability of modern control techniques and successful implementation cases for quadcopters, open-source autopilots and the UAV manufacturers remain tied to classic PID-based control techniques.

An overview of open flight control systems helps to understand the design decisions that tie its controllers to classical techniques. The autopilots are embedded computers running software that provides the necessary control and guidance for UAVs. An example of a popular open-hardware platform are the Pixhawk boards. The boards are based mostly on ARM STM32Fx (F3, F4 or F7), with some high-end versions working on STM32H7 processors. Not only does the hardware comprise the autonomous flight system, but also the firmware it runs, with the most popular (and well-supported) for execution on open platforms being Ardupilot and PX4. The capabilities of each Open-Source UAV autopilot is explored by [Aliane \(2024\)](#). The combination of backward compatibility with weak hardware, flexibility to operate across various UAV topologies, and broad configurability leads developers to opt for supporting other elements in the code, such as compatibility with new peripherals, communication interfaces, and Real-Time-Operating-System (RTOS) reliability, rather than remodeling the control structure.

ArduPilot, the autopilot studied in this work, is an advanced open-source software suite designed to control and navigate various autonomous unmanned vehicles, including drones, rovers, and submarines. Some academic work has attempted to implement modern control with drones and Ardupilot. More have taken the offboard strategy, running the modern controller on a companion computer connected to the autopilot, instead of embedding it in the flight controller. Examples include the work of [Muvva et al. \(2025\)](#), who implemented an MPC controller for static and dynamic trajectory tracking of agricultural drones, and [Hamza, Mohamed e El-Badawy \(2022\)](#), who implemented a robust H_∞ controller for a quadcopter.

The work of [Li, Liu e Baldi \(2021\)](#) already proposed an adaptation for the control strategy, that improves control adaptability for in-flight mass change while keeping the system plug-and-play for the final user. This work implemented the algorithm for the autopilot, but did not replace the existing embedded PID loops and was only tested in the Ardupilot Software-in-the-loop environment without practical testing.

In the literature, there is a lack of practical implementations of adaptations for open-source autopilot control systems that maintain good usability. This work is motivated by the possibility of reproducing successful concepts from academia and the cutting-edge aeronautical

industry in open-source flight controllers.

1.2 Objectives

Given the ever-increasing variety of aircraft, the growing requirements of applications, and the lack of innovation in control techniques for user-friendly systems in autopilots, this work aims to bring an academic technique to an open-source controller while maintaining a simplified user experience. In this way, open flight control systems would be better able to handle integration with demanding applications and aircraft with distinct characteristics.

For this study, ArduPilot is the chosen autopilot system, the test platform is a quadcopter, and the focus is on optimal control techniques. This work seeks analytical validation using mathematical models and experimental tests to demonstrate operation in real-world scenarios. The aim is to estimate most of the aircraft parameters using experimental procedures.

To achieve this, the work will seek to model the dynamics of the quadcopter considering all the significant physical phenomena in open-space flights and implement this model in simulation environment using Matlab/Simulink. To provide a consistent basis for comparison with the proposed new algorithm, the current control architecture of ArduPilot must be represented within a Model-in-the-Loop simulation framework coupled to the dynamic model in Simulink.

Then, an algorithm based on LQR and Linear-Quadratic-Integral (LQI), also coupled with dynamic simulation, will be implemented. Finally, the two control architectures will be compared via model by performance criteria.

1.3 Thesis Organization

The remainder of this dissertation is organized as follows:

- **Chapter 2:** This chapter presents the bibliographic revision about the control techniques used in the current Ardupilot algorithm and in the new algorithm developed by this work. Therefore, the text shows the fundamentals of PID control and Optimal Control, as well as presenting the state-of-the-art use of both algorithms in drones.
- **Chapter 3:** This chapter shows the dynamic equations of the quadcopter. It shows how the nonlinear system is described using Ordinary Differential Equations (ODEs), and how it is obtained using the Newton-Euler formalism. The chapter also shows the procedures for estimating the system parameters using experimental data and Computer Aided Design (CAD) models.
- **Chapter 4:** This chapter addresses flight control software, the computational perspective of modeling, and the tools for practical testing. Initially, Ardupilot and its capabilities

for controlling autonomous vehicles are presented. Then, the chapter also demonstrates how a Model-in-the-Loop (MITL) was developed to translate both the mathematics of the dynamic model and the control firmware of Ardupilot into a block simulator using Simulink. Following this, the LQR/LQI model that will replace the traditional control is presented, along with the simulation of this new architecture. Finally, it shows how the drone was built and operated throughout the entire project.

- **Chapter 5:** This chapter presents both simulated and experimental results for operating an autonomous quadcopter using Ardupilot. The original PID controllers are systematically compared with the new architecture, both qualitatively through graphical analysis and quantitatively using performance metrics.
- **Chapter 6:** This chapter aims to present the main conclusions of the work and also suggests future perspectives for the advancement of open-source flight control firmware using state-of-the-art modern control technologies.
- **Appendix A:** This appendix shows all the characteristics of the constructed drone, including the parts used and the physical parameters measured or estimated.
- **Appendix B:** This appendix shows additional algebraic deductions that were omitted in the mathematical modeling chapter.
- **Appendix C:** This appendix shows the diagrams of the Simulink model developed for this work.

2

Fundamentals

This chapter covers the fundamentals of control theory that are important for understanding this work. The first topic is Classical Control, with an emphasis on PID, to understand the current Ardupilot control strategy. The second topic is Optimal Control, with emphasis on linear-quadratic methods, for developing the new control system for the autopilot. Finally, performance evaluation techniques for control systems are discussed.

2.1 PID and Classical Control Techniques

The Proportional-Integrative-Derivative controller is, despite many modern approaches in Control Theory, the most widely used algorithm in the field of automatic control.

In industry, [Aström e Hägglund \(1988\)](#) showed that 97% of the control loops by that time were controlled with the PID. The research included 11000 control loops in different kinds of industrial plants. According to The author (2026)s, these controllers were enough to satisfy simpler performance requirements. Facts like this created a solid culture of using PID in process control, and even today, it is the dominant algorithm in industrial equipment.

In terms of quadrotor control, the main manufacturers; such as DJI, Parrot, 3D Robotics and AscTec; are all using the classical PID controller ([LI; SUN; JIN, 2015](#)). When it comes to the main open-source UAV control firmwares, such as ArduPilot and PX4, they also utilize PID-based control strategies ([Ardupilot Development Team, 2024b](#); [PX4 Development Team, 2025](#)).

The PD, PI, and PID controllers have been studied since the early works about quadrotor control. [Bouabdallah, Noth e Siegwart \(2004\)](#) showed the success of the classical approach to stabilize the vehicle's attitude, using both simulations, a test bench, and a real vehicle. Roll, pitch, and yaw angles were controlled using three different control loops. The work, however, considered a simplified and linearized model to project the controller and only tested the vehicle

at hovering, thus just verifying disturbance rejection. Similarly, [Li e Li \(2011\)](#) presents the control of a quadcopter using only PID controllers, but with the difference that both attitude and position are controlled with three additional loops.

Many studies have shown PID control as a strategy for complete positioning and orientation during hovering, but most of them have only used classical approaches as part of a hybrid design or as a reference standard. [Hoffmann, Goddemeier e Bertram \(2010\)](#), for example, made a comparison between PID and Integral-Backstepping controllers for attitude stabilization.

Further studies also analyzed the PID for trajectory tracking. [Lima et al. \(2015\)](#) demonstrated the performance of PID controlling the six degrees of freedom of a quadcopter during trajectory tracking. It showed that attitude control achieves zero error in steady state, but the controllers experience a time lag in position tracking, especially in the xy plane. The study, however, was carried out entirely in a simulated environment, without analyzing the controller's response to disturbance rejection.

[Pinto et al. \(2019\)](#) evaluated the path tracking task of quadrotors using PID control, but it considered an experimental approach and used additional control loops. The research used a cascade control structure with 12 control loops including: angular speeds, attitude, linear speeds, and position. At the end, it showed the system's dynamic response in both simulations and a real aircraft. But even this study did not demonstrate a system based entirely on classical strategies that could encompass the full complexity of the ArduPilot control architecture for autonomous flight.

As a final observation, there are also studies on the difficulties in controlling helicopters under traditional control techniques. A study by [Hoffmann et al. \(2007\)](#), a seminal reference analyzing the aerodynamic effects on the UAV model, details the difficulties for the PID to control the quadcopter at higher speed flights.

It is therefore noticeable the importance of the PID both in the industrial context and in the focus of research on unmanned aerial vehicles. This popularity is justified by the algorithm's flexibility to fit in different applications. It can be applied in a black-box approach, without deep process knowledge and, nonetheless, achieve a satisfactory transient response and zero steady-state error.

It is also an algorithm of uncomplicated implementation. The PID control technique consists of properly applying the sum of three types of control actions: a proportional action, an integral action, and a derivative action.

2.1.1 The Proportional Action

The proportional control action is that one proportional to the current error signal according to a gain K_p . As the name suggests, it increases proportionally the control signal when

the error is large. The control signal is given by the Equation 2.1

$$u(t) = K_p e(t) \quad (2.1)$$

where K_p is the proportional gain and $e(t)$ the error signal.

The main drawback of a pure proportional controller is that it produces a steady-state error. In case a constant load disturbance is present, this kind of behavior occurs even in processes with integrating dynamics (VISIOLI, 2006b).

The increase in the proportional action directly affects the system's response time and the steady-state error value.

2.1.2 The Integral Action

The integral action acts proportionally to the integral of the error, as depicted in the Equation 2.2

$$u(t) = K_i \int_0^t e(\tau) d\tau \quad (2.2)$$

where K_i is the integral gain. And the integral action transfer function is:

$$C_i(s) = \frac{K_i}{s} \quad (2.3)$$

The integrator allows reducing the steady-state error to zero when a step input is applied or a step load disturbance occurs. Including integral action, even a small positive error will always lead to an increase in the control signal, or a decrease in the control signal for a negative error. Lastly, it is worth mentioning that the increase in the K_i intensifies the oscillatory behavior and the overshoot magnitude.

2.1.3 The Derivative Action

Contrary to the integral action that is based on the past values of the error, the derivative action is based on the predicted values of the control error. The derivative has the capability to improve control performance as it can anticipate an incorrect trend of the control error and counteract for it (VISIOLI, 2006b). The derivative action is described by the Equation 2.4.

$$u(t) = K_d \frac{de(t)}{dt} \quad (2.4)$$

where K_d is the derivative gain. The corresponding transfer function of the derivative is:

$$C_d(s) = K_d s. \quad (2.5)$$

There is, however, a critical consideration that makes this form of derivative action not applicable in practice. The high frequency gain of the pure derivative action is responsible for the amplification of the measurement noise in the manipulated variable. This high frequency noise will receive high gains and, thus, generate an extremely oscillatory control signal, that, in practice, can damage the actuators.

The problem shown above can be solved by adding a low-pass filter to the derivative action. The filter must be selected in order to eliminate most of the noise while not affecting the dominant dynamics of the system. Considering the addition of a first-order low-pass filter, the derivative's transfer function becomes:

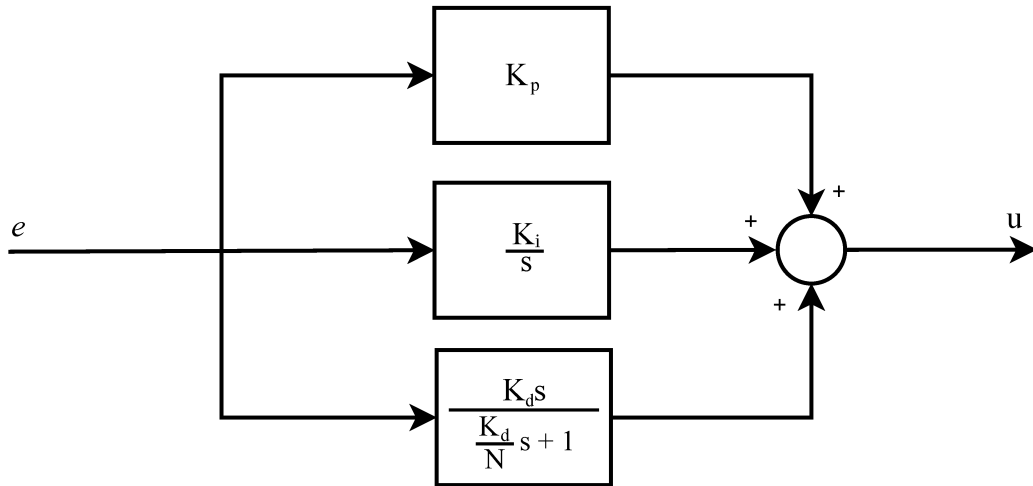
$$C_d(s) = \frac{K_d s}{\frac{K_d}{N} s + 1} \quad (2.6)$$

2.1.4 PID Control Structures

The most straightforward implementation is the parallel PID. This form is represented, in the time domain, using the simple sum Equation 2.7.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (2.7)$$

Figure 2 – Block diagram of a PID in parallel form.



Source: The author (2026).

Formulating the transfer function in S-domain and adding the derivative filter, the PID in parallel form is represented by Equation 2.8

$$C(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{\frac{K_d}{N} s + 1} \quad (2.8)$$

It can also be depicted by the diagram block in Figure 2.

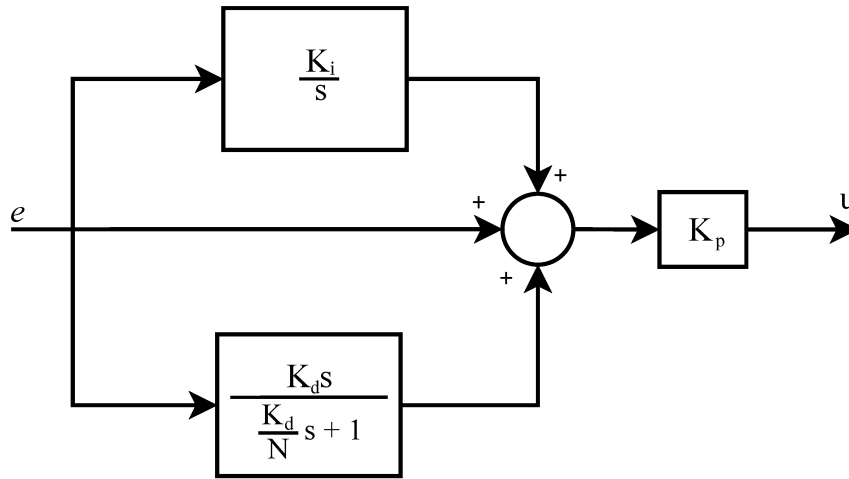
In this way, the three actions are completely decoupled. This allows for each of the control actions to be turned off separately.

Another implementation is the PID in ideal form. In this form, also called non-interacting form, the PID is described by the following transfer function of Equation 2.9 :

$$C(s) = K_p \left(\frac{1}{T_i s} + \frac{T_d s}{\frac{T_d}{N} s + 1} \right), \quad (2.9)$$

where K_p is the proportional gain, T_i is the integral time constant, and T_d is the derivative time constant. This form is shown in Figure 3.

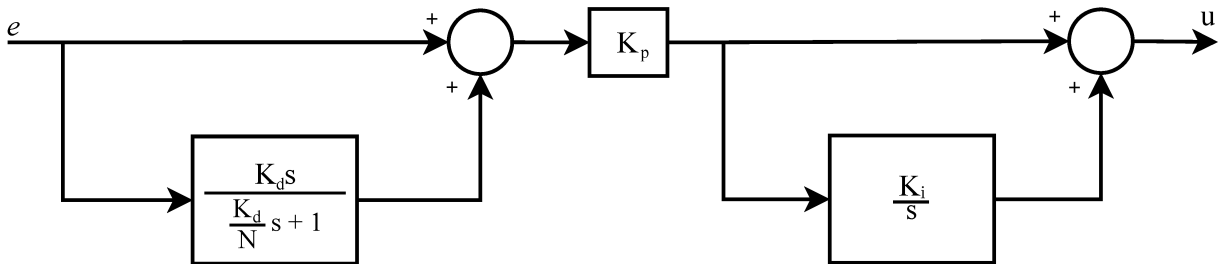
Figure 3 – Block diagram of a PID in ideal form.



Source: The author (2026).

Another common structure for PID controllers found in the literature is the series type. In this format, the derivative time constant interferes with the action caused by the integral time constant, and vice versa. For this reason, this structure is also referred to as *interacting form*. This strategy is represented by the transfer function presented in Equation 2.10 and depicted in Figure 4.

Figure 4 – Block diagram of a PID in series form.



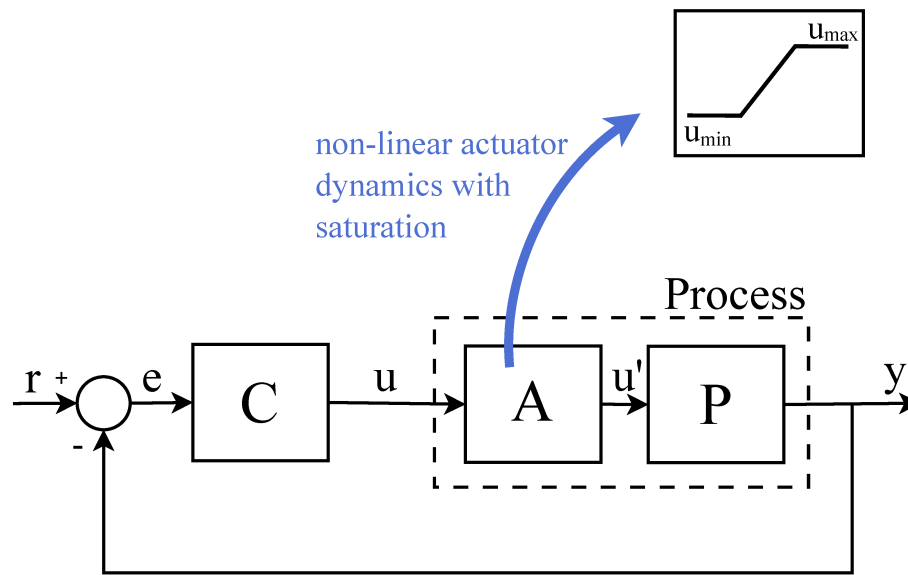
Source: The author (2026).

$$C(s) = K_p \left(1 + \frac{1}{T_i s} \right) \left(\frac{K_d s}{\frac{K_d}{N} s + 1} \right) \quad (2.10)$$

2.1.5 Integrator Anti-Windup

In real world applications, the actuator dynamics are included as part of the process model. For a wide operation point, the actuator dynamics might not strongly affect the system response. However, actuators have their physical limits, and in these operating regions they will exhibit a nonlinear behavior. Once saturated, the actuator can no longer provide the expected control input to the process. Figure 5 shows a control loop with saturating actuator dynamics.

Figure 5 – Representation of a closed-loop control system highlighting the actuator non-linear dynamics.



Source: The author (2026).

When the variable value reaches the actuator's maximum or minimum limit, the control signal saturates. In this case, the actuator will remain at its maximum (or minimum) limit regardless of the process output. If an integral-action controller is used, the error will continue to be integrated, and the integral term tends to become very large.

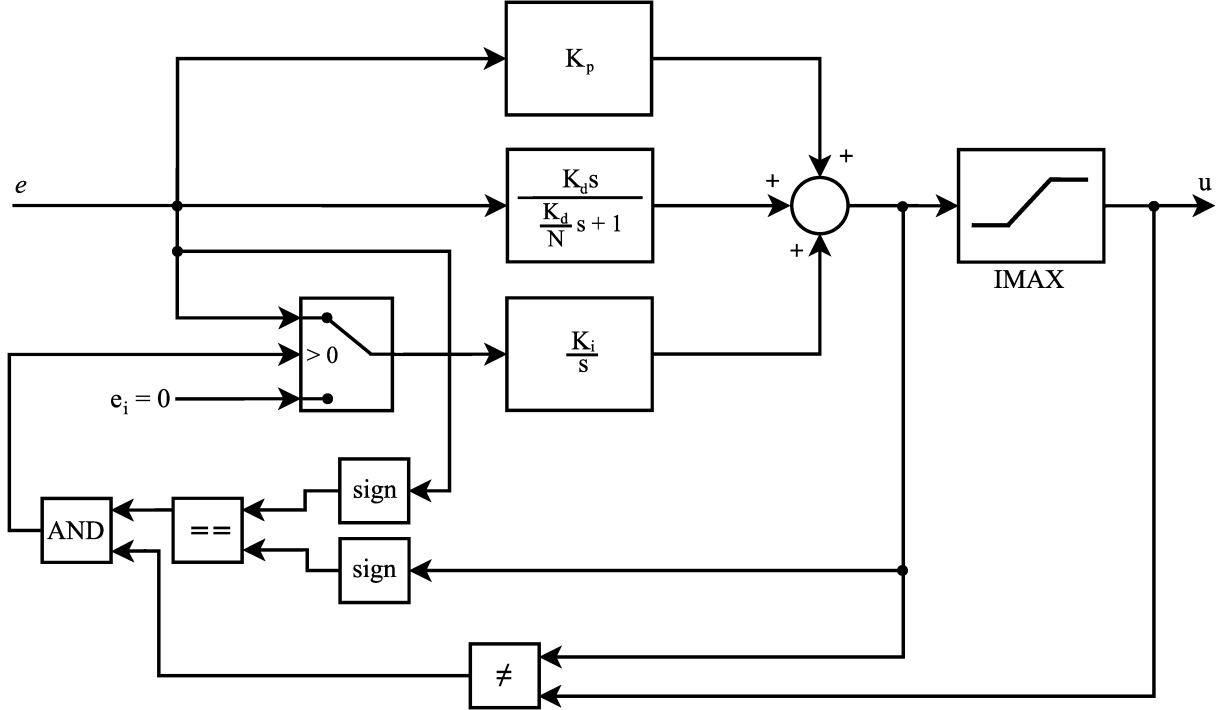
When this occurs, for the controller to return to linear operation (leave saturation), the integral term must discharge. To do so, the error signal must be allowed to change its sign and then apply it to the controller input for a long period of time. This results in intense overshoots and a slow, oscillatory transient response. This phenomenon is known as integrator windup.

The most intuitive way of avoiding integration windup is to avoid controls variable's saturation. A classical methodology, called Conditional Integration or Integrator Clamping, consists in switching off the integral action in certain conditions. Basically, this method sets the error to be integrated equal to zero.

A commonly applied Integrator Clamping technique is to stop integration when the control variable saturates and the control error and the control variable have the same sign. In this methodology, the integrator is not inhibited in cases where its action helps to push the control

variable away from saturation (VISIOLI, 2006a). The integrator clamping technique is shown in the diagram form in Figure 6.

Figure 6 – Block diagram of an anti-windup algorithm based on conditional integration.



Source: The author (2026).

2.1.6 PID Digital Implementation

A well-known approach for designing digital controllers is to discretize a previously designed continuous-time transfer function for that controller. There exist several methods for approximating a continuous-time controller into its digital version, given its transfer function.

Since a transfer function represents a differential equation, it is logical to obtain a difference equation approximating the derivatives with a forward difference (Euler's method)

$$\dot{x}(t) = \frac{dx(t)}{dt} \approx \frac{x(t+h) - x(t)}{h} = \frac{q-1}{h}x(t) \quad (2.11)$$

or a backwards difference

$$\dot{x}(t) = \frac{dx(t)}{dt} \approx \frac{x(t) - x(t-h)}{h} = \frac{q-1}{qh}x(t). \quad (2.12)$$

Another approximation, which corresponds to the trapezoidal method for numerical integration, is the *Tustin's Method*, also known as bilinear transform (ÅSTRÖM; WITTENMARK, 2011).

Given a continuous controller transfer function $C(s)$, the digital controller $C(z)$ is given by simply replacing the variable s by s' , where:

$$s' = \frac{z-1}{T} \quad (2.13)$$

for the Euler's method,

$$s' = \frac{z-1}{zT} \quad (2.14)$$

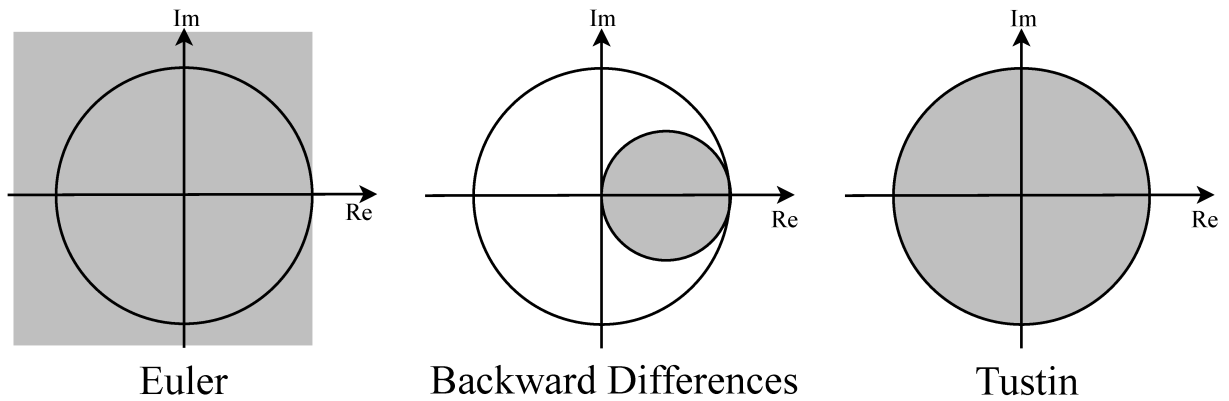
for the backward difference method, and

$$s' = \frac{2}{T} \cdot \frac{z-1}{z+1} \quad (2.15)$$

for Tustin's method. The variable T in these equations is the sample time of the discrete time system.

By using the Euler method, it is possible for stable continuous-time systems to be mapped into unstable discrete-time systems. With Backwards differences, a stable system in the S domain will always be stable in Z , however, some unstable continuous systems will become stable. With Tustin's method, the left half-plane S is transformed into the unit disk in the Z plane, representing a perfect mapping of the stability region between the domains. The mappings between the s -plane and the z -plane, for each discretization method, are shown in the Figure 7.

Figure 7 – Mapping of the stability region of s -plane on the z -plane for the transformations in Equations 2.13, 2.14 and 2.15.



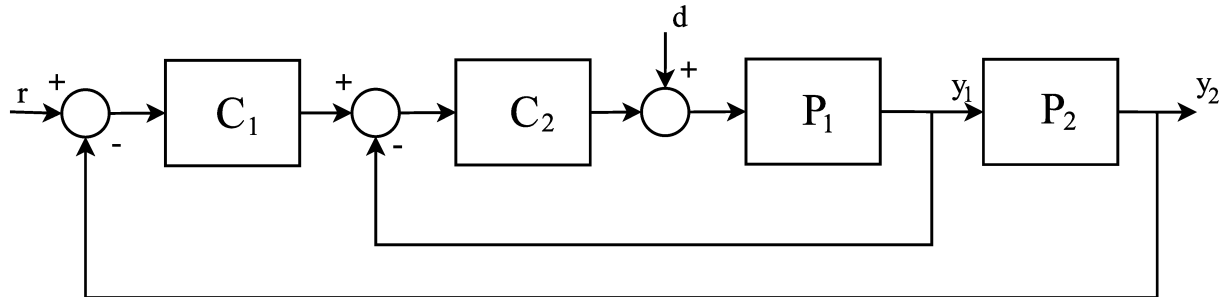
Source: Adapted from Åström e Wittenmark (2011).

2.1.7 Cascade Control

A typical and simplified cascade control system is depicted in Figure 8. The process transfer function is denoted by $P(s) = P_1(s)P_2(s)$, and the outputs y_1 and y_2 are the primary and secondary outputs, respectively. In this context, C_1 is the primary (or master) controller, and C_2 is the secondary (or slave) controller. Evidently, the output signal of the master controller serves as a setpoint for the slave controller.

The intention with this type of control structure is to isolate the slow dynamics of the process ($P_1(s)$) from the fast dynamics ($P_2(s)$). The effectiveness of the cascade control system is that disturbances affecting the secondary loop are compensated before they affect the main process output y_2 (VISIOLI, 2006c).

Figure 8 – Block diagram of typical cascade control structure.



Source: The author (2026).

If applied properly, the cascade control system has improved stability characteristics and therefore a higher gain in the primary loop can be adopted. Since the inner loops represent the control of faster dynamics, in a cascade structure the inner controller must always have a higher sampling rate than the outer ones.

2.1.8 Feedforward Action in Motion Control

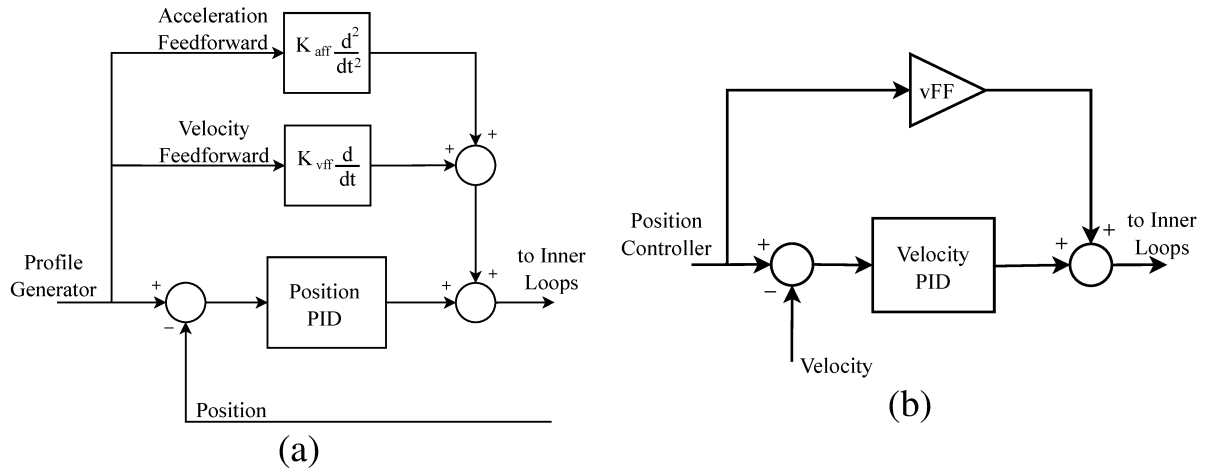
In servo or UAV systems, tracking errors occur during movement, even if the integrator sets this error to zero after reaching the desired waypoint. A common approach in industrial motion control systems is to combine classical feedback techniques with feedforward control. By injecting a velocity-proportional command directly into the output, the system responds faster to changes in movement compared to the feedback-driven counterpart of the system, which only acts after the existence of a position error.

One of the ways to apply feedforward control, and the most seen in motion control literature, is getting the derivative commands from motion profile, multiplying by predetermined gains and adding it to the position control signal. This work uses Gurocak (2016) as reference for this type of structure, shown in Figure 9 (a).

When the speed profile input is feedforward to the speed controller, the system responds faster to sudden changes in the desired trajectory. However, if the gain is set to a high value, the feedforward will create overshoots in the speed response.

One way to solve the overshoot caused by the velocity feedforward is by adding an additional feedforward action proportional to the second-order derivative of position. This so called acceleration feedforward eliminates the overshoots in speed while keeping the position tracking improvement.

Figure 9 – Common feedforward control approaches where (a) is the motion control approach and (b) is the 2-DoF control system.



Source: Adapted from [Gurocak \(2016\)](#).

Since servo errors can happen even in the jerk phases of the motion, further feedforward gains can be added. Jerk and jerk derivative feedforward are less common, but also used, for example in ultraprecision motion control applications.

Another philosophy for approaching the application of feedforward control is to treat it not as an anticipator of the position trajectory, but as a predictor of the control effort required. This scheme is shown in Figure 9 (b).

If disturbances are known to occur in a system, a feedforward term can be added to help the system reach the desired setpoints with less delay and less dependence on the integral term. This can be done through physical modeling of the disturbance, applying inverse transfer functions of the plant, or simply adding constant gains. This control technique is known as 2 degrees-of-freedom (2-DOF) PID, since it creates parallel paths to handle both disturbance rejection and setpoint following.

2.2 Classical Optimal Control Strategies

At the time of the technological expansion that allowed the popularization of multirotor drones, linear Optimal Control techniques had already been thoroughly explored in academia. Furthermore, these techniques already had a long and successful application in the aerospace sector, as exemplified by their use in numerous satellites and Boeing aircrafts. Therefore, the use of Optimal Control for quadcopter control seems undeniable, and it promises real improvements compared to the use of PIDs.

Indeed, and for such reasons, techniques like LQR have also been explored since the beginning of the spread of drones for civilian uses. The pioneering work of [Bouabdallah, Noth e Siegwart \(2004\)](#), which studied PIDs for the control of microquadrotors at hover, also analyzed

the application of Linear-Quadratic control methods. Another early work on LQR for quadcopters, by [Cowling et al. \(2007\)](#), analyzed this controller for path following. Since the LQR is projected around one specific operating point, the responsibility for maintaining stability was transferred from the low-level controller to the trajectory generator.

The study conducted by [Martins, Cardeira e Oliveira \(2019\)](#) applies LQI control to a drone in trajectory tracking. In this work, state feedback is used to control the position and attitude. Furthermore, integral action is employed to correct for perturbations or parametric uncertainties. However, the study only considers the linearized system around the hover operating point, and therefore the maximum linear and angular velocities of the trajectory are 0.05 m/s and 0.05π rad/s, respectively.

Other works, like [Okasha et al. \(2017\)](#), address the problem of the LQR's inability to handle nonlinearities by applying gain scheduling. In this work, a gain schedule is created based on the variation of the yaw angle. The study shows that a fixed LQR makes it impossible to the drone to perform helical trajectories with varying yaw, but that a gain schedule with interpolation allows the trajectory to be followed smoothly. The study also reports the need for integral gain, since despite being stable, the trajectory had steady-state errors.

The work of [Joukhadar et al. \(2019\)](#) goes further and applies LQI associated with feedback linearization control. The nonlinear feedback linearization technique was used to actively cancel the coupled dynamics (such as the gyroscopic effects of the motors and inertial couplings) inherent in the quadcopter's state variables. By canceling the nonlinear component, the system was mathematically transformed into a simplified linear model. It is on this equivalent model that the LQI controller acts.

Beyond the literature discussed above, there are numerous other studies applying linear-quadratic optimal control methods. Many explore the limitations of simpler implementations to create more robust, stable, and higher-performance solutions. The state of the art of optimal controllers for unmanned aerial vehicles is vast, and the relevance of studying practical implementations of such controllers in real quadcopters is irrefutable.

2.2.1 Linear State Feedback Control Laws

A linear system in state-space representation is described by the pair of equations:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) + Du(t).\end{aligned}\tag{2.16}$$

where $A \in \mathbb{R}^{n \times n}$ is the state matrix, $B \in \mathbb{R}^{n \times m}$ the input matrix, $C \in \mathbb{R}^{p \times n}$ the output matrix, and $D \in \mathbb{R}^{p \times m}$ the forward transfer matrix. The numbers n , m and p are equivalent to the number of states, inputs and outputs respectively. Then, the feedback state feedback control law is:

$$u(t) = -Kx(t) + r(t)\tag{2.17}$$

where K is a $m \times n$ gain matrix and $r(t)$ the reference input. Figure 10 shows the block diagram of the state feedback control law.

If the external reference input is absent, the state feedback control law is called a *regulator* that is designed to deliver desired transient response for nonzero initial conditions and/or reject disturbances to maintain the equilibrium state $\tilde{x} = 0$, where $\tilde{x}(t) = 0$ denotes the deviation from the equilibrium point.

This feedback law has the objective to reach a predetermined performance for the closed-loop system

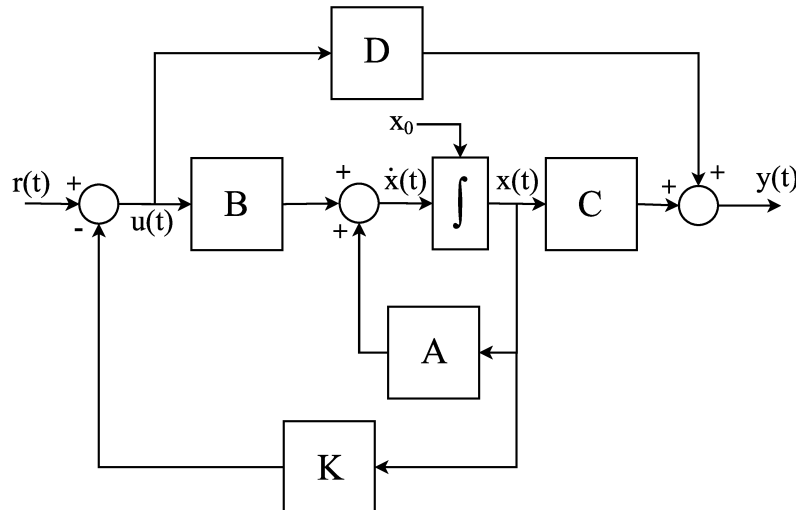
$$\begin{aligned}\dot{x}(t) &= (A - BK)x(t) + Br(t) \\ y(t) &= Cx(t) + Du(t).\end{aligned}$$

It is possible to shape the closed-loop transient response of the system from eigenvalue placement of the matrix $A - BK$. The desired transient response characteristic are the same used in classic control: rising time, peak time, percent overshoot and settling time (for a step-response).

It exists analytical methods for arbitrary eigenvalue placement in ways to reach the desired response characteristics for the state feedback control law. For example the eigenvalues can be placed using: the Controller Canonical Form, the Bass-Gura Formula, the Ackermann's formula, or computationally using the MATLAB `place` function.

A pure state feedback control law can influence the transiente response, but not the steady state characteristics of the closed-loop system. The best way to accomplish the steady-state requirements is revisiting a concept from classical control: zero steady-state tracking error can be achieved for *type I* systems that include integral action on the tracking error. This control system, often refered as *servomechanisms*, adds a integral and a gain multiplying the reference

Figure 10 – Block diagram of a linear system state-space representation with state feedback control.



Source: The author (2026).

input. This ensures that both transient and steady-state response characteristics are achieved, plus it adds robustness to the steady-state value against open-loop state space equations uncertainty. Figure 11 shows the block diagram of the state feedback tracker.

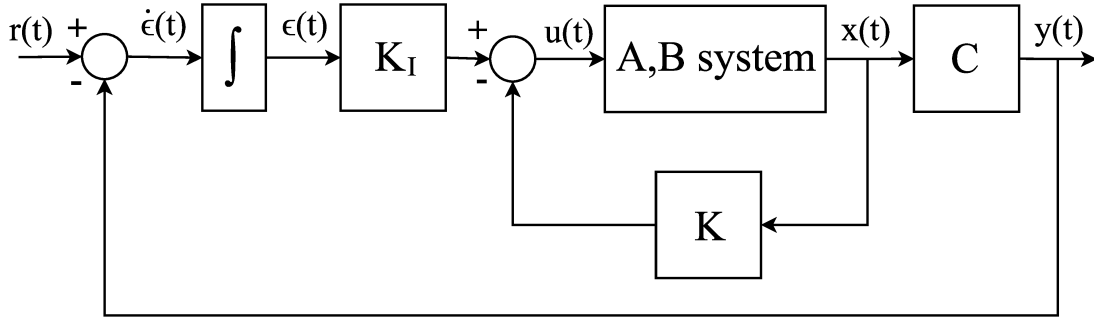
The control law, in this case will be of the form:

$$\dot{\epsilon}(t) = r(t) - y(t)$$

$$u(t) = -Kx(t) + k_I \epsilon(t)$$

$$u(t) = - \begin{bmatrix} K & -k_I \end{bmatrix} \begin{bmatrix} x(t) \\ \epsilon(t) \end{bmatrix}$$

Figure 11 – Block diagram of the state feedback tracking system.



Source: The author (2026).

In this scenario the input $u(t)$ is interpreted as a state-feedback control law with an augmented states vector of dimension $(n + 1)$ that includes both the open-loop states $x(t)$ and the integrator state vector $\epsilon(t)$. Considering the direct transfer matrix D to be null, the closed-loop state equation that the desired eigenvalue placement is performed will be:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{\epsilon}(t) \end{bmatrix} = \begin{bmatrix} A - BK & Bk_I \\ -C & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ \epsilon(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r(t)$$

$$y(t) = \begin{bmatrix} C & 0 \end{bmatrix} \begin{bmatrix} x(t) \\ \epsilon(t) \end{bmatrix}$$

To place desired closed-loop eigenvalues, in this case, it is necessary that the pair

$$\left(\begin{bmatrix} A & 0 \\ -C & 0 \end{bmatrix}, \begin{bmatrix} B \\ 0 \end{bmatrix} \right)$$

to be controllable. Since the condition is satisfied, it is possible to place $\begin{bmatrix} K & -k_I \end{bmatrix}$ using Bass-Gura formula, Ackermann's formula or MATLAB's place function.

2.2.2 A General Formulation for the Optimal Control Problem

Any optimal control strategy can be described by using a general formulation. At first, let the nonlinear state equation of a system be defined as

$$\dot{x}(t) = f(x(t), u(t), t), \quad x(t_0) = t_0, \quad (2.18)$$

then exists a performance index, cost function or objective function to be minimized, that equals to

$$J = \int_{t_0}^{t_f} L(x(t), u(t), t) dt + \varphi(x(t_f), t_f). \quad (2.19)$$

Here, $L(x(t), u(t), t)$ is called the loss function and penalizes the states or inputs of the system depending on time t . In addition, the $\varphi(x(t_f), t_f)$ term is the states' terminal cost.

The optimization problem is to determine an optimal state path and input signal $(x^*(t), u^*(t))$ that minimize the performance index, subject to constraints and initial states. This, then, is the problem of regulation: maintaining the trajectory close to the original equilibrium while expending a reasonable amount of control effort.

For a generic optimal control problem the optimal solution $(x^*(t), u^*(t))$ is obtained and verified using mathematical tools such like the Pontryagin's Minimum Principle, variational calculus and dynamic programming. These equations are difficult to solve and further optimality verification requires nontrivial analysis. Then, this work may be focused only in the particular case of Linear-Quadratic methods.

2.2.3 Linear-Quadratic (LQ) Techniques in Optimal Control

In most cases of applying optimal control it is considered the linear time invariant state-space equation 2.26 and seek to minimize the quadratic performace index

$$J = \frac{1}{2} \int_{t_0}^{t_f} [x^T(t) Q x(t) + u^T(t) R u(t)] dt + \frac{1}{2} x^T(t_f) S x(t_f), \quad (2.20)$$

in which the loss function is

$$L(x, u) = \frac{1}{2} (x^T(t) Q x(t) + u^T(t) R u(t)) = \frac{1}{2} \begin{bmatrix} x^T & u^T \end{bmatrix} \begin{bmatrix} Q & 0 \\ 0 & R \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}$$

and the terminal cost function is $\varphi(x) = x^T S x$, and both of them are in quadratic forms. The matrices $Q \in \mathbb{R}^{n \times n}$, $R \in \mathbb{R}^{m \times m}$ and $S \in \mathbb{R}^{n \times n}$ are weighting matrices, all of them symmetric, with Q and S positive semidefinite and R positive definite. Q is responsible for the performance weighting in state control, R is the weighting matrix for control effort, and S is the terminal state weighting.

This case, where the goal is to minimize a quadratic performance index, is commonly called a Linear Quadratic Regulator (LQR) problem. This particular case of optimal control

admits an analytical solution. As demonstrated in Williams e Lawrence (2007), with the solution of the optimization problem it is possible to determine the inputs for the linear system to follow an optimal trajectory, and apply a state feedback control law $u(t) = -Kx(t)$, where the time-varying gain matrix will be:

$$K(t) = -R^{-1}B^T P(t), \quad (2.21)$$

where $P(t)$ is a matrix defined for each time $t \in [t_0, t_f]$, and is obtained by solving the well-known Riccati differential equation:

$$-\dot{P}(t) = A^T P(t) + P(t)A - P(t)BR^{-1}B^T P(t) + Q \quad (2.22)$$

with condition $P(t_f) = S$.

One can consider an even more particular case for optimal control: the scenario where the LQR has an infinite horizon. This case can be considered as the linear steady-state LQR, and it is interpreted as the same problem as equation 2.23, but now with $t_0 = 0$, $S = 0$ and $t_f \rightarrow \infty$, in which the performance index become:

$$J = \frac{1}{2} \int_0^\infty \left[x^T(t)Qx(t) + u^T(t)Ru(t) \right] dt \quad (2.23)$$

As the time goes to infinity the Riccati differential equation can exhibit different behaviors. It can be unbounded, or it can converge to a limiting solution $P(\infty)$, which can be zero, positive semi-definite, or positive definite. Lets consider the cases where $P(t)$ converges and then $\dot{P}(t) = 0$. As the final time tends to infinity the differential Riccati equation solution tends to a steady-state value $\bar{P}$. It means that it results an equilibrium solution represented by the time-invariant matrix $\bar{P}$, and the equation, now referred to as the *Algebraic Ricatti Equation* (ARE), is:

$$A^T \bar{P} + \bar{P}A - \bar{P}BR^{-1}\bar{P} + Q = 0 \quad (2.24)$$

Consequently, solving this algebraic equation now produces a time-invariant matrix $K = -R^{-1}B^T \bar{P}$ for applying the state feedback control law. This approach is used in the vast majority of cases, as it guarantees stability and results in greater computational simplicity for embedded systems.

The Linear Quadratic Regulator can be transformed into a tracking system by adding the augmented state matrix to the performance index. In this case, we have an LQI type control system, which will follow the same gain application scheme as the servomechanism shown in the Figure 11, with a performance index defined by Equation 2.25,

$$J = \int_0^\infty \left\{ \begin{bmatrix} x^T(t) & \varepsilon^T(t) \end{bmatrix} Q_{aug} \begin{bmatrix} x(t) \\ \varepsilon(t) \end{bmatrix} + u^T Ru \right\} dt \quad (2.25)$$

where Q_{aug} is the augmented weighting matrix.

2.2.4 Digital Implementation of *Steady-State* LQR and LQI Controllers

A discrete-time linear system in state-space is described by the pair of equations:

$$\begin{aligned} x(k+1) &= A_d x(k) + B_d u(k) \\ y(k) &= C_d x(k) + D_d u(k). \end{aligned} \quad (2.26)$$

Starting from a plant in continuous state space, the following matrix operations are used to discretize it using Euler's method:

$$\begin{aligned} A_d &= e^{AT} \\ B_d &= \left(\int_0^T e^{A\tau} d\tau \right) B \end{aligned}$$

The matrices of the output equation remains the same: $C_d = C$ and $D_d = D$.

Once the system is discretized, the optimization problem can be developed again for the discrete-time domain. Therefore, the steady-state LQR performance index is rewritten in the form of the summation of Equation 2.27:

$$J = \frac{1}{2} \sum_{k=0}^{\infty} \left[x^T(k) Q x(k) + u^T(k) R u(k) \right]. \quad (2.27)$$

In this new scenario, the gain matrix for state feedback control is:

$$K = (R + B_d^T \bar{P} B_d)^{-1} B_d^T \bar{P} A_d, \quad (2.28)$$

where $\bar{P}$ is the solution of the Discrete-Time Algebraic Riccati Equation (DARE), defined in Equation 2.29:

$$0 = A_d^T \bar{P} A_d - \bar{P} + Q - A_d^T \bar{P} B_d (R + B_d^T \bar{P} B_d)^{-1} B_d^T \bar{P} A_d \quad (2.29)$$

For a solution to the DARE to exist, the matrix pair (A, B) must be stabilizable and the pair (A, Q) must be detectable.

2.3 Performance Assessment of Control Systems

The performance assessment of a control loop can be achieved using a set of mathematical tools that can be either stochastic or deterministic. In this work, the deterministic performance assessment will be used. These latter sets of techniques are better suited to more traditional design specifications of control systems, such as set-point tracking and load disturbance rejection,

and how the controller achieves these tasks. These evaluations will be useful for comparing well-tuned and poorly tuned controllers, as well as for verifying the improvement in the application of optimal controller designs compared to classical designs.

Evaluating the performance of a control system is a complex task that must follow a basic sequence of actions. Initially, a performance index, arbitrarily defined, is calculated from the available data to quantify the efficiency of the system in operation. Next, a reference value, or benchmark, is defined that represents the desired performance, allowing current performance to be compared to this standard.

Using this logic, it is possible to draw up a metric to evaluate the quality of the controller that is not based solely on observation. Different approaches can be adopted at each stage, and various control system characteristics can be assessed. These steps must be integrated coherently to form a consistent overall strategy. Thus, the metrics selected here will be those substantial considering the flight control of a quadrotor.

To quantitatively evaluate the performance of the quadcopter control systems, three main performance indexes are taken: the Integral of Absolute Error (IAE), the Integral Squared Input (ISU), and Total Variation (TV).

The IAE measures the total absolute deviation from the route. Unlike other performance indices, it does not concern itself with when the deviation occurs or the intensity of the peaks. It is defined by Equation 2.30.

$$IAE = \int_0^{\infty} |e(t)| dt \quad (2.30)$$

It is important not only to measure the output error but also to evaluate the control effort. In this sense, the chosen index is the ISU shown in Equation 2.31, which physically represents the energy spent to operate the system. It is also time-independent, but unlike the IAE, it penalizes the intensity of peaks more because it includes the squared error. In this case, the chosen index is squared because the power consumption in an electrical system (such as BLDC motors) as a function of the controlled variable is quadratic. The combination of IAE and ISU was used to compare optimal control systems for UAVs in (AHMDIA et al., 2025).

$$ISU = \int_0^{\infty} u(t)^2 dt \quad (2.31)$$

Finally, TV, shown in equation 2.32, quantifies the rate of change of the inputs. Physically, it represents how fast the actuators have to operate to deliver the inputs required by the control system. Systems with a high TV tend not to respect the physical limits of the actuator dynamics, and also represent greater wear on these components.

$$TV = \int_0^{\infty} \left| \frac{du(t)}{dt} \right| dt \quad (2.32)$$

Mathematical Modeling

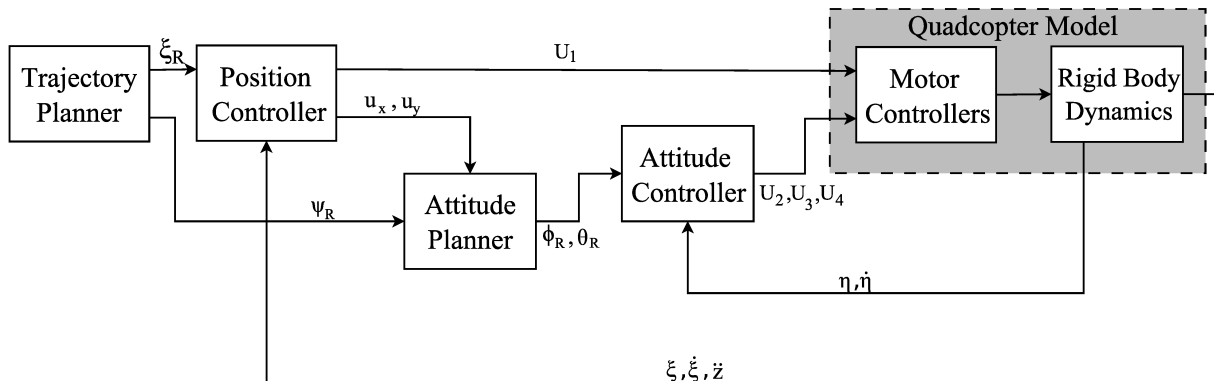
This chapter presents the complete modeling of a quadcopter. The first section introduces the transformations that relate reference frames. Section 3.3 presents the basic dynamics of the system deduced from the Newton-Euler formalism, and then Section 3.4 develops the nonlinear models of the plant. Section 3.5 shows experimental procedures to obtain the system's parameters.

3.1 The quadcopter from the perspective of Control Systems

The trajectory tracking task is challenging because the vehicle must follow a desired trajectory in six-dimensional space ($SE(3)$), despite having only four independent control inputs available. As a result, the quadrotor is inherently underactuated. The system's inputs are a thrust force U_1 , a roll torque U_2 , a pitch torque U_3 , and a yaw torque U_4 .

It is common to adopt a cascade control structure, as the one shown in Figure 12, to

Figure 12 – Overview of the multirotor UAV control loops: the outer position control loop, the intermediate attitude control loop, and the innermost motors control loop.



Source: Adapted from Mahony, Kumar e Corke (2012).

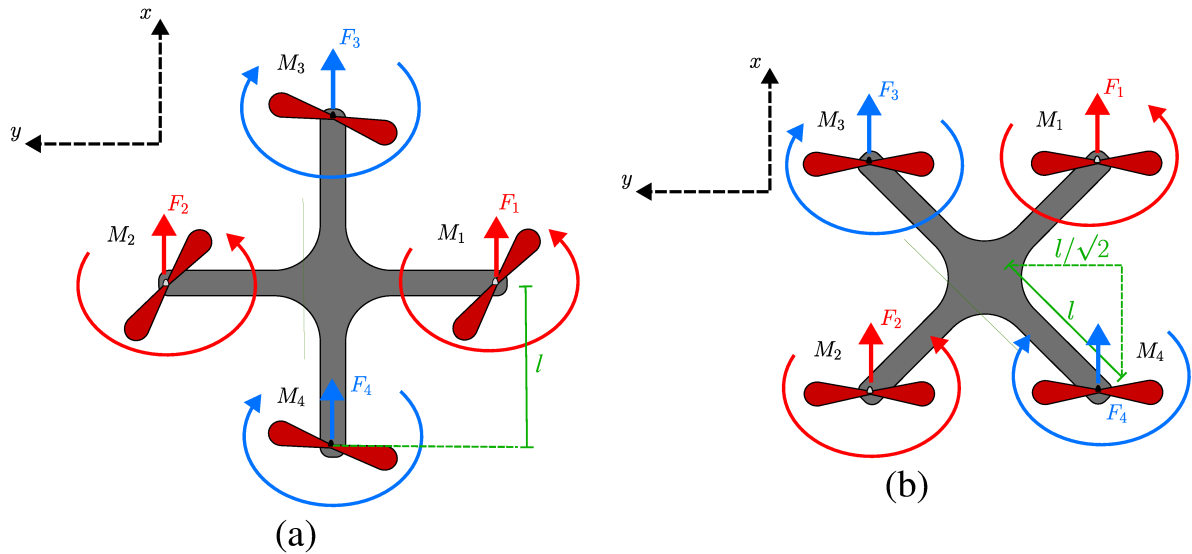
deal with different subsystem dynamics, each with a distinct response time. The highest level of this structure is the translational motion, which has the lowest bandwidth. Next, and with faster response times respectively, there is attitude control, and the rotation speed control of the BLDC motors (MAHONY; KUMAR; CORKE, 2012).

The only control loops performed outside the flight controller are those responsible for the motor angular speeds. For this task, usually, there are dedicated circuits named Electronic Speed Controllers (ESCs). The simplified control structure is shown in Figure 12.

The roll and pitch motions are coupled with the linear motion in the xy plane. The quadcopter tilt angles cause the propellers' force to be decomposed into vertical and horizontal components. Therefore, the control inputs for horizontal motion from the position controller can be used as setpoints, ϕ_R and θ_R , for the attitude roll and pitch controllers. The *Attitude Planner* block then, by trigonometric operations, transforms the virtual control inputs of the linear subsystem u_x and u_y into tilt angles for the next control loops.

The main control inputs to the quadcopter are generated by the power distribution between the vehicle's motors. The thrust signal (U_1) is given by the sum of the upward forces of the four motors. The roll signal is generated by the difference in thrust between motors M1 and M2 (in the plus configuration, as shown in Figure 13.(a)) or the M1-M4 and M2-M3 pairs (in the X configuration, as shown in Figure 13.(b)). Similarly, the pitch signal is generated by the difference in thrust between motors M3 and M4 (in the plus configuration) or the M1-M3 and M2-M4 pairs (in the X configuration).

Figure 13 – Quadrotor UAV in (a) *plus* and (b) *X* configurations. The scheme shows the motors numbering accordingly with the Ardupilot standard.



Source: The author (2026).

Finally, the yaw signal is created by the imbalance in the rotations of pairs of motors that rotate in the same direction. In both configurations, the yaw moment is created by the difference

in rotations between the M1-M2 and M3-M4 pairs. Yaw torque is generated due to the principle of conservation of angular momentum.

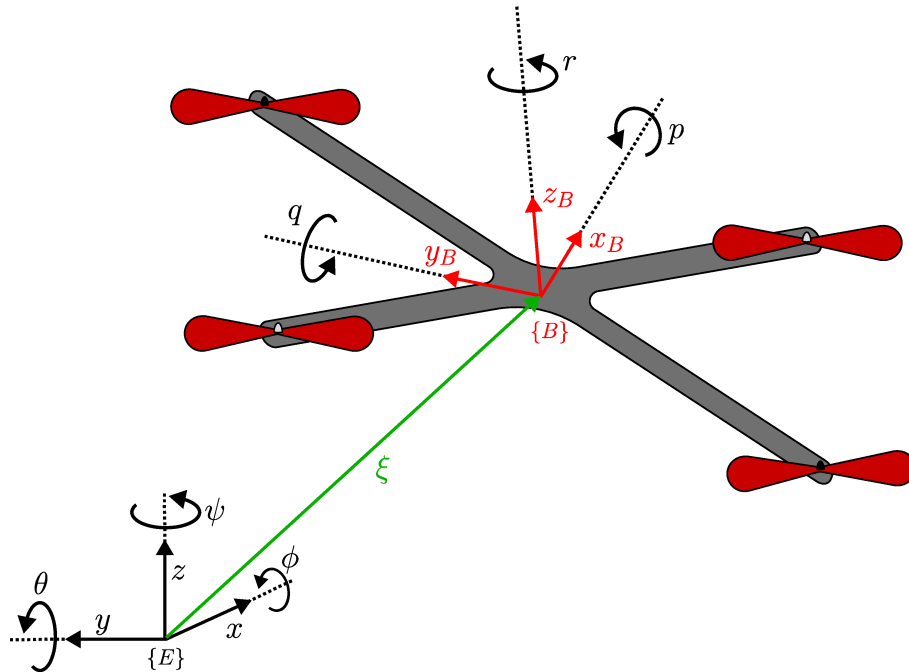
3.2 Coordinate Frames

The first step in creating a dynamic model of a aerial robot, such as a UAV, is to represent it as a 6-DOF rigid body. To properly locate a rigid body in space, it is necessary to specify a point in it (three Cartesian coordinates) and the object's orientation (three rotation angles) (NIKU, 2020). These are the six essential pieces of information to define the UAV.

This rigid body can be represented with a coordinate frame attached to it. Since the UAV is always attached to this frame, its position and orientation relative to the frame are always known. This coordinate frame B , fixed to the rigid body, is named the body frame (or non-inertial frame). Subsequently, it is necessary to relate the moving frame B to a fixed frame, relative to a fixed point on the Earth. This last coordinate frame E is named the Earth frame (or inertial frame).

The fixed coordinate system to the quadcopter is defined as $B = \{\vec{x}_b, \vec{y}_b, \vec{z}_b\}$. In which, $\vec{x}_b$ is the flight direction axis, $\vec{y}_b$ is orthogonal to $\vec{x}_b$ with the negative direction towards starboard, and $\vec{z}_b$ is oriented upwards and orthogonal to the $\vec{x}_b O \vec{y}_b$ plane. Meanwhile, the Earth frame is defined as $E = \{\vec{x}, \vec{y}, \vec{z}\}$ (LIMA, 2015). The relation between coordinate frames is exhibited in Figure 14.

Figure 14 – Coordinate frames of a quadrotor UAV.



Source: The author (2026).

A popular method to represent a coordinate frame rotation is through homogeneous

transformation matrices. A transformation is defined as the act of making a movement in space. The orientation of a rigid body can be fully determined after a sequence of three consecutive rotations.

Niku (2020) demonstrates that to rotate a moving frame around the x-axis of the inertial reference frame, the body frame must be pre-multiplied by a R_x rotation matrix, as shown in Equation 3.1. Let ϕ , θ and ψ be the roll, pitch and yaw angles respectively.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & -\sin \phi \\ 0 & \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}. \quad (3.1)$$

Similarly, Equation 3.2 gives the rotation around the y-axis of the inertial frame by pre-multiplying the rotating frame by a matrix R_y :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}, \quad (3.2)$$

and equation 3.3 the rotation around the z-axis of the inertial frame by pre-multiplying the rotating frame by a matrix R_z :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}. \quad (3.3)$$

There are, in total, 12 possible rotation sequences to determine the orientation of a rigid body. The set of three rotations to define a body rotation in three-dimensional Euclidean space is often called Euler Angles, but this denomination is often conflictual in literature (BOUABDALLAH, 2007). The Tait-Bryan rotations, also called nautical angles, are the most often adopted rotation pattern using the Euler Angles, and consist of the sequential x-y-z rotation. The R_{xyz} , or Tait-Bryan rotations, is the only rotation sequence adopted during this work, and it is defined by Equation 3.4.

$$R_{xyz} = R_z(\psi)R_y(\theta)R_x(\phi) \quad (3.4)$$

It is important to mention that sequences of rotations relative to an inertial reference will be different from the same sequences relative to a moving reference. In the case of rotations relative to the body frame, the transformation matrices will be post-multiplied. Thus, for example, the rotation around the x-y-z axes of the inertial reference frame will be equal to the z-y-x rotation of the non-inertial reference frame:

$$R_{xyz}(\phi, \theta, \psi) = R_{z_b y_b x_b}(\psi, \theta, \phi).$$

Then, the final transformation that relates the Earth frame with the body frame is defined by the matrix in Equation 3.5. For the sake of simplicity, the capital letters S_a and C_a are used as abbreviations for the sine and cosine of an angle a .

$$R_{xyz}(\phi, \theta, \psi) = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi - S\psi C\phi & C\psi S\theta C\phi + S\psi S\phi \\ S\psi C\theta & S\psi S\theta S\phi - C\psi C\phi & S\psi S\theta C\phi - C\psi S\phi \\ -S\theta & C\theta S\phi & C\theta C\phi \end{bmatrix} \quad (3.5)$$

The rates of change of the Euler angles, however, are not equal to the angular velocities referenced to the body frame. For example, for the R_{xyz} sequence, the roll angle (applied first) will not be around the final roll axis of the body frame, as the pitch and yaw rotations have already moved these axes. In the same way, the rate of change of pitch (second rotation applied) will not be around the final pitch axis, as the yaw rotation has moved this axis.

Thus, the idea is to consider the effect of these changes on the rotation vector:

$$\vec{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \Rightarrow \vec{\omega} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R_x \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R_x R_y \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}.$$

The change rates of the Euler angles are usually taken, as shown in the survey by Chovancová et al. (2014), using the inverse transformation T , derived from the previous assumptions, in which

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = T \begin{bmatrix} p \\ q \\ r \end{bmatrix}.$$

and p , q and r are the change of rate of roll, pitch and yaw angles in the body reference frame.

Finally, the matrix T is defined as in Equation 3.6.

$$T = \begin{bmatrix} 1 & \sin \phi \tan \theta & \cos \phi \tan \theta \\ 0 & \cos \phi & -\sin \phi \\ 0 & \frac{\sin \phi}{\cos \theta} & \frac{\cos \phi}{\cos \theta} \end{bmatrix} \quad (3.6)$$

In order to formalize the definitions of position and orientation vectors in the inertial and non-inertial frames, some definitions are necessary. Let the linear and angular velocity vectors in the body frame be $\vec{V}^B = \begin{bmatrix} u & v & w \end{bmatrix}^T$ and, $\vec{\omega}^B = \begin{bmatrix} p & q & r \end{bmatrix}^T$ respectively. The generalized velocities vector $\vec{v}$ that contains all velocities in the body frame will be $\vec{v} = \begin{bmatrix} \vec{V}^B & \vec{\omega}^B \end{bmatrix}^T$.

Similarly, let the linear position and attitude in the inertial frame be defined as $\xi = \begin{bmatrix} x & y & z \end{bmatrix}^T$ and $\eta = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$, respectively. The general vector position and attitude in the fixed frame are then grouped in the vector $\vec{q} = \begin{bmatrix} \vec{\xi} & \vec{\eta} \end{bmatrix}^T$

The UAV position change rate relative to the Earth frame is defined by the product of the R_{xyz} transformation matrix and the linear velocities vector as in Equation 3.7.

$$\dot{\vec{\xi}} = R_{xyz} \vec{V}^B \quad (3.7)$$

The UAV's angle rate of change relative to the Earth frame is defined by the product of the matrix T and the body frame angular speeds as in Equation 3.8.

$$\dot{\vec{\eta}} = T \vec{\omega}^B. \quad (3.8)$$

Using the Equations 3.7 and 3.8, it is possible to get a transformation that relates all the velocities in the Earth frame $\dot{q}$ with all the UAV's measurements in the non-inertial frame of reference. Equation 3.9 describes this beforementioned transformation.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} R_{xyz} & 0_{3 \times 3} \\ 0_{3 \times 3} & T \end{bmatrix} \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} \quad (3.9)$$

3.3 Newton-Euler Formalism

The Newton-Euler formalism defines that the equations of motion of a rigid body can be defined using the linear and the rotational versions of Newton's Second Law. Before introducing the complete equation, it's important to mention the role of the moment of inertia in the model. Depending on the considerations adopted, the model may not be compatible with the real scenario.

It is known from Newton's Second Law of rotation that the torque $\vec{\tau}$ on a rigid body is defined by the product of the moment of inertia J and the angular acceleration $\ddot{\vec{\omega}}^B$:

$$\vec{\tau} = J \ddot{\vec{\omega}}^B \quad (3.10)$$

For a three-dimensional rigid body, this equation may be represented in a matrix form. The quantities I_{xx} , I_{yy} and I_{zz} are called the moments of inertia of the body about the respective axes, and I_{xy} , I_{xz} , I_{yz} are the products of inertia with respect to the coordinate axes. These

quantities describe the way in which the mass of a rigid body is distributed in relation to the axes of the moving reference frame (MERIAM; KRAIGE, 2013). More broadly, the moments, using the moment of inertia matrix, or inertia tensor, is defined as:

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} \quad (3.11)$$

If the UAV is axis-symmetric, as usual for quadcopters, the inertia matrix can be simplified as shown in Equation 3.12:

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix}. \quad (3.12)$$

Therefore, it is important to understand whether adding loads to the UAV will make the mass distribution excessively asymmetric, such that it is ideal to use the inertia tensor presented by Equation 3.11.

One can then define the dynamics of a rigid body, subject to external forces applied to its center of mass fixed in the body frame, using the Newton-Euler formulation with the following relation (BOUABDALLAH, 2007):

$$\begin{bmatrix} \vec{F}^B \\ \vec{\tau}^B \end{bmatrix} = \begin{bmatrix} mI_{3 \times 3} & 0_{3 \times 3} \\ 0_{3 \times 3} & J^B \end{bmatrix} \begin{bmatrix} \vec{V}^B \\ \vec{\omega}^B \end{bmatrix} + \begin{bmatrix} \vec{\omega}^B \times m\vec{V}^B \\ \vec{\omega}^B \times J^B \vec{\omega}^B \end{bmatrix} \quad (3.13)$$

The Newton-Euler dynamic model can be rewritten in a compact form. In this formulation B is the inertia matrix and C is the Coriolis-centripetal matrix, which contains nonlinear forces and moments due to centripetal and Coriolis forces induced by angular velocities. The vector ζ contains all the forces and torques contributing to the UAV motion, including gravity (MARTINI et al., 2022).

$$B\ddot{q} + C\dot{q} = \zeta \quad (3.14)$$

Let the general vector of forces $\vec{F}^B$ and moments $\vec{\tau}^B$ in the body frame be as follows:

$$\vec{\Lambda} = [\vec{F}^B \quad \vec{\tau}^B]^T. \quad (3.15)$$

The model presented in Equation 3.14 can also be expressed in the non-inertial, and defined in Equation 3.16, where $\vec{\Lambda}^B$ is a vector containing the forces and moments contributing to the vehicle's motion in the body frame.

$$B^B \vec{v} + C^B \vec{v} = \vec{\Lambda} \quad (3.16)$$

In cases where the rigid body is axis-symmetric, a simplified version of the matrix C^B is derived from applying the vector products of the Equation 3.13 as follows:

$$C^B = \begin{bmatrix} 0 & 0 & 0 & 0 & mw & -mv \\ 0 & 0 & 0 & -mw & 0 & mu \\ 0 & 0 & 0 & mv & -mu & 0 \\ 0 & 0 & 0 & 0 & I_{zz}r & -I_{yy}q \\ 0 & 0 & 0 & -I_{zz}r & 0 & I_{xx}p \\ 0 & 0 & 0 & I_{yy}q & -I_{xx}r & 0 \end{bmatrix} \quad (3.17)$$

3.4 UAV Model Derivation

To reach a state-space model, the dynamic model of Equation 3.16 is rewritten by isolating the generalized acceleration vector $\vec{v}^B$.

$$\vec{v}^B = (B^B)^{-1}(-C^B \vec{v}^B + \vec{\Lambda}^B) \quad (3.18)$$

At the moment, the generalized vector of forces and moments $\vec{\Lambda}^B$ is a generic representation of all inputs that contribute to the vehicle's motion. This vector, nevertheless, may be decomposed into a series of physical phenomena that act on the UAV alongside the control inputs.

The contribution of the control inputs to the system motion consists of the thrust force and the roll, pitch, and yaw moments. It can be represented as a generalized vector of input forces and moments $\vec{\Lambda}_{in}$. The vector is formed from the control inputs U_1, U_2, U_3, U_4 .

$$\vec{\Lambda}_{in} = \begin{bmatrix} 0 & 0 & U_1 & U_2 & U_3 & U_4 \end{bmatrix}^T$$

The first fundamental phenomenon that affects the quadcopter displacement is gravity. The gravitational influence can be considered a force vector directed downwards along the z-axis of the Earth frame. Let $\vec{F}_g^E$ be the gravitational force expressed in the Earth frame, then,

$$\vec{F}_g^E = \begin{bmatrix} 0 \\ 0 \\ -m\vec{g} \end{bmatrix}. \quad (3.19)$$

In order to be included in the dynamic model, it must be referred to the body frame. This is done by applying the R_{xyz} transformation in the vector of Equation 3.19, which leads to

$$\vec{F}_g^B = (R_{zyx})^{-1} \vec{F}_g^E \Rightarrow \vec{F}_g^B = \begin{bmatrix} mg \cdot S\theta \\ -mg \cdot C\theta S\phi \\ -mg \cdot C\theta C\phi \end{bmatrix}.$$

Finally, the force vector can be included in a generalized forces and moments vector. The decomposed gravitational force will occupy the first three lines of a vector $\vec{\Lambda}_g$. This vector is represented in Equation 3.20:

$$\vec{\Lambda}_g = \begin{bmatrix} \vec{F}_g^B \\ 0_{3 \times 1} \end{bmatrix}. \quad (3.20)$$

Another fundamental physical contribution that affects motion is the gyroscopic effect. The gyroscopic effect refers to the tendency of a rotating body to maintain its orientation and resist changes to its axis of rotation. In this sense, the gyroscopic effect is always present in multirotor drones whenever the input torques change the direction of the propeller rotation axes.

The net angular momentum rate $\vec{H}_{net}$, or the torque caused by the gyroscopic effect, is assumed to be the cross product of the drone's angular velocities in the body frame and the net angular momentum of the propellers:

$$\vec{H}_{net} = \vec{\omega}^B \times \vec{H}_{net} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ \vec{H}_{net} \end{bmatrix}.$$

Which makes the moments caused by the gyroscopic effect those in the following vector:

$$\vec{H}_{net} = \begin{bmatrix} qJ_r\Omega \\ -pJ_r\Omega \\ 0 \end{bmatrix}, \quad (3.21)$$

where J_r is the propeller's moment of inertia and Ω is the propeller's net rotational speed. From Equation 3.21, it's observed that, for a roll movement (input U_2), the gyroscopic effect creates a rotation around y_b . Similarly, for a pitch movement (input U_3), the gyroscopic effect creates a rotation around x_b .

These moments can also be represented as a generalized vector of forces and moments Λ_{gr} , as in equation 3.22. In the equation, the variable Ω is expanded into the rotational speeds of the individual propellers. The signals assigned to the rotations depend on the direction of rotation of each motor, and the one adopted for this equation follows the standard Ardupilot's numbering.

$$\Lambda_{gr} = \begin{bmatrix} 0_{3 \times 1} \\ qJ_r(\Omega_1 + \Omega_2 - \Omega_3 - \Omega_4) \\ -pJ_r(\Omega_1 + \Omega_2 - \Omega_3 - \Omega_4) \\ 0 \end{bmatrix} \quad (3.22)$$

Up to this point, we have discussed the main physical phenomena addressed in quadcopter models that satisfy the design of simpler controllers. However, considering that the vehicle's task is trajectory following, that the drone travels at higher linear speeds and in open space, and that the UAV is eventually large in volume, additional aerodynamic effects must be considered. Even using a simplified approach, the addition of aerodynamic drag can provide considerable improvements to the controller design.

As shown in Svacha, Mohta e Kumar (2017), as the quadcopter's speed increases, aerodynamic effects such as blade flapping, induced drag, and thrust variations caused by induced speed become increasingly significant, to the point of degrading the robot's control performance. The study shows that compensating for these effects with a simplified drag model improves forward flight by up to 75%.

The study considers the induced drag linear in the horizontal velocities of the body frame. It also considered that other drag-like aerodynamic effects can be approximated as linear in the horizontal motion of the body frame and that the parasitic drag is negligible for normal flight regimes. In this sense, the drag force F_d is modeled as show in Equation 3.23:

$$F_d = -k_d \Omega_t V^{B*} \quad (3.23)$$

where k_d is the drag constant, Ω_t is the sum of all propellers rotation speeds and V^{B*} is the speed of the quadrotor expressed in the body frame (including only xy components). Similar to other physical phenomena, drag can be represented as a generalized vector of forces and torques. This is done by decomposing F_d into components in the direction of the velocities u and v :

$$\Lambda_d = \begin{bmatrix} -k_d \Omega_t u \\ -k_d \Omega_t v \\ 0_{4 \times 1} \end{bmatrix} \quad (3.24)$$

The forces and torques that contribute to the quadcopter's motion are compiled in the sum of Equation 3.25.

$$\vec{\Lambda} = \vec{\Lambda}_{in} + \vec{\Lambda}_g + \vec{\Lambda}_{gr} + \vec{\Lambda}_d \quad (3.25)$$

Thus, the generalized acceleration vector can be rewritten as shown in Equation 3.26:

$$\vec{v}^B = (B^B)^{-1} (C\vec{v}^B + \vec{\Lambda}_{in} + \vec{\Lambda}_g + \vec{\Lambda}_{gr} + \vec{\Lambda}_d) \quad (3.26)$$

Considering that the quadcopter's mass distribution is axis-symmetrical and, therefore, the simplified version of the Coriolis matrix is used, the nonlinear model that defines the dynamics of the UAV in the body frame will be:

$$\begin{cases} \dot{u} = vr - wq + g \sin \theta - k_d \Omega_t u \\ \dot{v} = wp - ur - g \cos \theta \sin \phi - k_d \Omega_t v \\ \dot{w} = uq - vp - g \cos \theta \cos \phi + \frac{U_1}{m} \\ \dot{p} = \frac{(I_{yy} - I_{zz})}{I_{xx}} \cdot q \cdot r - \frac{J_r}{I_{xx}} \cdot q \cdot \Omega + \frac{U_2}{I_{xx}} \\ \dot{q} = \frac{(I_{zz} - I_{xx})}{I_{yy}} \cdot p \cdot r - \frac{J_r}{I_{yy}} \cdot p \cdot \Omega + \frac{U_3}{I_{yy}} \\ \dot{r} = \frac{(I_{xx} - I_{yy})}{I_{zz}} \cdot p \cdot q + \frac{U_4}{I_{zz}} \end{cases} \quad (3.27)$$

Further, the model achieved in Equation 3.27 might change as new terms are added in $\vec{\Lambda}^B$ variable of equation 3.18. Other physical phenomena can be added to the model in order to improve its accuracy, such as rolling moment, ground effect (BOUABDALLAH, 2007), or aerodynamic effects caused by parasitic drag forces (MARTINI et al., 2022).

Since the control is done considering position and attitude in an inertial frame, the variables obtained in Equation 3.27 must be transformed to the Earth frame:

$$\begin{bmatrix} \ddot{x} & \ddot{y} & \ddot{z} \end{bmatrix}^T = R_{xyz} \begin{bmatrix} \dot{u} & \dot{v} & \dot{w} \end{bmatrix}^T, \\ \begin{bmatrix} \ddot{\phi} & \ddot{\theta} & \ddot{\psi} \end{bmatrix}^T = T \begin{bmatrix} \dot{p} & \dot{q} & \dot{r} \end{bmatrix}^T.$$

Or, in a compact formulation it is defined by Equation 3.28:

$$\vec{\ddot{q}} = \begin{bmatrix} R_{xyz} & 0_{3 \times 3} \\ 0_{3 \times 3} & T \end{bmatrix} \vec{\dot{v}} \quad (3.28)$$

Note that this is the same relationship as Equation 3.9; however, here it is applied to accelerations.

This final dynamic model, reached at equations 3.27 and 3.28, is very suitable for the simulation environment. This model, however, is too complex for most analyses or controller designs. With that in mind, a simplified model that transforms equation 3.27 to the Earth frame of reference is proposed.

To transform the rotational dynamics, the small angle approach is used. It states that the roll and pitch angles applied to move any multicopter horizontally are very small. Considering the small angles, T becomes an identity matrix, which implies that body and Earth frame equations will be the same. To transform the translational dynamics Equation 3.4 is used. The algebraic development to refer to the drag forces for the Earth frame is deduced in the Appendix B.2.

$$\left\{ \begin{array}{l} \ddot{x} = -k_d \Omega (\cos \psi - \sin \psi) \dot{x} + \frac{1}{m} (\cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi) U_1 \\ \ddot{y} = -k_d \Omega (\cos \psi + \sin \psi) \dot{y} + \frac{1}{m} (\sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi) U_1 \\ \ddot{z} = -g + \frac{1}{m} (\cos \phi \cos \theta) U_1 \\ \ddot{\phi} = \frac{(I_{yy} - I_{zz})}{I_{xx}} \cdot \dot{\theta} \cdot \dot{\psi} - \frac{J_r}{I_{xx}} \cdot \dot{\theta} \cdot \Omega + \frac{U_2}{I_{xx}} \\ \ddot{\theta} = \frac{(I_{zz} - I_{xx})}{I_{yy}} \cdot \dot{\phi} \cdot \dot{\psi} - \frac{J_r}{I_{yy}} \cdot \dot{\phi} \cdot \Omega + \frac{U_3}{I_{yy}} \\ \ddot{\psi} = \frac{(I_{xx} - I_{yy})}{I_{zz}} \cdot \dot{\phi} \cdot \dot{\theta} + \frac{U_4}{I_{zz}} \end{array} \right. \quad (3.29)$$

The next step in the mathematical model is to get the propellers' rotation speeds. It can be done by identifying the BLDC motor dynamics (BOUABDALLAH, 2007), applying motor mixing algorithms (Ardupilot Development Team, 2024a) or simply relating the rotation speed and the thrust generated by the propellers at steady-state (LIMA, 2015).

The last approach, adopted in this work, is justified since the BLDC motor dynamics are much faster than the rigid-body dynamics of the UAV. To get the equivalent propeller angular speed, it is made the assumption that the thrust force generated by each motor F_m is related to its rotation by a quadratic function (PINTO et al., 2019) presented in Equation 3.30

$$F_m(\Omega_m) = b\Omega_m^2, \quad (3.30)$$

where b is a coefficient proportional to the propeller's thrust coefficient C_T .

Likewise, the drag moment caused by each propeller τ_d is also related to the rotation speed by the quadratic expression of Equation 3.31

$$\tau_d(\Omega_m) = d\Omega_m^2, \quad (3.31)$$

where d is a coefficient proportional to the torque coefficient caused by drag C_Q .

From this point on, the equations are specific to each topology of multicopter drone. Both the number of propellers and the way they are distributed will result in different equations. For this work, the equations of quadcopters in $+$ (*plus*) and X topologies will be analyzed. This work follows the standard motor enumeration given by the Ardupilot project.

The first input U_1 (thrust force) is defined as the sum of the individual propellers' thrust forces. Hence, using Equation 3.30, the thrust is:

$$\begin{aligned} U_1 &= F_{m_1} + F_{m_2} + F_{m_3} + F_{m_4} \\ U_1 &= b\Omega_1^2 + b\Omega_2^2 + b\Omega_3^2 + b\Omega_4^2 \\ U_1 &= b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{aligned} \quad (3.32)$$

The second input U_2 (roll moment), in the *plus* topology, is the difference between the moments generated by motors 2 and 1:

$$\begin{aligned} U_2 &= \tau_{m_2} - \tau_{m_1} \\ U_2 &= bl\Omega_2^2 - bl\Omega_1^2 \\ U_2 &= bl(\Omega_2^2 - \Omega_1^2) \end{aligned} \quad (3.33)$$

The second input U_3 (pitch moment), in the *plus* topology, is the difference between the moments generated by motors 4 and 3:

$$\begin{aligned} U_3 &= \tau_{m_4} - \tau_{m_3} \\ U_3 &= bl\Omega_4^2 - bl\Omega_3^2 \\ U_3 &= bl(\Omega_4^2 - \Omega_3^2) \end{aligned} \quad (3.34)$$

The last input U_4 (yaw moment) is the moment opposite to the drag torque of the propellers. Since propeller pair 1-2 has a higher rotational speed, using the Equation 3.30, the yaw moment is:

$$\begin{aligned} \tau_d &= \tau_{d_1} + \tau_{d_2} - \tau_{d_3} - \tau_{d_4} \\ \tau_d &= d(\Omega_1^2 + \Omega_2^2 - \Omega_3^2 - \Omega_4^2) \\ U_4 &= -\tau_d \\ U_4 &= d(-\Omega_1^2 - \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \end{aligned} \quad (3.35)$$

By algebraically manipulating Equations 3.32 to 3.35, we arrive at the following relations to individually determine each motor rotation speed for a quadcopter in *plus* topology:

$$\begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \\ \Omega_4 \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{1}{4} \cdot \frac{U_1}{b} - \frac{3}{4} \cdot \frac{U_4}{d} - \frac{U_2}{bl} - \frac{1}{2} \cdot \frac{U_3}{bl}} \\ \sqrt{\frac{1}{4} \left(\frac{U_1}{b} + \frac{U_4}{d} \right) + \frac{1}{2} \cdot \frac{U_3}{bl} + \frac{U_2}{bl}} \\ \sqrt{\frac{1}{4} \left(\frac{U_1}{b} + \frac{U_4}{d} \right) - \frac{1}{2} \cdot \frac{U_3}{bl}} \\ \sqrt{\frac{1}{4} \left(\frac{U_1}{b} + \frac{U_4}{d} \right) + \frac{1}{2} \cdot \frac{U_3}{bl}} \end{bmatrix} \quad (3.36)$$

For a quadcopter in *X* topology, the equations that relate U_1 and U_4 to the rotation speeds of the motors will be the same as for plus topology. In this case, however, the input U_2 will be defined by the difference of thrust forces between the propeller pairs 2-3 and 1-4:

$$\begin{aligned}
U_2 &= (F_{m_2} + F_{m_3}) \cdot \frac{l}{\sqrt{2}} - (F_{m_1} + F_{m_4}) \cdot \frac{l}{\sqrt{2}} \\
U_2 &= \frac{bl}{\sqrt{2}} \cdot (\Omega_2^2 + \Omega_3^2) - \frac{bl}{\sqrt{2}} \cdot (\Omega_1^2 + \Omega_4^2) \\
U_2 &= \frac{bl}{\sqrt{2}} (-\Omega_1^2 + \Omega_2^2 + \Omega_3^2 - \Omega_4^2)
\end{aligned} \tag{3.37}$$

At last, the input U_3 will be defined by the difference of thrust forces between the propeller pair 2-4 and 1-3:

$$\begin{aligned}
U_3 &= (F_{m_2} + F_{m_4}) \cdot \frac{l}{\sqrt{2}} - (F_{m_1} + F_{m_3}) \cdot \frac{l}{\sqrt{2}} \\
U_3 &= \frac{bl}{\sqrt{2}} \cdot (\Omega_2^2 + \Omega_4^2) - \frac{bl}{\sqrt{2}} \cdot (\Omega_1^2 + \Omega_3^2) \\
U_3 &= \frac{bl}{\sqrt{2}} (-\Omega_1^2 + \Omega_2^2 - \Omega_3^2 + \Omega_4^2)
\end{aligned} \tag{3.38}$$

By algebraically manipulating equations 3.32, 3.35, 3.37 and 3.38, we arrive at the following relations to individually determine each motor rotation speed for a quadcopter in X topology:

$$\begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \Omega_3 \\ \Omega_4 \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{1}{4} \left(\frac{U_1}{b} - \frac{\sqrt{2}}{bl} U_2 - \frac{\sqrt{2}}{bl} U_3 - \frac{U_4}{d} \right)} \\ \sqrt{\frac{1}{4} \left(\frac{U_1}{b} + \frac{\sqrt{2}}{bl} U_2 + \frac{\sqrt{2}}{bl} U_3 - \frac{U_4}{d} \right)} \\ \sqrt{\frac{1}{4} \left(\frac{U_1}{b} + \frac{\sqrt{2}}{bl} U_2 - \frac{\sqrt{2}}{bl} U_3 + \frac{U_4}{d} \right)} \\ \sqrt{\frac{1}{4} \left(\frac{U_1}{b} - \frac{\sqrt{2}}{bl} U_2 + \frac{\sqrt{2}}{bl} U_3 + \frac{U_4}{d} \right)} \end{bmatrix} \tag{3.39}$$

3.5 Determination of the System Parameters

Each UAV will have a series of unique parameters that depend on the vehicle's design. The drone's propeller layout, size, and geometry define a series of numerical coefficients that will dictate the system's dynamics.

The first parameters to be obtained are the moments of inertia that make up the inertia tensor of the equation 3.11. Since the geometry of a drone is complex and its mass distribution is not uniform, the best way to obtain the vehicle's inertia matrix is through computer-aided (CAD) design software.

Therefore, a 3D model of the drone, exhibited in Figure 15, was built in SolidWorks. The model was created from a series of previously designed parts available in public repositories.

Figure 15 – CAD model inspired by the experimental drone.



Source: The author (2026).

The mass of each of the drone's real components was weighed on a digital scale and assigned to the equivalent CAD part. In addition to the full-body inertia tensor, SolidWorks was also used to calculate the moment of inertia of the thrusters around the engines' axis of inertia.

3.5.1 Estimating the propeller coefficients with test stands

The next parameters to be estimated are the thrust coefficient b and the drag coefficient d . These values can be computed in different ways, namely:

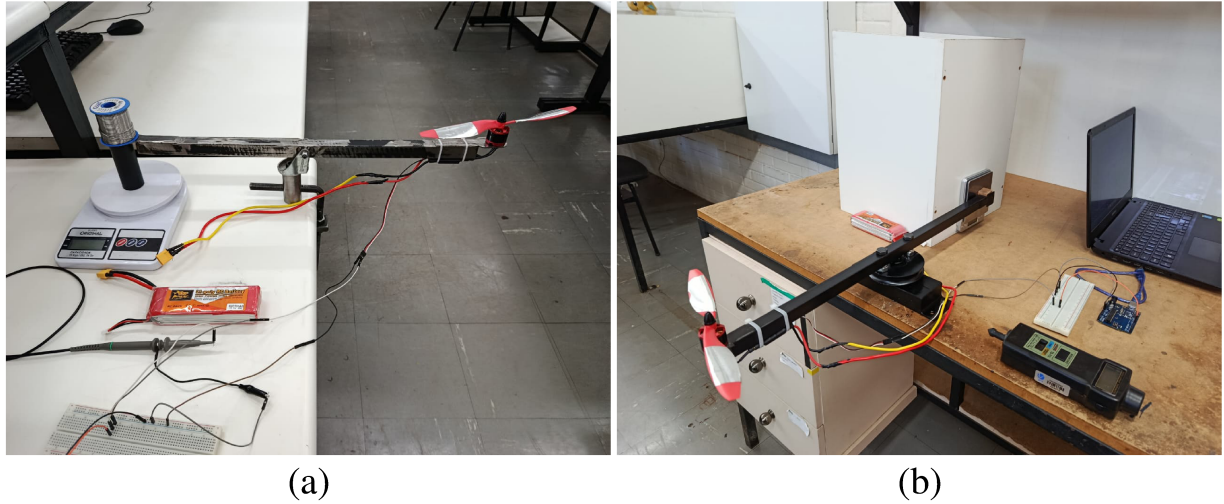
- Analytical calculation using blade element theory;
- Experimental estimation using a thrust test stand;
- Finite Element Analysis methods; and
- Experimental estimation using wind tunnels.

This work addresses only the estimation using thrust and drag test stands. This can be satisfactorily done by designing simple mechanical structures and measuring the forces and rotational speeds generated.

The first structure designed is the thrust test bench. It consists of a metal bar balanced on a rotating axis at its center. The axis of rotation of the structure is perpendicular to the plane of the surface on which it is installed. At one end, a propeller coupled to a BLDC motor is installed, and at the opposite end, there is a digital scale. The thrust stand is shown in Figure 16 (a).

The experiment consists of varying the pulse width of the control signal using a microcontroller connected to an ESC. As the signals vary, the thrust forces are measured by the scale. At the same time, an optical tachometer is used to measure the rotation of the propeller.

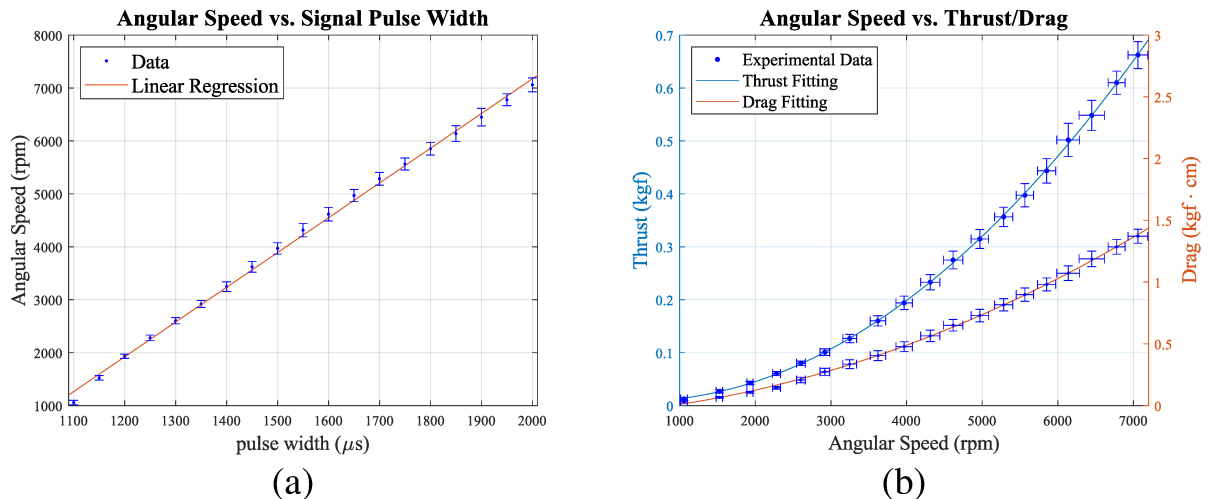
Figure 16 – Test benches for estimating propeller parameters, where: (a) is the thrust test stand and (b) is the drag test stand.



Source: The author (2026).

For the experimental procedure, some precautions had to be taken to avoid measurement errors. The propeller side must always be positioned outside the mounting surface, leaving enough free space below the propeller so that the ground effect does not interfere with the force measurements of the scale. In addition, a counterweight must be added to the end of the scale in order to prevent the weight of the propeller and the bar itself from causing the structure to rotate

Figure 17 – Curve fitting for the thrust test, where: (a) is a linear fit relating the signal pulse width and the rotation speed and (b) is a second-order polynomial fit relating the thrust/drag forces and the rotation speed.



Source: The author (2026).

at lower thrusts. Finally, the use of reflective tape is recommended to ensure accurate readings from the optical tachometer.

The second structure is the drag test stand, shown in Figure 16 (b). It consists of a metal bar supported on a bearing at its center. The axis of rotation is parallel to the plane of the table. At one end, a propeller coupled to a BLDC motor is installed, and at the opposite end, a precision digital scale is fixed to a vertical surface.

A MATLAB script is used to perform data analysis. It calculates the average values and the standard deviation at each reading point for both thrust and drag tests.

3.5.2 Aerodynamic drag estimation

The drag coefficient k_d , as presented in equation 4, is obtained experimentally from logs of a real flight. In the autonomous mission prepared for the experiment, the drone navigates north and returns south repeatedly. As it navigates between one waypoint and another, the speed V^{B*} changes from $1m/s$ to $10m/s$ in regular steps of $1m/s$. The drone always performs a yaw rotation before heading to the indicated waypoint, so that the speed on the x-axis of the body frame u is always positive, and thus $V^{B*} = u$.

Upon reaching each waypoint, the mission includes a delay command to allow the drone time to fully decelerate and hover for a few seconds before yawing. These hover periods collect pitch offsets that are removed from the data as a bias caused by the wind.

From equation 3.23, we derive the relationship between quantities obtained in flight that are related to the aerodynamic drag coefficient:

$$F_d = K_d \Omega_t u$$

$$(m \cdot a_{IMU_x}) \cdot \cos \phi = K_d \Omega_t u.$$

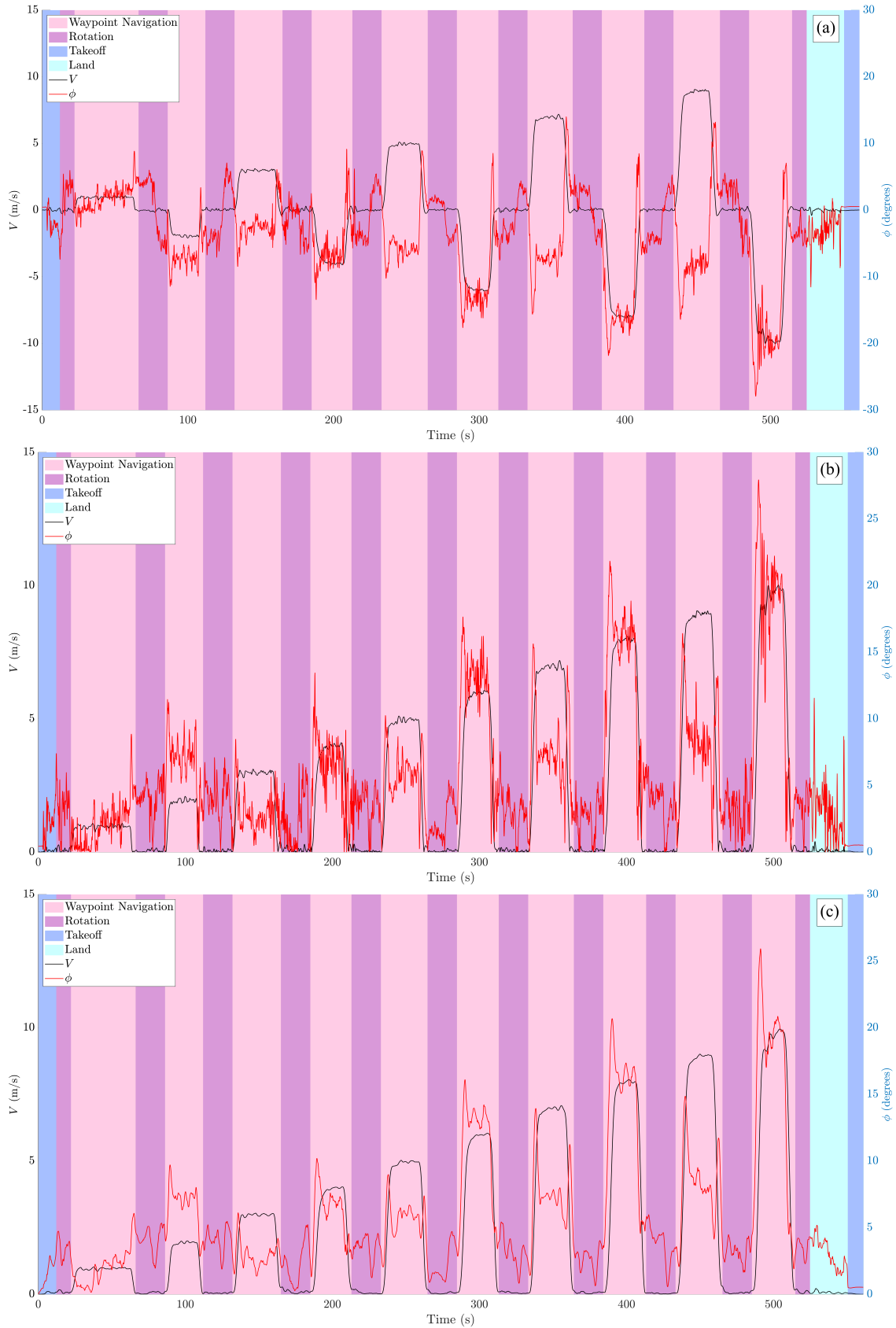
Isolating the measured acceleration a_{IMU} ,

$$a_{IMU} = \frac{k_d}{m} \cdot \Omega_t u \cos \phi \quad (3.40)$$

which can be approximated by a linear function (SVACHA; MOHTA; KUMAR, 2017), where the dependent variable is acceleration and the independent variable is the product $\Omega_t u \cos \phi$.

The data are obtained from onboard log messages, which comprise groups of variables read during a flight. The data groups analyzed are Position Control North (PSCN), Canonical Vehicle Attitude (ATT), Executed Mission Command Information (MISE), the PWM channels output (RCOU), and the inertial measurement unit (IMU) readings. The PSCN logs contain the speed in the North-South direction, from which the values of u are obtained. Similarly, the values of ϕ are obtained from ATT, Ω_t from RCOU, and a_{IMU} from IMU. The MISE readings serve to separate the flight stages (takeoff, navigation, delay, yaw turn, and landing).

Figure 18 – Steps in data processing where: (a) is the raw data, (b) the modulus of the measured quantities, and (c) the data with filtered pitch measurements.



Source: The author (2026).

First, the pitch data, which is the noisiest, was processed. Figure 18.(a) shows the raw pitch and velocity data in the different stages of flight. The next step was to obtain the magnitudes of the attitude and velocity signals, exhibited in Figure 18.(b). After this step, it is observed that in the delay and rotation stages there is an almost continuous offset in the pitch angle. To facilitate the capture of angle and velocity signals only during steady-state flight, the signal was filtered using a moving average with a 25-sample window.

The main cause of this continuous offset in pitch angle is the wind acting as a constant force from the south. Since the position controller attempts to keep the drone stationary, the drone must maintain a sufficient tilt so that the thrust decomposed in the north direction is enough to overcome the disturbance caused by the wind. For data regression, it is necessary to filter out these disturbances caused by the environment. The bias caused by the wind was calculated from the arithmetic mean of the pitch angle during the hover periods (delay + yaw rotation) of the test flight. The angles considered were those of the filtered data presented in Figure 18 (c).

The speed has no bias, since the controller works with a trajectory profile at constant speed and a control law with integral action that cancels out position and speed errors caused by the wind.

The bias caused by the wind was subtracted when the drone was navigating south and added when navigating north. Figure 19 (a) shows the horizontal speed and pitch when the drone is navigating against the wind and with the wind, respectively, without correction. Figure 19) shows the same data with the bias corrected.

From the unbiased pitch graphs, steady-state operating ranges were selected by visual inspection. The arithmetic mean and standard deviation of the values in each of these ranges were calculated for the pitch and speed variables. This entire process was repeated to calculate the sum of the motor speeds and the IMU acceleration at their average steady-state values.

The errors from the sum of the pitch values with the bias caused by the wind were propagated using the Equation 3.41, and the errors from the Ω product were propagated using the Equation 3.42

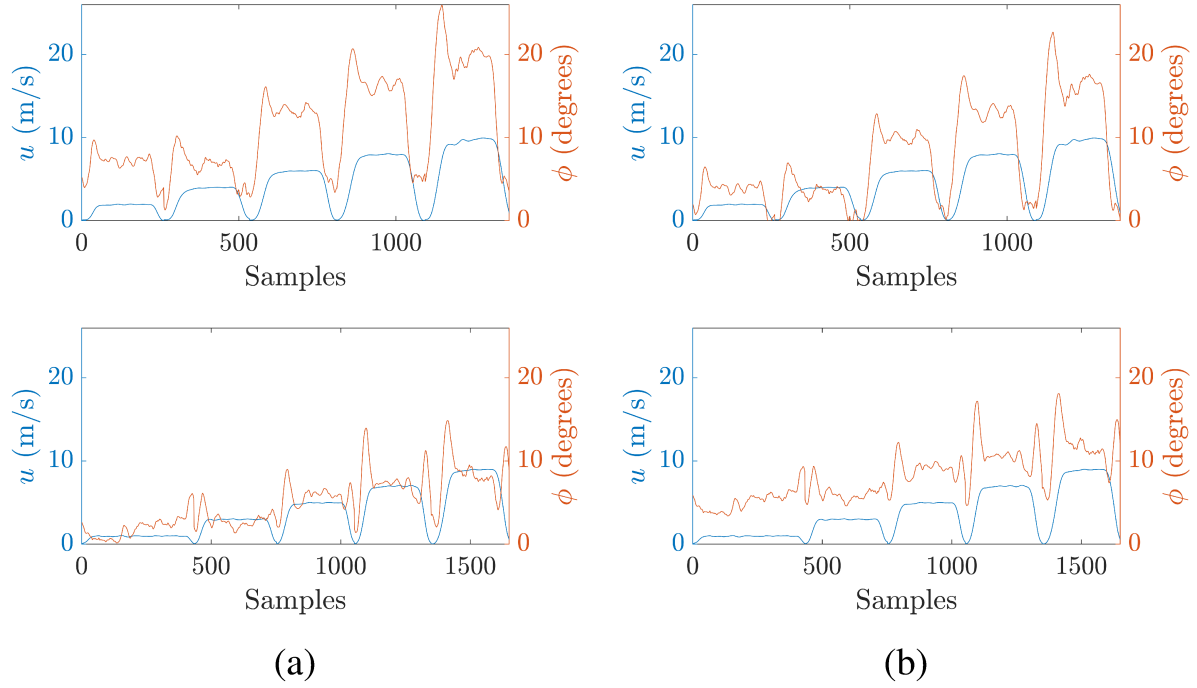
$$(\sigma_{\bar{R}})^2 = (\sigma_{\bar{a}})^2 + (\sigma_{\bar{b}})^2 \quad (3.41)$$

$$\left(\frac{\sigma_{\bar{R}}}{\bar{R}}\right)^2 = \left(\frac{\sigma_{\bar{a}}}{\bar{a}}\right)^2 + \left(\frac{\sigma_{\bar{b}}}{\bar{b}}\right)^2 \quad (3.42)$$

where $\sigma_{\bar{R}}$ is the propagated standard deviation of the result, and $\sigma_{\bar{a}}$ and $\sigma_{\bar{b}}$ are the standard deviations of the mean input variables $\bar{a}$ and $\bar{b}$.

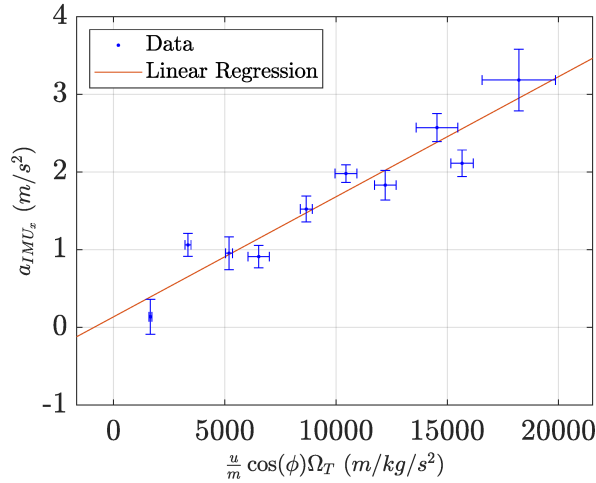
Based on the average value of the variable for each segment of the flight, a linear regression was performed using the equation 3.40. The resulting linear fitting is shown in Figure 20, and has a slope corresponding to $k_d = 1.545 \cdot 10^{-4} \frac{N \cdot s^2}{m \cdot rad}$.

Figure 19 – Flight periods corresponding to navigation between waypoints, against and with the wind respectively, where: the graphs on the left (a) contain the pitch curves before correction for the deviation caused by the wind, the graphs on the right (b) contain the pitch curves after correction.



Source: The author (2026).

Figure 20 – Linear regression used to compute the induced aerodynamic drag coefficient k_d .



Source: The author (2026).

3.6 Linear State-Space Representations

Even if a simulation approach using body-frame Equations 3.27 and transformation matrices of Equation 3.5 and Equation 3.6 is employed, linear representations around specific

operating points are necessary to design linear-quadratic control systems.

Considering the general nonlinear time-varying state equations $\dot{x} = f(x(t), u(t), t)$ and $y(t) = h(x(t), u(t), t)$ the linear state-space model around a specified trajectory is defined as:

$$\begin{aligned}\dot{x}_\delta(t) &= Ax_\delta(t) + Bu_\delta(t) \\ y_\delta(t) &= Cx_\delta(t) + Du_\delta(t).\end{aligned}\tag{3.43}$$

where the subscript δ denotes small deviations of the state, inputs and outputs of the system. As shown in Williams e Lawrence (2007), the model is obtained analytically using the Taylor series, and since the deviations are small the high-order terms can be neglected to obtain the linear state-space representation of Equation 3.43.

If the nonlinear system is not explicitly time-dependent and the nominal trajectory is an equilibrium condition, then the coefficient matrices in the linear state equation are constant. In other words the linearization yields a time-invariant linear system in this case.

Some works, like Lima (2015), linearize analytically the simplified system in Earth frame, like the shown in Equation 3.29. This approach, while accurate, becomes excessively complex as the modeler adds more physical phenomena or do not simplify actuator dynamics.

Since this work aims to develop a Model-in-the-Loop that increasingly incorporates more model features, the numerical calculation of linearization was chosen. This is done by using the `linmod` command of MATLAB (The MathWorks, Inc., 2026). It uses the dynamic model developed in Simulink rearranged to get the trim points as arguments. The function applies numerical perturbation linearization, a technique used to approximate complex, nonlinear systems by introducing small changes (perturbations) to inputs and states. It empirically computes the system's Jacobian matrices by measuring how the system's outputs change in response to these targeted variations, bypassing the need for explicit analytical derivatives. Therefore, the Simulink model requires the user to indicate in the diagram which signals will receive disturbances.

Since the system exhibits many nonlinearities, a single linearization would not be sufficient for a regulator-type controller, such as the LQR, to control the system for trajectory tracking. Thus, the system was linearized for 15 operating points, from which the total horizontal velocity in the body frame $V^{B*} = \sqrt{u^2 + v^2}$ and the yaw rate r were chosen to be varied. The forward velocities were 0m/s , 1.5m/s , 3m/s , 4.5m/s , and 6m/s , and the yaw rates were -2rad/s , 0rad/s , and 2rad/s .

Materials and Methods

4.1 Ardupilot Software Architecture

Ardupilot is a collection of open source autopilot software made to run on a wide range of hardware. It enables the creation of trusted, autonomous, and unmanned vehicle systems. The project contains not only the controller firmware, but also a suite of tools for the vehicle development process and operation ([Ardupilot Development Team, 2024b](#)).

The project includes different kinds of firmware for each type of vehicle and mission, namely: *Copter*, *Plane*, *Rover*, *Sub* and *Antenna Tracker*. This work will only address the firmware called *Copter* and its particularities.

Copter, in turn, is an open-source autopilot system for multicopters, traditional helicopters (single or dual rotor), or alternative rotor flying vehicles. It provides a wide variety of flight modes. The user is capable of choosing from fully manual modes to fully autonomous ones. The key features of *Copter* include:

- easy to operate manual modes, like the *Altitude Hold* mode;
- loiter and *PosHold* modes, which holds the vehicle's position using its GPS, accelerometers and barometer;
- a *return-to-launch* mode, that guides the vehicle back to the takeoff position automatically;
- autonomous missions made through Ground Control Station (GCS) softwares or mission scripts in *Auto* mode; and
- failsafes, which issue alarms or trigger actions in unfavorable flight conditions.

A final remark is that Ardupilot firmwares are quite configurable, ensuring that they can be applied to as many use cases as possible. There is a wide list of parameters that can be set

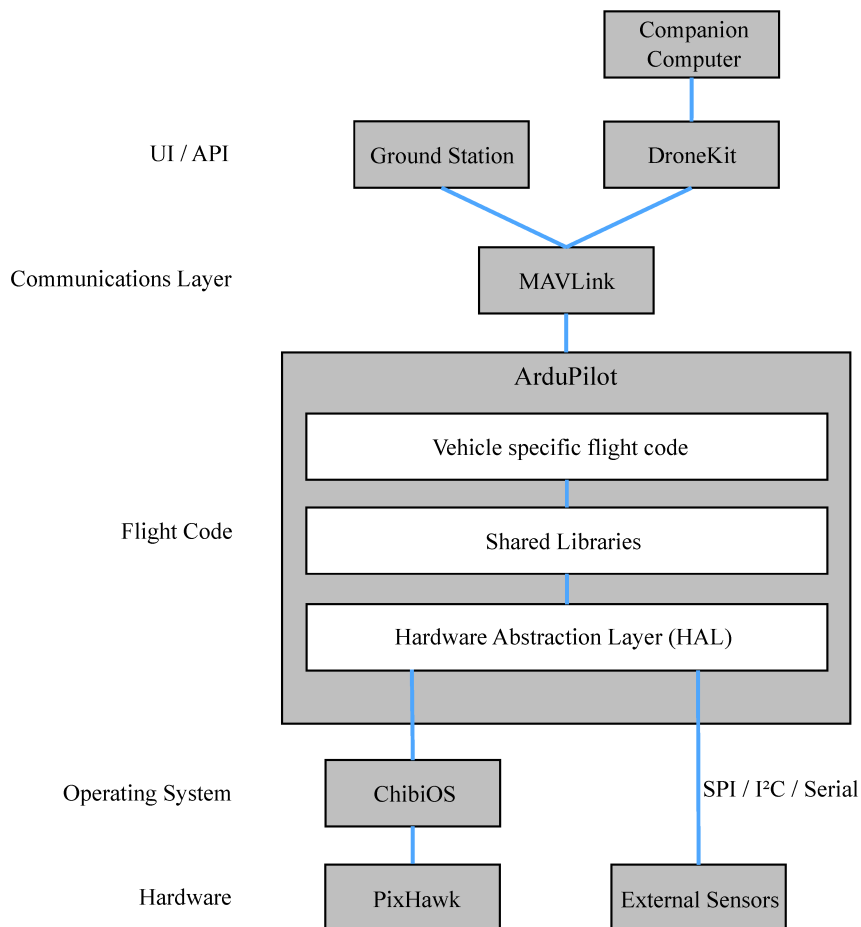
to control the vehicle behaviour. These configurations are stored in the persistent storage of the vehicle. The parametrization can be done using a GCS, MavProxy, or Python scripting, both through the MAVLink protocol, a topic that will be later discussed.

4.1.1 Firmware Organization

Since the ArduPilot code base is very large, including about 700,000 lines in the core git tree, not all aspects of the software will be discussed. The ArduPilot code architecture can be broadly divided into five main parts: vehicle code, shared libraries, hardware abstraction layer (HAL), tools, and external support code ([Ardupilot Development Team, 2024a](#)).

The vehicle codes define the firmware for each vehicle type. The shared libraries are code resources shared among all vehicles, such as sensor drivers, attitude and position estimators (Kalman Filters), and control codes (PID controllers). The HAL is the software layer that defines the interface between the code and hardware-specific features. Finally, there is support for external resources such as those used in PixHawk boards and the MAVLink interfaces.

Figure 21 – Top view diagram of a generic ArduPilot use case.



Source: Adapted from [Ardupilot Development Team \(2024a\)](#)

The tools part, otherwise mentioned, is a set of miscellaneous software provided for

testing, autotesting, controller tuning, and more. They are not part of the autopilot main system, and so, these tools are not going to be addressed.

The diagram in Figure 21 shows the top view of the ArduPilot software architecture. The firmware consists of the ArduPilot flight code, the Operating System, and hardware-specific support. External to the flight controller, an autonomous system might have a companion computer or ground station, in which the autonomous missions are programmed.

4.1.2 Autonomous Mission Development

There are two main ways to program autonomous missions using the tools provided by the ArduPilot project. The first method consists of passing flight instructions from a GCS or a companion computer through the MAVLink protocol. The second approach is made by embedding a flight script directly on the flight controller.

The main approach that autopilot systems use for monitoring and controlling unmanned missions is using the MAVLink protocol. The Micro Air Vehicle Link (MAVLink) is a communication protocol for unmanned systems that specifies a comprehensive set of messages, exchanged between the autonomous vehicles and ground control stations (KOUBAA et al., 2019). The protocol defines a mechanism on the structure of messages and how to serialize them on the application layer. The MAVLink messages, which are binary-serialized messages, are reduced in size when compared with other types of serialization (such as XML or JSON). The protocol supports different types of transport layers and mediums thanks to its lightweight structure. It can be transmitted both by TCP/IP networks or serial telemetry low bandwidth channels.

The easiest way to develop an autonomous vehicle with MAVLink and ArduPilot is using a companion computer. In this approach, an external hardware would pass MAVLink commands to the flight controller using the MAVLink protocol. This hardware, usually called a companion or flight computer, is a board running embedded Linux. The autonomous mission can be developed using Python scripts and the *Pymavlink* package. This approach is widely adopted since using a Linux system with Python programming allows the autonomous mission to be integrated with computer vision, path planning, and geoprocessing algorithms. This way, it is also possible to control actuators integrated into the vehicle that could not be integrated using just a regular flight controller.

On the other hand, the onboard autonomous mission development can be implemented using ArduPilot's Lua scripting support. Lua scripts can be used to monitor or manipulate the vehicle state and execute MAVLink commands. Multiple scripts can be run at once, in parallel with the flight code, and they are stored on the controller's SD card. Scripts are run at low priority on the system, ensuring the flight code continues its execution even if a script is taking a long time to process.

4.1.3 Simulation and Testing

As an auxiliary tool in the experimental procedure and for simulation, MAVProxy was adopted as the main GCS. The MAVProxy is a minimalist UAV ground control station for autonomous systems using the MAVLink protocol.

It is a command-line, console-based app, but it also includes plugins to provide a basic graphical user interface (GUI). The tool has the ability to forward messages from a UAV over the network to multiple ground station software on other devices ([ArduPilot Development Team, 2024c](#)). Using commands, it is possible to arm and disarm the vehicle, set mission waypoints, change between flight modes, configure geofences, and set any vehicle parameters.

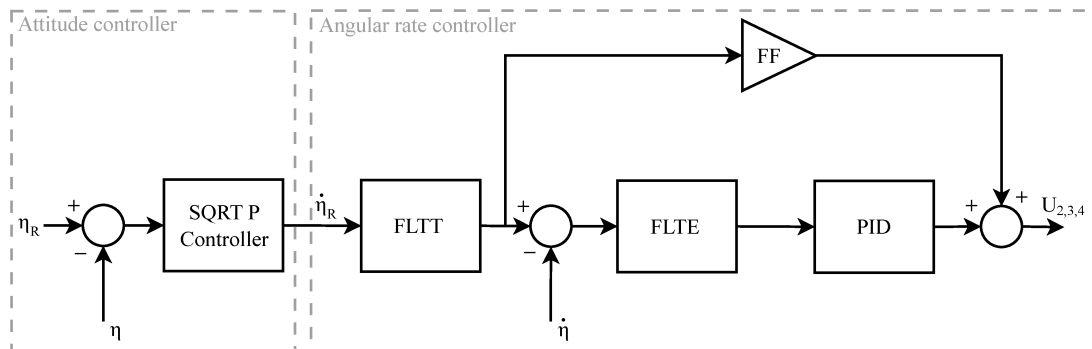
However, the biggest justification for using MAVProxy is the integration with simulation interfaces. The Software-in-the-loop (SITL) simulator, for example, can be used in conjunction with MAVProxy for validations during the development of new firmware.

The SITL allows running ArduPilot directly on a computer, without the need of a flight controller. When running in SITL, sensor data comes from a flight dynamics model in a flight simulator. The simulator supports a wide range of vehicles, for example: multi-rotor aircraft, fixed-wing aircraft, ground vehicles, or underwater vehicles. The SITL is a build of the autopilot source code in C++, giving developers a way to test the behavior of the code without any hardware.

4.2 ArduPilot's Control Architecture Modeling

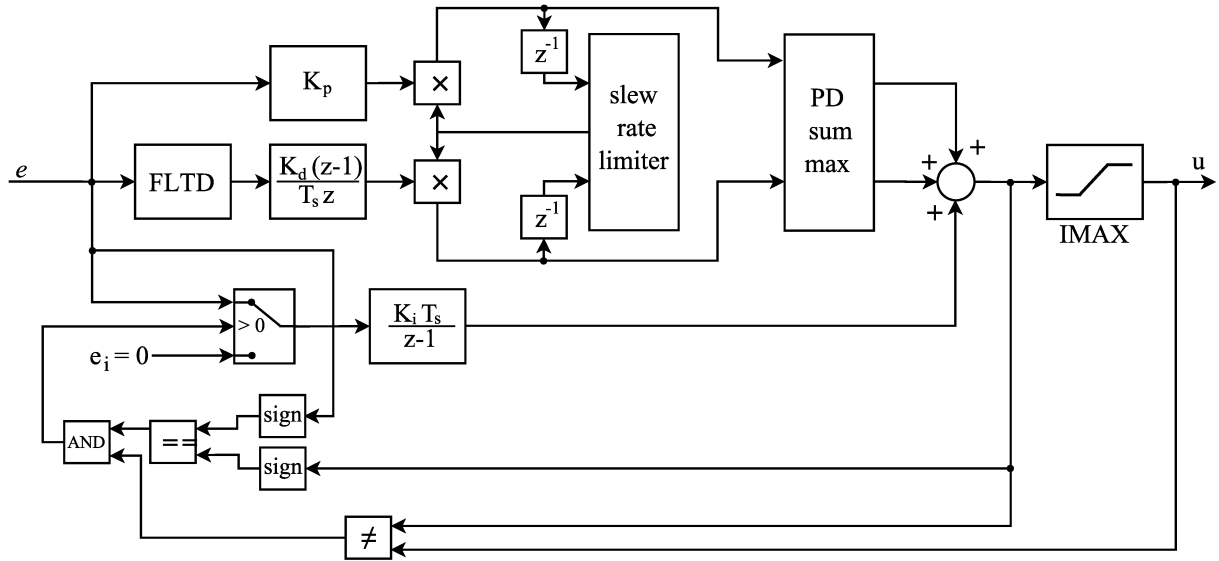
ArduPilot's control strategy is based on cascade control, as illustrated in Fig. 12. Rotation rates and Euler angles are also controlled separately, in which angular rate control is cascaded with attitude control using a P–PID scheme, where attitude regulation relies solely on proportional gains. The xy linear motion control employs a similar P–PID cascade. The attitude control structure is shown in Figure 22. The PIDs are discretized using Euler's method and in parallel form.

Figure 22 – ArduPilot attitude controller block diagram. The structure is the same for the xy linear motion control, but with a traditional P controller for position.



Source: The author (2026).

Figure 24 – Ardupilot PID with all its available features.



Source: The author (2026).

The controller also has the *slew rate limiter*, that acts as a damper over the control signal, smoothing the input change rates. Finally, the Ardupilot's PID also includes a mechanism to clamp the sum of the P and D control signals, limiting the control action generated by them. This last feature limits the generation of excessive control signals in large vehicles or those with slow actuators.

Both the dynamic model and the control loops were implemented using MATLAB and Simulink. The main Simulink file contains a structure similar to the cascade architecture shown in Figure 12.

The trajectory planning block was implemented using Ardupilot Software-in-the-Loop flight logs. Python scripts command a simulated UAV in AUTO mode, where waypoint navigation (WPNav) interpolates target points in the north-east-down frame. The resulting simulation logs are processed to extract trajectory setpoints into a CSV file, which is then used in the Simulink model.

An attitude planner block was also implemented. It transforms the control signals of linear speeds in the xy plane into roll and pitch setpoints ϕ_R and θ_R . The transformation uses . The angular references are calculated using the u_x and u_y virtual.

The controller, the rigid body dynamics, and the rotor dynamics blocks were implemented following the concepts developed in Chapters 2 and 3, using custom Simulink blocks and compiled MATLAB functions.

4.3 Optimal Control Architectures and Implementation

The best way to implement optimal control in multirotor drones is by maintaining a cascaded control strategy. This allows one to begin, initially, by developing attitude control. This strategy retains the structure developed by Ardupilot for position control. In this project, zero steady-state error is considered a requirement for angular variables but not for angular rates.

To develop the optimal controller, a linear state-space representation of the rotational subsystem is necessary. This system can be created by partitioning the matrices A, B, C, and D obtained through linearization to obtain only the states ϕ , θ , ψ , p , q , and r . Furthermore, since the goal is to integrate the angles phi, theta, and psi, they will use output feedback and must therefore be considered in matrix C. Thus, the rotational system can be described as follows:

$$\begin{aligned} A_{rot_{6 \times 6}} &= A(7 : 12, 7 : 12) \\ B_{rot_{6 \times 3}} &= B(7 : 12, 2 : 4) \\ C_{rot_{3 \times 6}} &= \begin{bmatrix} I_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} \\ D_{rot_{3 \times 3}} &= 0_{3 \times 3} \end{aligned} \tag{4.3}$$

The controller consists of an LQR/LQI with three augmented states which will be the integrals of the angles:

$$\varepsilon = \begin{bmatrix} \int (\phi_R - \phi) dt & \int (\theta_R - \theta) dt & \int (\psi_R - \psi) dt \end{bmatrix}^T.$$

If we consider ε_ϕ , ε_θ , and ε_ψ as the integrals of the roll, pitch, and yaw errors respectively, then the state vector will be $x = [\phi \ \theta \ \psi \ p \ q \ r \ \varepsilon_\phi \ \varepsilon_\theta \ \varepsilon_\psi]^T$ with inputs being the torques $u = [U_2 \ U_3 \ U_4]$ which implies a gain matrix $K_{3 \times 9}$.

Another detail to consider when establishing the control system architecture is how linearization delivers state variables and outputs. As seen in Equation 3.43, the quantities are represented as deviation variables (δx and δu). Therefore, to control the system displaced from the linearized operating point, this displacement must be considered in the control law. Trajectory-following systems, such as autonomous drones, fall into this type of control problem. Considering the control law of a LQR for deviation variables, we have:

$$\delta u = -K\delta x,$$

which means that the actual input for control will be

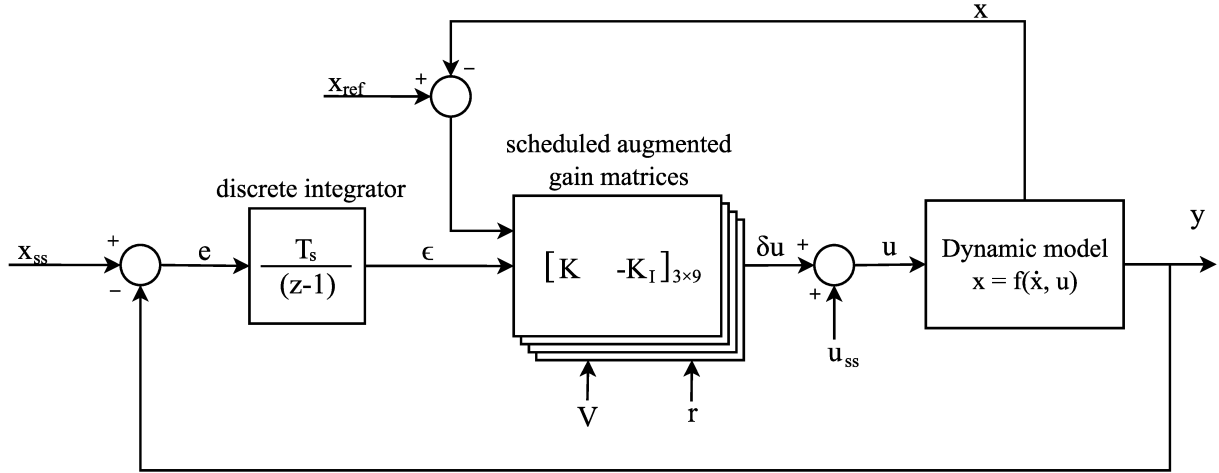
$$u = -K(x - x_{ss}) + u_{ss}$$

where x_{ss} and u_{ss} are the steady-state inputs for $f(\dot{x}, u)$ linearized at a given operating point. Considering the integral action taken by the output feedback, the control law will be:

$$u = -K(x - x_{ss}) - K_I(r - y) + u_{ss} \tag{4.4}$$

Figure 25 shows the rotational control of the nonlinear quadcopter driven by the control action of equation 4.4.

Figure 25 – Optimal control strategy developed for application in quadcopters.



Source: The author (2026).

The logic behind LQI for tracking is that even if the tracker adapts the optimal regulator to a trim point offset from that of the linearization, it does not guarantee the rejection of disturbances or parametric uncertainties. When unmodeled dynamics, disturbances such as wind or measurement/estimation noise occur, an integral action is necessary to eliminate the steady-state error regardless of the conditions.

This strategy alone would work satisfactorily if the drone's nonlinearity were not significant and a tracking task did not cause the aircraft to move between such diverse operating points. But that is not what happens in real-world flight scenarios.

A simple check reveals variations between the gain matrix calculated for hovering and for a flight with a cruising speed of 6 m/s. Both operating points are typically part of the same autonomous flight mission, but each one leads to clearly different K matrices:

$$K|_{r=0, V=0} = \begin{bmatrix} 6.9961 & 0 & 0 & 0.5076 & 0 & 0 & -24.3940 & 0 & 0 \\ 0 & 7.0483 & 0 & 0 & 0.5200 & 0 & 0 & -24.4353 & 0 \\ 0 & 0 & 6.8046 & 0 & 0 & 1.0368 & 0 & 0 & -8.3791 \end{bmatrix},$$

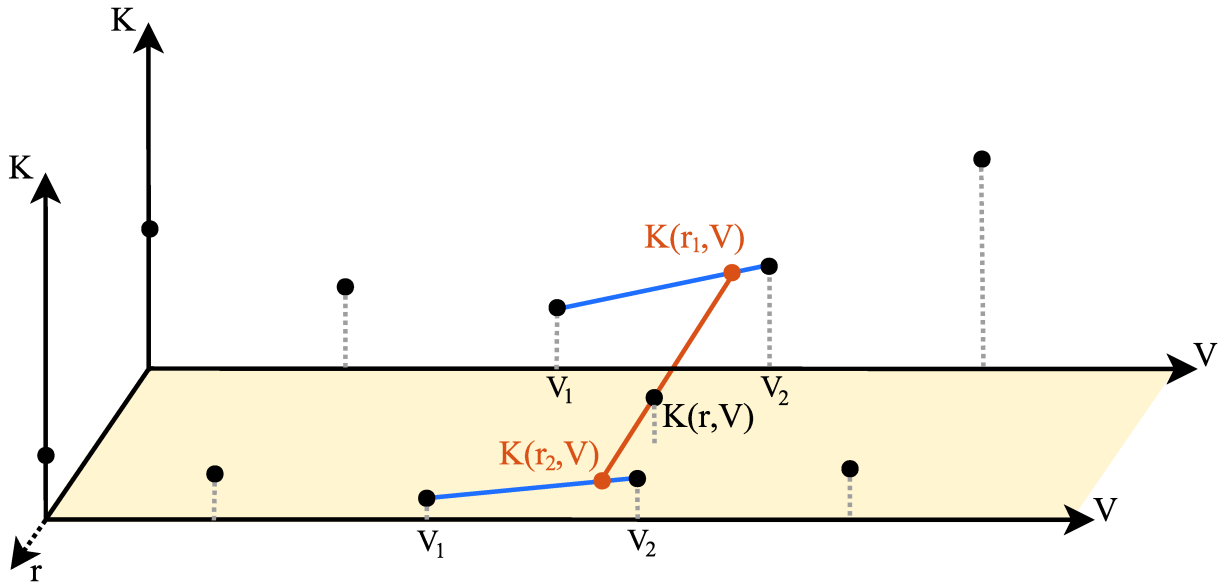
$$K|_{r=0, V=6} = \begin{bmatrix} 6.9424 & 0 & -0.4204 & 0.5058 & 0 & 0.0150 & -24.1922 & 0 & 0.4941 \\ 0 & 7.0483 & 0 & 0 & 0.5200 & 0 & 0 & -24.4353 & 0 \\ 1.6535 & 0 & 6.7348 & 0.0463 & 0 & 1.0428 & -6.7382 & 0 & -8.3096 \end{bmatrix}.$$

Therefore, in order to allow the drone the flexibility to operate in a flight envelope containing several operating points, a gain schedule was designed for the control system. This schedule includes the gain matrix K , the steady-state states x_{ss} , and the steady-state inputs u_{ss} . For the 15 operating points obtained, 15 gain matrices and their respective steady-state states and inputs were calculated using MATLAB.

During gain matrix switching, unwanted transients occur in the control signal. The greater the distance between the operating points, the greater these disturbances are during the application of the gain schedule. Therefore, a strategy is needed to smooth the change between distinct operating points. For this work, a bilinear interpolator was chosen that considers the total velocity V and the yaw rate R as dependent variables.

Bilinear interpolation is a mathematical technique used to calculate intermediate values in a two-dimensional grid, using the values of four known adjacent points. The procedure consists of applying linear interpolation along one of the grid directions and then performing a new linear interpolation in the orthogonal direction, thus obtaining a smooth estimate for the desired point. Figure 26 shows how this technique would be used to estimate a gain value from a grid formed by gain values K as a function of r and V .

Figure 26 – Interpolation algorithm used to estimate gain matrices, steady-state states, and steady-state inputs at operating points intermediate to those obtained with linearization.



Source: The author (2026).

This operation can be described by the sequence of equations 4.5:

$$\begin{aligned}
 K(r, V_1) &= \frac{r_2 - r}{r_2 - r_1} K(r_1, V_1) + \frac{r - r_1}{r_2 - r_1} K(r_2, V_1), \\
 K(r, V_2) &= \frac{r_2 - r}{r_2 - r_1} K(r_1, V_2) + \frac{r - r_1}{r_2 - r_1} K(r_2, V_2), \\
 K(r, V) &= \frac{V_2 - V}{V_2 - V_1} K(r, V_1) + \frac{V - V_1}{V_2 - V_1} K(r, V_2).
 \end{aligned} \tag{4.5}$$

The gain matrices were calculated at the 15 operating points using the MATLAB `lqi` function. For each case, the translational subsystem was extracted and discretized using Euler's method. Thus, the function output is a discrete LQI that considers the sampling period with

which the state-space system was discretized. The same matrices Q and R were used to calculate all LQIs.

The weight matrices were obtained using Bryson's Rule. This metric is a heuristic that considers that the weight assigned to the feedback of a state is inversely proportional to the maximum allowed value for that variable squared, and that the control cost weight is inversely proportional to the maximum amplitude of that signal squared. Thus, the values of the main diagonals of the weight matrices are defined by the Equations 4.6:

$$\begin{aligned} Q_{ii} &= \frac{1}{\max(x_i)^2} \\ R_{ii} &= \frac{1}{\max(u_i)^2}. \end{aligned} \quad (4.6)$$

Bryson's Rule does not have such a direct physical parameter for the integral of the error, so the metric adopted to assign a weight to the increased states is based on the tolerance time for the steady-state error to be nullified. For roll and pitch, the tolerable time to stabilize a 10-degree error is considered to be 0.1 seconds. For yaw, which has slower dynamics, the tolerable time to stabilize a 20-degree error is considered to be 1 second. Therefore,

$$\max(\varepsilon_\phi) = \max(\varepsilon_\theta) = 10 \cdot 0.1 = 1$$

$$\max(\varepsilon_\psi) = 20 \cdot 1 = 1$$

The complete list of maximum acceptable values for all variables is presented in Table 1.

Table 1 – Table showing the maximum acceptable values for states and inputs used to calculate weights based on Bryson's Rule.

Weighted Variable	Maximum Allowed Value
ϕ	5°
θ	5°
ψ	10°
p	$90^\circ/s$
q	$90^\circ/s$
r	$60^\circ/s$
ε_ϕ	1
ε_θ	1
ε_ψ	20
U_2	$1 \text{ N} \cdot \text{m}$
U_3	$1 \text{ N} \cdot \text{m}$
U_4	$1 \text{ N} \cdot \text{m}$

Bryson's rule, while satisfying the requirements of controlled systems in most cases, is only a starting point for tuning. The ideal gains for each case are usually adjusted by trial and error or metaheuristic algorithms. In the case of drone tuning, few adjustments were needed

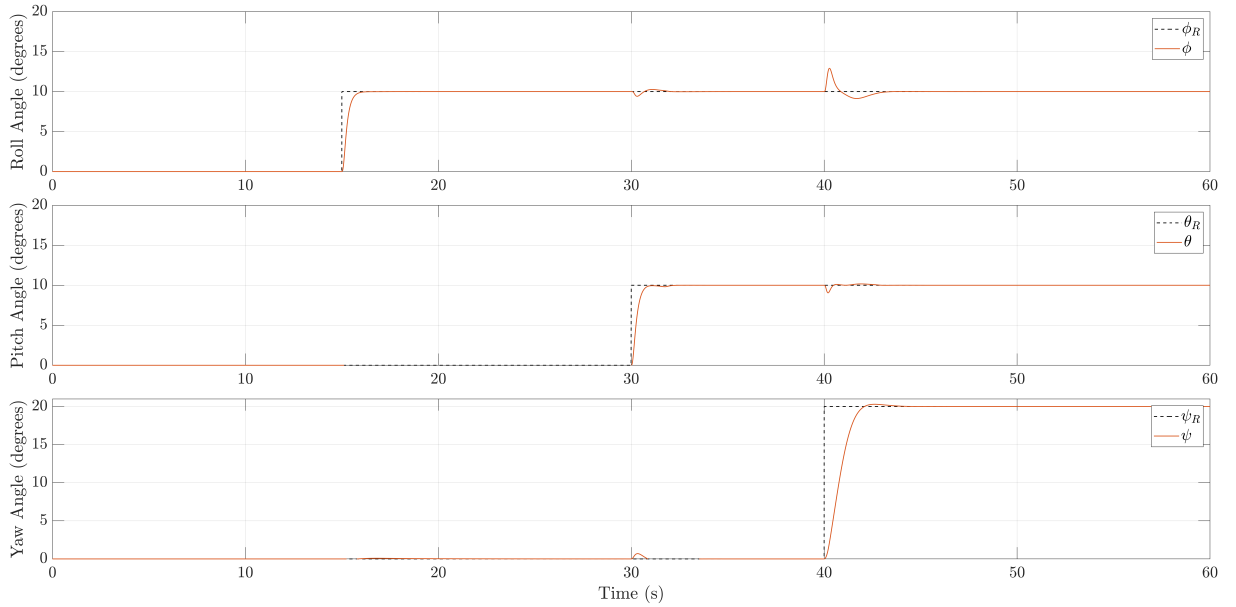
reduce the yaw angle settling time. After the adjustments, the gain matrix for controlling the rotational subsystem is:

$$Q = \begin{bmatrix} 131.3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 131.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 32.8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3282.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3282.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 80 \end{bmatrix} \quad (4.7)$$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (4.8)$$

The performance of this controller is verified by step inputs in the simulated quadcopter system, as shown in Figure 27. This simulation model eliminates the cascade position controller.

Figure 27 – Step responses used to verify attitude control performance.



Source: The author (2026).

All the linearized points used the same Q and R weighting matrices for the LQI controller design. All the gain matrices K used by the gain-scheduling algorithm are shown in Appendix D.

4.4 UAV Assembly and Experimental Approach

The proposed project, shown in Figure 28, uses a customized quadcopter based on the DJI F450 frame, in which the letter *F* denotes the number of propellers equal to four and the number 450 the distance, in millimeters, from one propeller to the other on the same diagonal. The frame is used in conjunction with 10-inch propellers, which have a maximum width of 4.5 inches (1045 standard). This is a widely used platform for hobbyists and research prototypes.

The drone has four 30A ESCs soldered on the F450 center part of the frame, which is a power distribution circuit board. These ESCs control four 2212/920KV BLDC motors. The 920KV indicates that the motor outputs 920 revolutions per Volt. The whole system is supplied by a 5200mAh 3S Li-Po battery through a power module that delivers regulated energy to the power circuit, soldered on the power distribution board, and to the digital controller and its peripherals.

The core circuit is formed by a Pixhawk 2.4.8 flight controller, a M8N GNSS module, a PPM encoder, a RC receiver, radio telemetry receiver, and a safety switch. The Pixhawk is intalled over a vibration damper to reduce noise on the internal IMUs, and the GNSS module is installed over a carbon fiber stick to isolate M8N's compass from the electromagnetic noise caused by the BLDC motors.

Moreover, a Raspberry Pi 3 B+ is used as the companion computer. Since the 5V output of the power module can not supply the Raspberry Pi, it is used a buck step-down converter that

Figure 28 – UAV prototype used for the project The vehicle is based on the F450 frame and the Pixhawk 2.4.8 controller.



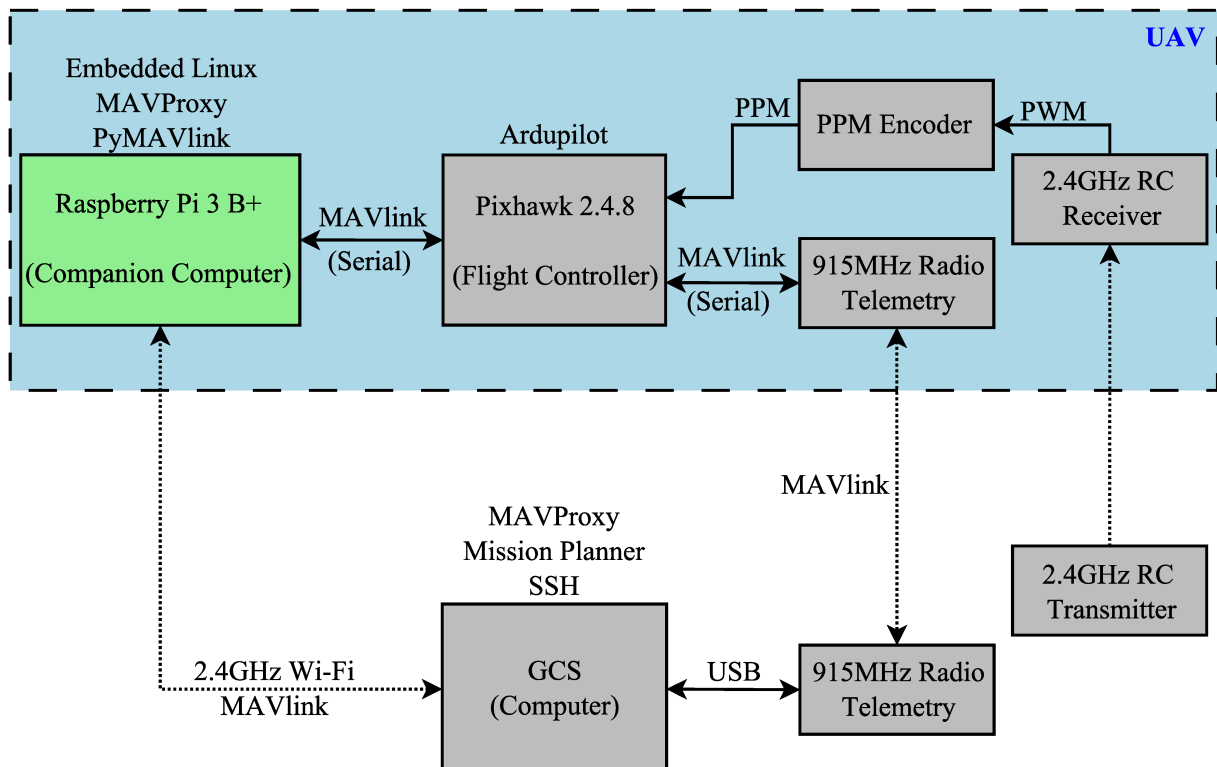
Source: The author (2026).

takes the energy directly from the power distribution board.

The UAV has a variety of communication interfaces between its components and between the drone and the ground station. Many of these interfaces were used during the experimental procedures for different purposes. For unitary tests on the vehicle it is used the direct connection between the computer and the flight controller using the 915MHz radio telemetry connected through a MAVlink protocol interface controlled by a MAVProxy command terminal. Tests using the companion computer was carried using a SSH interface with Wi-Fi network. In this cases, the companion computer was used to connect with the controller with a serial MAVProxy interface or to run Python scripts that manage autonomous missions. The missions implemented used mainly the AUTO flight mode.

Even that it was mostly not used, the UAV always operated with a radio controller as a fail-safe rule. The RC receiver gets the control signals at 4 RF channels and transforms it in 4 PWM signals. A encoder takes these 4 signals and transforms in a single PPM encoded signal passed to the flight controller. The complete representation of the communication interfaces in the drone prototype is shown in Figure 38.

Figure 29 – Communication interfaces between UAV and GCS and between UAV modules.



Source: The author (2026).

5

Results

This chapter presents the results obtained with the implementation of the model, the replication of the Ardupilot control in simulation, and the proposition, design, and implementation of a new controller for quadcopter trajectory tracking. Finally, the performance of both the Ardupilot PID controllers and their optimal control-based replacement will be evaluated in different situations.

5.1 Simulation Model Validation

The first validation step consisted of verifying whether the dynamic model was consistent with the real behavior of a quadcopter. For this, a Model-in-the-Loop of the dynamic system and the control loops of Ardupilot, described in Chapter 3 and Section 4.2, were implemented with MATLAB/Simulink.

The trajectory planning block was implemented using Ardupilot SITL flight logs. Python scripts command a simulated UAV in AUTO mode, where waypoint navigation (WPNav) interpolates trajectory setpoints. Resulting simulation logs are processed into a CSV of trajectory setpoints for use in the Simulink model.

An implemented attitude planner simulink block transforms the control signals of linear speeds in the xy plane into ϕ and θ setpoints. The angular references are calculated using trigonometric operations with the u_x and u_y virtual inputs. The detailed transformation is deduced in Section B.1 of Appendix B.

The controller, the rigid body dynamics, and the rotor dynamics blocks were implemented following the concepts developed in Chapter 3, using custom Simulink blocks and compiled MATLAB functions.

The controllers used the default gains of the Ardupilot control firmware. In this configu-

ration, the feedforward gains are all considered to be zero, and both the slew rate limiter and PD sum maximum features are turned off. The gains of all PID, PI and P controllers and the digital filters cutoff frequencies are shown in Table 2. The table also shows the maximum integration values for the anti wind-up algorithms and the maximum acceleration for the SQRT controllers.

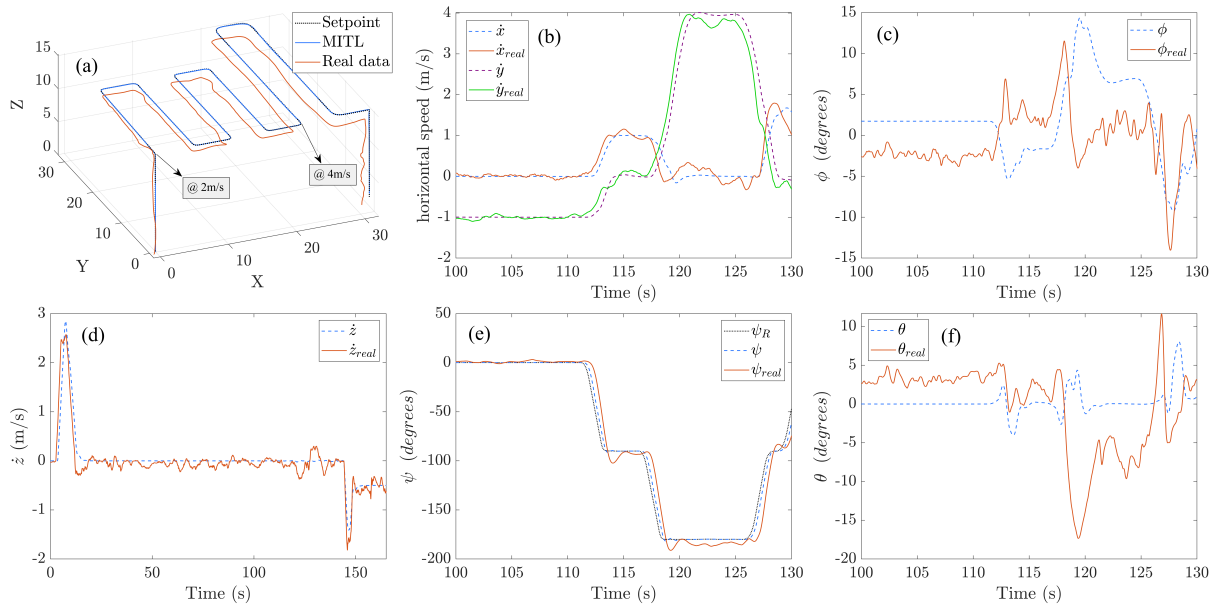
Table 2 – Ardupilot controller parameters used in the simulations.

	Parameter name	Parameter value
Roll Rate (p)	P	0.135
	I	0.135
	D	0.0036
	FLTD	20Hz
	FLTT	20Hz
	I_{max}	0.5
Pitch Rate (q)	P	0.135
	I	0.135
	D	0.0036
	FLTD	20Hz
	FLTT	20Hz
	I_{max}	0.5
Yaw Rate (r)	P	0.18
	I	0.018
	FLTD	20Hz
	FLTE	2.5Hz
	FLTT	20Hz
	I_{max}	0.5
Roll Angle (ϕ)	P	4.5
	a_{max}	110000 centidegrees/s ²
Pitch Angle (θ)	P	4.5
	a_{max}	110000 centidegrees/s ²
Yaw Angle (ψ)	P	4.5
	a_{max}	27000 centidegrees/s ²
Z Acceleration ($\ddot{z}$)	P	0.5
	I	1
	FLTE	20Hz
	I_{max}	800 d%
Z Velocity ($\dot{z}$)	P	5
	FLTE	5Hz
	FLTD	5Hz
XY Velocity ($\dot{x}$ and $\dot{y}$)	P	2
	I	1
	D	0.25
	FLTD	5Hz
	FLTE	5Hz
	I_{max}	1000
XY Position (x and y)	P	1
Z Position (z)	P	1

The same Python script used to generate the trajectory in the SITL was executed in the companion computer of the real F450 quadcopter. The trajectory consists in a *zig-zag* path with constant altitude. In the first two sections the UAV flies at 2 m/s , and in the last section at 4 m/s .

Outputs for an AUTO waypoint trajectory, using the Simulink MITL, are shown in Figure 30. It is compared with a flight of the real quadcopter. The linear speed and yaw controls matched well between the model and real system, while the final xy position showed a reasonable bias due to low-cost GPS limitations.

Figure 30 – AUTO flight comparison between the WPNav setpoints and both real and MITL trajectories, in which: (a) is the 3D position, (b) is the xy-plane speeds, (c) is the roll angle, (d) is the z-axis speed, (e) is the yaw angle and (f) the pitch angle.



Source: The author (2026).

Finally, roll and pitch showed the largest mismatch, with a constant bias at 1 m/s x-axis speed, with high amplitude peaks. It is important to note that the autonomous flight experiments were intended as preliminary validation tests. Therefore, a statistical quantitative analysis would require additional trials under controlled conditions to filter out disturbances, noise, and measurement uncertainties. Since constant winds cause bias and wind gusts cause transient disturbances on the model, overlaying data from multiple missions would be essential for a more thorough validation of the model.

Nevertheless, a quantitative analysis was performed on the tests. For this, the Root Mean-Squared Error (RMSE) and Normalized Root Mean-Squared Error metrics were used. The RMSE measures the accumulated difference between the MITL trajectory and the actual autonomous mission. The normalized error is taken as the same RMSE divided by the standard deviation of the actual flight mean. It shows the significance of the absolute error value in relation

to the mean error. These indices are defined by the Equation 5.1 and:

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^N (y - \hat{y})^2}, \quad (5.1)$$

where y is an output of the real system and $\hat{y}$ an output of the simulated system, and equation 5.2

$$NRMSE = \frac{RMSE}{\sigma}, \quad (5.2)$$

where σ is the standard deviation of the real flight mean.

The results of this analysis are shown in Table 3. It also shows which was the maximum error registered along the trajectory. The positional deviations, seemingly significant in the previous graphs, despite presenting high absolute values, are small proportionally to what the trajectory demands. The linear velocities show a satisfactory match considering the amount of disturbances in the actual flight. Finally, the truly considerable deviations appear in the angles, especially roll and pitch. It is impossible to determine from a single flight test whether such errors are entirely attributable to disturbances or whether there are also inconsistencies in the determinations of parameters that affect torques U_1 and U_2 . Among the possible parameters for poor estimation are: I_{xx} , I_{yy} , a and k_d .

Table 3 – Quantitative analysis of deviations between the simulated model and a real flight.

States	RMSE	NRMSE%	Maximum Error
x	1.5200 m	14.68	2.89 m
y	2.1024 m	24.87	4.76 m
z	0.1938 m	5.17	0.30 m
$\dot{x}$	0.2216 m/s	53.08	0.92 m/s
$\dot{y}$	0.2108 m/s	16.12	1.03 m/s
$\dot{z}$	0.1955 m/s	37.94	0.69 m/s
ϕ	5.0153°	163.40	20.53°
θ	4.4805°	104.19	21.59°
ψ	26.5190°	33.67	40.60°

Even so, this model is considered satisfactory for an initial analysis of the Ardupilot control system and fully adequate for the development of LQR/LQI controllers.

5.2 Analysis of Simulated Control Systems

For the comparison tests between the control systems, a zigzag trajectory with sections of different sizes was used. In each section, the cruising speed xy changes, being: 1.5 m/s, 3 m/s, and 4.5 m/s. These speeds coincide with 3 of the values defined for the linearization trim-points. Therefore, by testing them, the controller's performance is verified for different designed gain matrices. The trajectory is illustrated in Figure 31, from a screenshot of SITL.

The flights were verified in terms of performance using IAE, ISU, and TV. Error analysis was performed on the x , y , z , ϕ , θ , and ψ states. Cost analysis was performed on the inputs U_1 , U_2 , U_3 , U_4 and the engine speed setpoints Ω . For engine speeds, indices were calculated for each actuator separately, and then the average of these indices was taken.

$$ISU = \frac{1}{4} \sum_{i=1}^4 ISU(\Omega_i) \quad (5.3)$$

$$ISU = \frac{1}{4} \sum_{i=1}^4 TV(\Omega_i) \quad (5.4)$$

Figure 31 – Visualization of the test trajectory presented in a SITL window.



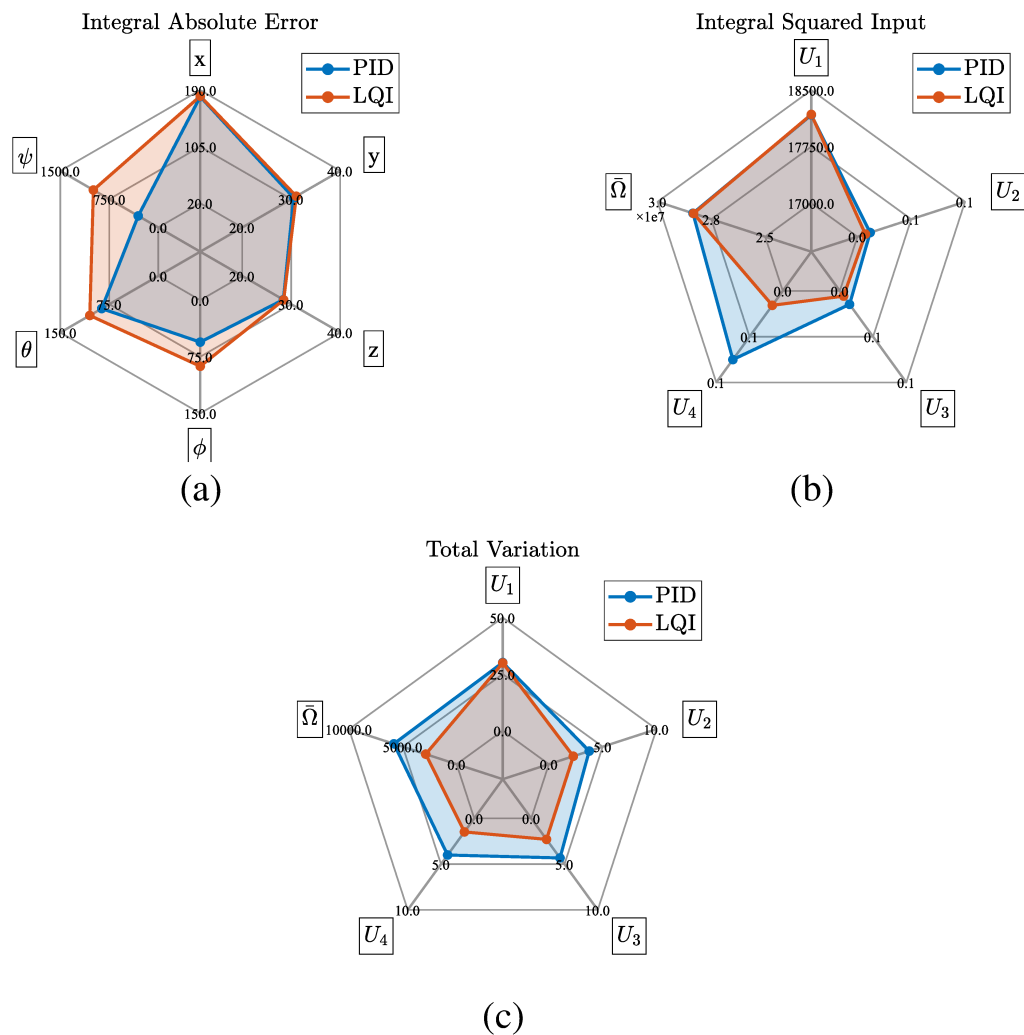
Source: The author (2026).

The results show that for the tuning initially obtained for the optimal controller and for the default gains of the ArduPilot, both controllers exhibit similar trajectory performance, with the PID controller even outperforming the optimal controller in some cases. For roll and pitch, the performance is quite similar, with the PID controller showing just over ten degrees of accumulated error for ϕ and θ . The largest discrepancy between the controllers is found for yaw, as the accumulated error for this angle is hundreds of degrees greater for the LQI. Since both use the same PID controllers for the translational subsystem, the result is that the performances for x , y , and z trajectory tracking are practically identical. This result is shown graphically in the figure 32.(a). The results of the performance indices become even clearer when observing the curves for each of the variables in the Figure 33.

For the criterion that measures the energy cost of control, the ISU, the LQI obtains slightly better results for pitch and yaw torque. This behavior is expected, precisely because these attitude controllers were less aggressive when compared to the PID, delivering greater error and lower energy consumption.

Finally, the only substantial advantage of the LQI lies in the rate of change achieved by the actuators. As can be seen in the char of Figure 32, the rate of change is always less than or equal to that achieved by the PID. This becomes even more evident when observing the angular rates of change in Figure 34, or in how the speeds requested from the motors behave, in Figure 35.

Figure 32 – Comparison between the performance indices of the PID and optimal controllers.



Source: The author (2026).

The reduction in absolute error, energy consumption, and actuator variation can be represented numerically in a general way based on the average of the IAE, ISU, and TV indices. Table 4 presents the average performance indices for the drone with attitude control using both PID and LQI.

Table 4 – Average values of the IAE, ISU, and TV indices for the PID and LQI attitude controllers tested for tracking the same trajectory.

Mean Index	Value
$\overline{IAE}_{PID}$	41.1406
$\overline{IAE}_{LQI}$	43.5199
$\overline{ISU}_{PID}$	5.6748e6
$\overline{ISU}_{LQI}$	5.6693e6
$\overline{TV}_{PID}$	1.1738e3
$\overline{TV}_{LQI}$	581.5778

For each of the indices, the reduction achieved by applying the optimal control strategy is calculated as shown in Equations 5.5, 5.6, and 5.7:

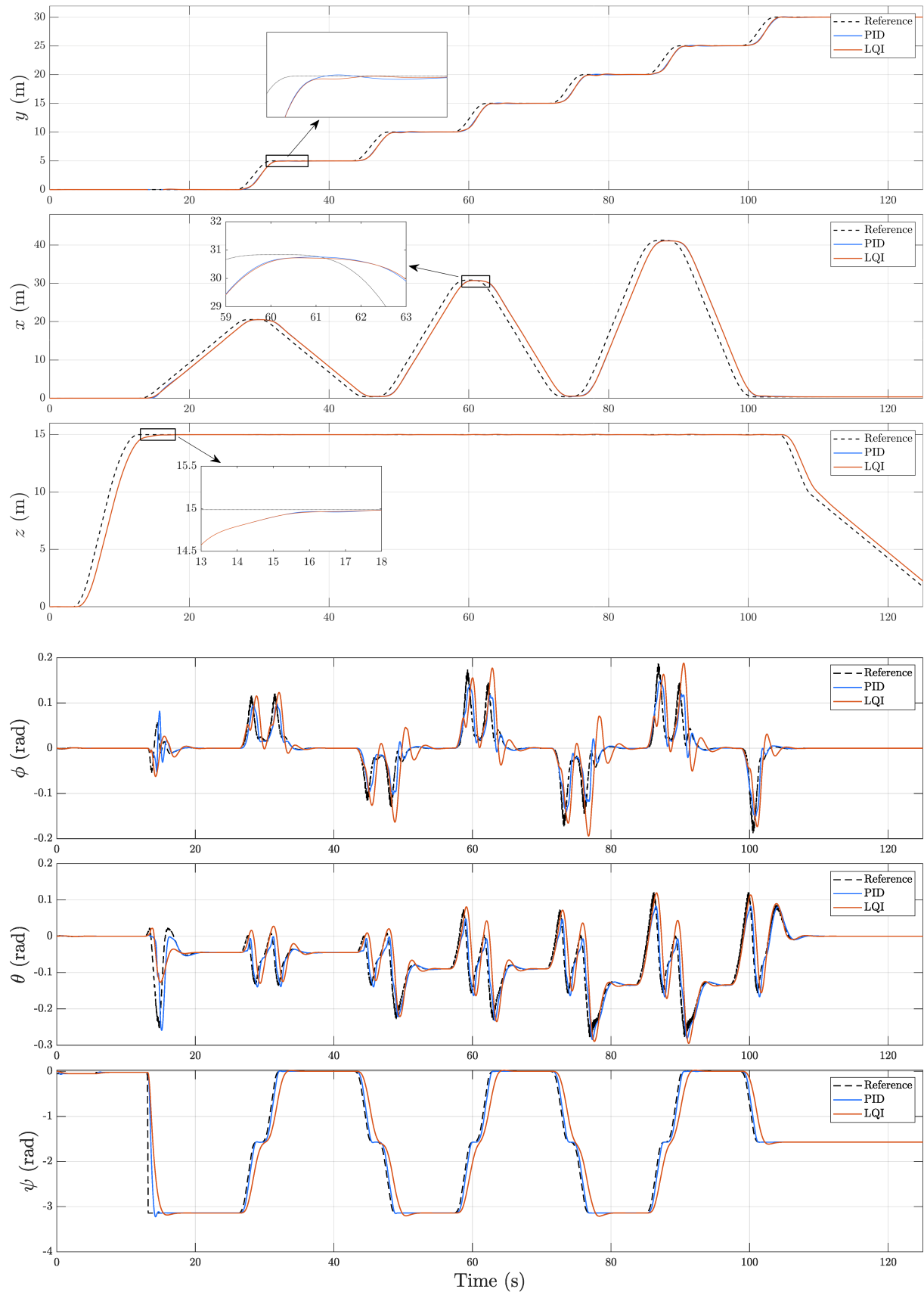
$$decrease_{\%} = \frac{\overline{IAE}_{PID} - \overline{IAE}_{LQI}}{\overline{IAE}_{PID}} = -5,76\%, \quad (5.5)$$

$$decrease_{\%} = \frac{\overline{ISU}_{PID} - \overline{ISU}_{LQI}}{\overline{ISU}_{PID}} = 0.1\%, \quad (5.6)$$

$$decrease_{\%} = \frac{\overline{TV}_{PID} - \overline{TV}_{LQI}}{\overline{TV}_{PID}} = 50,45\%. \quad (5.7)$$

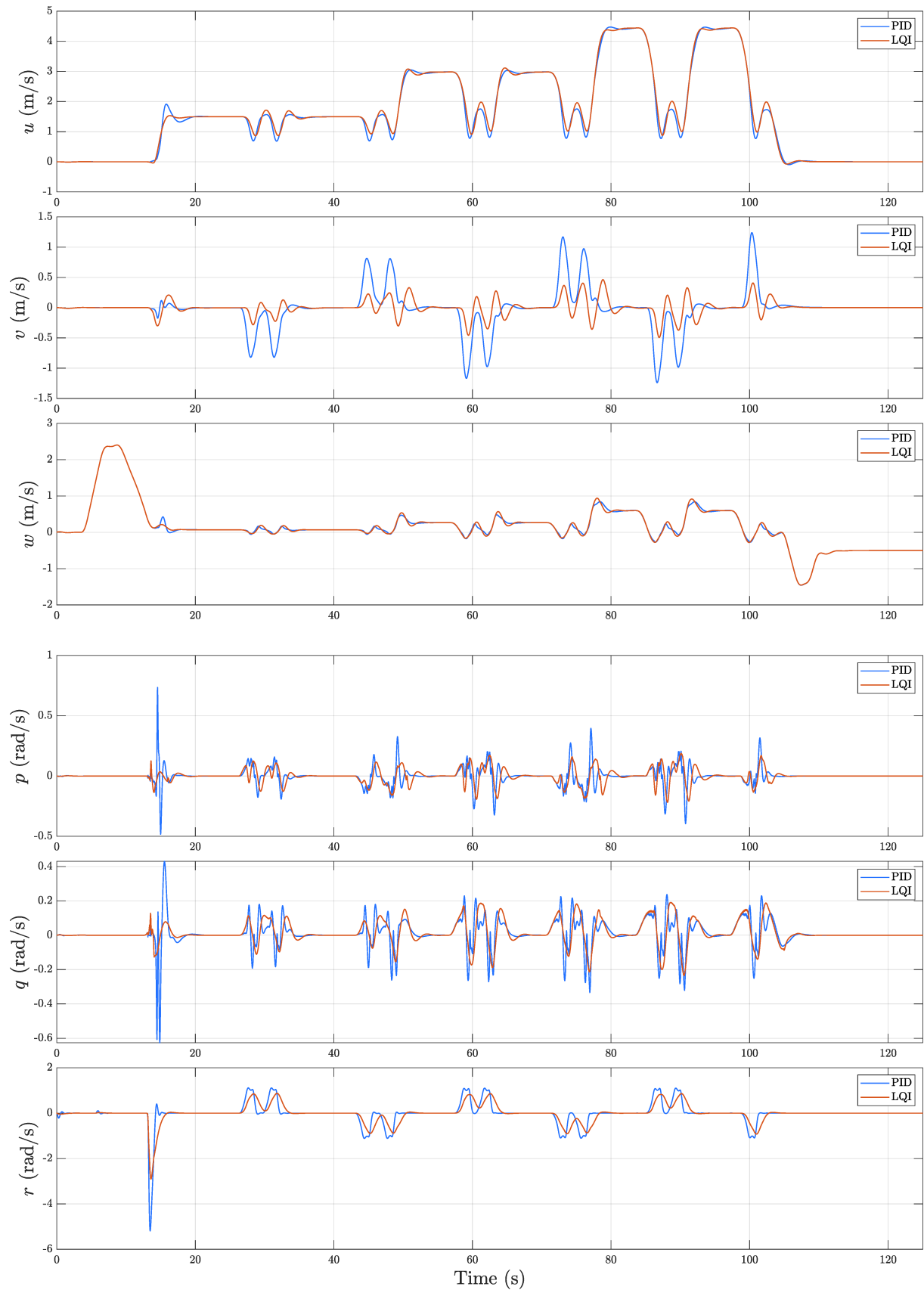
These results show that while the trajectory tracking error increases by 5.76% when switching from the PID attitude controller to the LQI, both the ISU and TV metrics decrease. However, the reduction in control cost is very small, as evidenced by the 0.1% decrease in the ISU. This is because the reductions occur primarily in torques U_2 , U_3 , and U_4 , whereas the bulk of energy consumption stems from the thrust input U_1 . In contrast, the decrease in actuator rate of change is substantial, with the average TV dropping by 50.45%. Thus, even when only the attitude control is optimized for cost reduction, the inputs are significantly adjusted to reduce excursion.

Figure 33 – Comparison between position and attitude of the PID and LQI controlled systems in simulation.



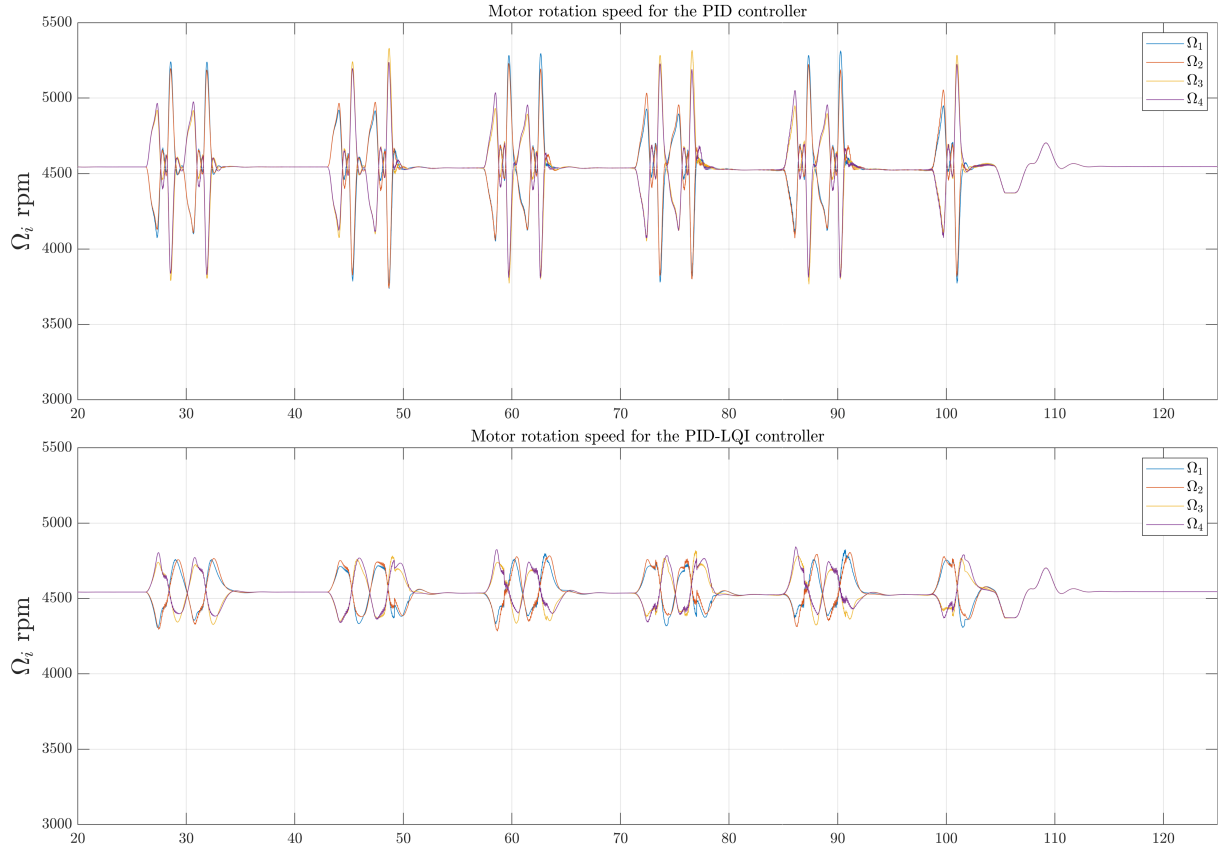
Source: The author (2026).

Figure 34 – Comparison between linear velocities and angular rates of the PID and LQI controlled systems in simulation.



Source: The author (2026).

Figure 35 – Comparison between BLDC motor speeds of the PID and LQI controlled systems in simulation.



Source: The author (2026).

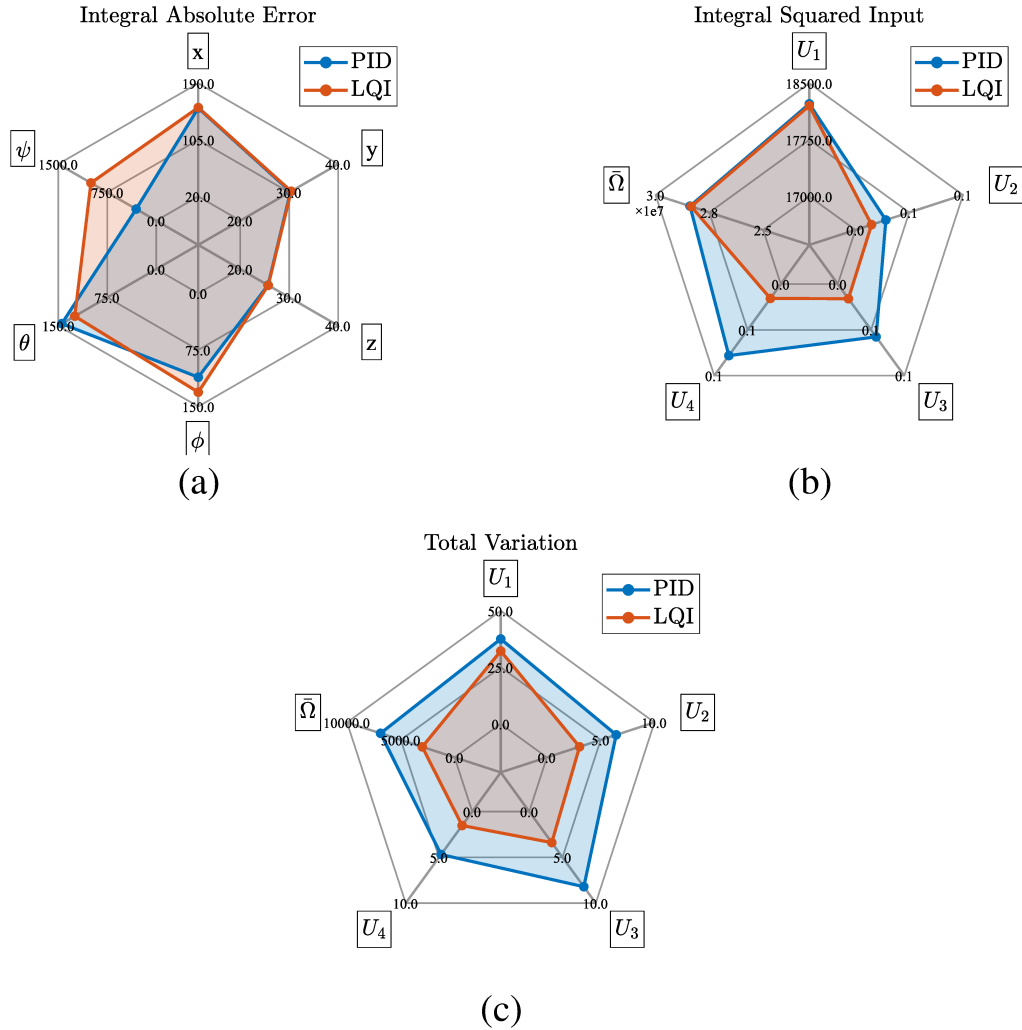
It can be observed that in both cases the controller exhibits reduced trajectory tracking capability in sections where the velocity profile has a non-zero derivative. This is an expected characteristic for controllers that do not use an internal model, such as PID and LQI controllers.

The classical way to partially solve this problem is by adding a feedforward action. The Ardupilot already contains this feature, and it can be used in the translational PIDs to improve the regular tracking performance. Since both the original Ardupilot controllers and the proposed LQI uses the same translational controller it is done to compare how the translational feedforward actions affects each strategy.

A test with intermediate feedforward gains for xy velocity, z velocity, and z acceleration is presented in the form of performance indices in Figure 36. The error, when compared to the autonomous mission tests without feedforward, is smaller for the x and z variables; these are variables that, for most of the trajectory, exhibit velocity setpoints with non-zero derivatives (the sections on the y are thinner and the UAV has no time to perform full acceleration). However, the addition of feedforward gains aggressively degrades the energy cost and, mainly, the total variation of the system, as observed in Figures 36.(a) and 36.(b).

A necessary step in validating a gain schedule is to verify that the interpolation adequately approximates the variation of gains and steady-state values outside the operating points for which

Figure 36 – Comparison between the performance indices of the PID and optimal controllers with feedforward gains in the linear speed control loops.



Source: The author (2026).

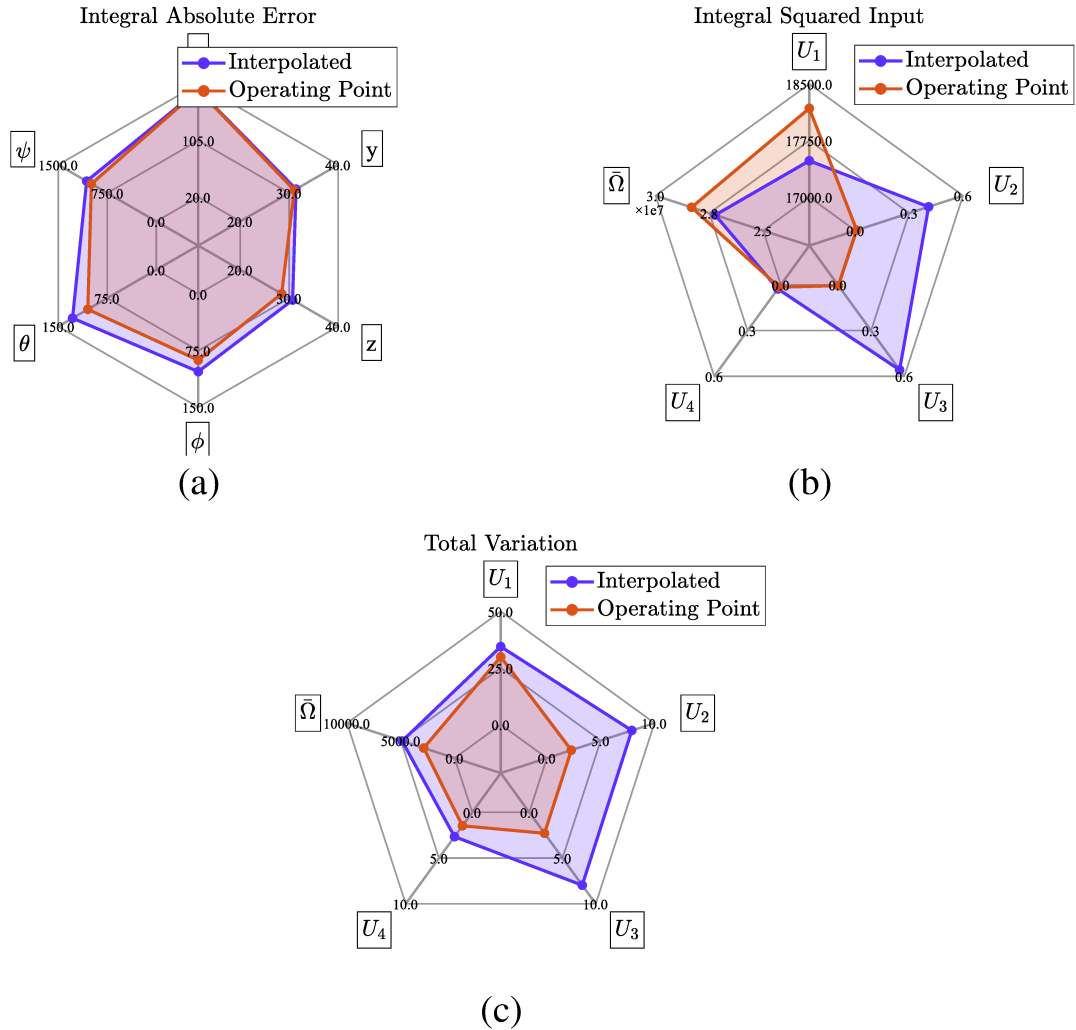
the controller was designed. Since state matrices can vary non-linearly between operating points, this can result in interpolated gain matrices that do not regulate the system at unmapped operating points.

In an initial check, the same trajectory shown in Figure 31 is tested with a new set of xy velocities for which the LQIs were not designed: $2m/s$, $3.5m/s$ and $5m/s$.

It can be seen from Figure 37 that the performance of the controller with interpolated gains is practically the same as that with analytically calculated gains for accumulated error. There is only a subtle difference for the error of the variables z , ϕ and θ .

For ISU, the control cost is much higher for mission testing with the original operating points. This is because reaching the same waypoints at lower speeds takes longer, requiring the drone to maintain its thrust for a longer period and consequently increasing the accumulated value $ISU(U_1)$. Since the variables Ω_i are proportional to U_1 , the energy consumed by the

Figure 37 – Comparison between the performance indices of the optimal controller operating both at linearized operating points and at operating points with interpolated K , X_{ss} and U_{ss} .



Source: The author (2026).

propellers is also greater. In this case, the fact that the gains were calculated or interpolated was not responsible for the change in the index, but rather the trajectory analyzed.

Although U_2 and U_3 appear much larger (and they actually are larger in the interpolated trajectory due to greater roll and pitch aggressiveness), it should be considered that, on a larger scale, the cost to hover is substantially greater than the energy expended by the torques.

The reason for the higher TV on the trajectory using interpolated gains is not related to the fit of the gains used, but rather to the aggressive profile of the route with higher cruising speeds. For such a magnitude so dependent on the trajectory, the performance index is not very representative.

It is important to emphasize that the linear interpolation only satisfies a flight envelope that does not deviate too much from the maximum values of speeds $V = 6m/s$ and $r = \pm 2rad/s$.

To expand this flight envelope, it is necessary to add more linearization points, or even identify the system's behavior between linearization points to propose better interpolation techniques that do not require an excessively large schedule table.

6

Conclusion

This work addressed one of the current challenges in Unmanned Aerial Vehicle (UAV) engineering: bridging the gap between academic innovations and practical applications. As proposed in the introduction, despite the theoretical advancement of modern control strategies, open-source autopilots and the commercial industry still predominantly rely on classic PID-based architectures. In this context, the feasibility analysis for implementing optimal controllers directly in the flight firmware proved to be of fundamental importance in filling this technological gap in the literature. By transferring consolidated control concepts to open-source flight systems, this study demonstrated that it is possible to bridge academic rigor with industrial applicability in aerial vehicles.

A major contribution of this work, however, lies in the reconstruction of the ArduPilot control architecture into a computational model and the subsequent implementation of Model-in-the-Loop (MITL) simulation. This solid simulated foundation not only enabled the theoretical and practical comparisons carried out in this research with the already established standard, but also consolidated a satisfactorily reliable testing environment. The development and validation of this MITL model opens valuable doors, offering an effective platform for the design, testing, and development of new modern controllers going forward.

When comparing the original architecture with the developed linear-quadratic-integral (LQI) optimal control strategy, the results showed a strong similarity in trajectory tracking capability. It was found that there is no absolute and inherent superiority of one technique over the other in isolation. The PID controller can, in fact, outperform the LQI in certain performance metrics, or vice versa, depending intrinsically on the tuning adopted and the weighting matrices chosen. This behavior reinforces that control effort and precision depend intimately on meticulous fine-tuning, validating the viability of both approaches under the appropriate design conditions. Although the preliminary tests indicate that further tuning and validation are still required, the proposed control architecture presents promising features and remains a relevant direction for

future development.

The main results show that replacing the PID attitude controller with the LQI, for the given trajectory, increases the trajectory tracking error by 5.76%. The decrease in control cost is marginal (0.1%), since the savings are limited to the torque inputs, whereas thrust dominates the energy consumption. In contrast, the actuator rate of change is significantly reduced, with the average TV decreasing by 50.45%, indicating substantially smoother control inputs.

Finally, aiming at expanding the flight envelope and overcoming linear limitations, the association of the optimal controller with non-linear techniques proved to be an extremely promising technical path. The implementation of approaches such as gain-scheduling allows the control to adapt dynamically to variations in aircraft dynamics at different operating points. The evolution and deepening of this strategy have great potential to improve UAV performance in adverse operating conditions, ensuring stability where fixed linear grids would fail. Thus, this research not only delivers a tangible and validated solution for the ArduPilot ecosystem, but also paves the way for the next generation of aeronautical onboard systems.

It is important to highlight that this performance similarity only considers the replacement of the rotational subsystem controller. Replacing the translational subsystem may present new performance gains or even make the aircraft's operation more flexible, given the capabilities of the proposed architecture to handle system nonlinearities. These findings reinforce the relevance of the proposed architecture and suggest that further refinement and validation may enhance its practical applicability.

6.1 Future Work

Some of the future work that would provide more solid evidence of the possibility of integrating open-source autopilots and modern control techniques includes:

- The implementation of the controller LQI in C++ within the SITL and subsequently in the real flight controller;
- The implementation of the optimal controller with gain-schedule for the translational control subsystem;
- The systematic analysis of flight performance, energy consumption, and computational load as performance metrics to be performed in real flights, similarly to what was done with IAE, ISU, and TV for simulated flights;
- The study of robustness and, if possible, an analysis of the stability of the quadcopter subjected to the original Ardupilot controllers or optimal controllers;
- Implement other Ardupilot features (Kalman filter, trajectory generator, and object avoidance algorithm) in MITL;

Bibliography

AHMDIA, A. et al. Comparative Analysis of Optimal Controllers Performance for Unmanned Aerial Vehicles. In: *2025 15th International Conference on Electrical Engineering (ICEENG)*. Cairo, Egypt: IEEE, 2025. p. 1–6. ISBN 979-8-3315-1901-8. DOI: <https://doi.org/10.1109/ICEENG64546.2025.11031304>. Available from: <https://ieeexplore.ieee.org/document/11031304>. Accessed on: 16 Apr. 2026. Cited on page 37.

ALIANE, N. A Survey of Open-Source UAV Autopilots. *Electronics*, v. 13, n. 23, p. 4785, dez. 2024. ISSN 2079-9292. DOI: <https://doi.org/10.3390/electronics13234785>. Available from: <https://www.mdpi.com/2079-9292/13/23/4785>. Accessed on: 27 Apr. 2026. Cited on page 18.

AMARAL, L. R. D. et al. UAV applications in Agriculture 4.0. *REVISTA CIÊNCIA AGRONÔMICA*, v. 51, n. 5, 2020. ISSN 1806-6690. DOI: <https://doi.org/10.5935/1806-6690.20200091>. Available from: <http://periodicos.ufc.br/revistacienciaagronomica/article/view/84923>. Accessed on: 24 Apr. 2026. Cited on page 15.

ANAC - Agência Nacional de Aviação Civil. *RBAC-E 94 EMD 03 — Agência Nacional de Aviação Civil ANAC*. 2023. Available from: <https://www.anac.gov.br/assuntos/legislacao/legislacao-1/rbha-e-rbac/rbac/rbac-e-94>. Accessed on: 10 Feb. 2026. Cited on page 14.

ARANTES, G.; MARTINS-FILHO, L. S.; SANTANA, A. C. Optimal On-Off Attitude Control for the Brazilian Multimission Platform Satellite. *Mathematical Problems in Engineering*, v. 2009, n. 1, p. 750945, jan. 2009. ISSN 1024-123X, 1563-5147. DOI: <https://doi.org/10.1155/2009/750945>. Available from: <https://onlinelibrary.wiley.com/doi/10.1155/2009/750945>. Accessed on: 28 Apr. 2026. Cited on page 17.

ArduPilot Development Team. *ArduPilot Copter — Copter documentation*. 2024. Available from: <https://ardupilot.org/copter/index.html>. Accessed on: 06 Sep. 2025. Cited 2 times on page 49 and 60.

Ardupilot Development Team. *ArduPilot Development Site*. 2024. Available from: <https://ardupilot.org/dev/index.html>. Accessed on: 06 Sep. 2025. Cited 2 times on page 21 and 59.

Ardupilot Development Team. *MAVProxy — MAVProxy documentation*. 2024. Available from: <https://ardupilot.org/mavproxy/#>. Accessed on: 06 Sep. 2025. Cited on page 62.

ASTRÖM, K. J.; HÄGGLUND, T. *Automatic tuning of PID controllers*. Research Triangle Park, NC: Instrument Society of America, 1988. ISBN 978-1-55617-081-2. Cited on page 21.

BALAS, G. J. Flight Control Law Design: An Industry Perspective. *European Journal of Control*, v. 9, n. 2-3, p. 207–226, jan. 2003. ISSN 09473580. DOI: <https://doi.org/10.3166/ejc.9.207-226>. Available from: <https://linkinghub.elsevier.com/retrieve/pii/S0947358003702763>. Accessed on: 28 Apr. 2026. Cited on page 17.

BLACKMORE, L.; AÇIKMEŞE, B.; SCHARF, D. P. Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization. *Journal of Guidance, Control, and Dynamics*, v. 33, n. 4, p. 1161–1171, jul. 2010. ISSN 0731-5090, 1533-3884. DOI: <https://doi.org/10.2514/1.47202>. Available from: <https://arc.aiaa.org/doi/10.2514/1.47202>. Accessed on: 28 Apr. 2026. Cited on page 17.

BLIGHT, J. D.; DAILEY, R. L.; GANGSAAS, D. Practical control law design for aircraft using multivariable techniques. *International Journal of Control*, v. 59, n. 1, p. 93–137, jan. 1994. ISSN 0020-7179, 1366-5820. DOI: <https://doi.org/10.1080/00207179408923071>. Available from: <https://www.tandfonline.com/doi/full/10.1080/00207179408923071>. Accessed on: 28 Apr. 2026. Cited on page 17.

BOUABDALLAH, S. *Design and control of quadrotors with application to autonomous flying*. Tese (Doutorado) — Lausanne, EPFL, 2007. DOI: <https://doi.org/10.5075/EPFL-THESIS-3727>. Accessed on: 04 Sep. 2025. Cited 4 times on page 41, 44, 48, and 49.

BOUABDALLAH, S.; NOTH, A.; SIEGWART, R. PID vs LQ control techniques applied to an indoor micro quadrotor. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sendai, Japan: IEEE, 2004. p. 2451–2456. ISBN 978-0-7803-8463-7. DOI: <https://doi.org/10.1109/IROS.2004.1389776>. Accessed on: 23 Sep. 2025. Cited 2 times on page 21 and 30.

BRASIL. Instituto Nacional de Pesquisas Espaciais. *Sobre o Satélite*. 2024. Available from: <https://www.gov.br/inpe/pt-br/programas/amazonia1/sobre-o-satelite/sobre-o-satelite>. Accessed on: 28 Apr. 2026. Cited on page 17.

BURGGRÄF, P. et al. Quadrotors in factory applications: design and implementation of the quadrotor's P-PID cascade control system: Modeling and implementation. *SN Applied Sciences*, v. 1, n. 7, p. 722, jul. 2019. ISSN 2523-3963, 2523-3971. DOI: <https://doi.org/10.1007/s42452-019-0698-7>. Accessed on: 30 Sep. 2025. Cited on page 63.

CARAC - Canadian Aviation Regulatory Advisory. *Canadian Aviation Regulations (SOR/96-433)*. AARBH, 2025. Available from: <https://tc.canada.ca/en/corporate-services/acts-regulations/list-regulations/canadian-aviation-regulations-sor-96-433>. Accessed on: 10 Feb. 2026. Cited on page 14.

CHATZARGYROS, G. et al. UAV Inspections of Power Transmission Networks with AI Technology: A Case Study of Lesbos Island in Greece. *Energies*, v. 17, n. 14, p. 3518, jul. 2024. ISSN 1996-1073. DOI: <<https://doi.org/10.3390/en17143518>>. Available from: <<https://www.mdpi.com/1996-1073/17/14/3518>>. Accessed on: 27 Apr. 2026. Cited on page 16.

CHEN, C.-J. et al. Identification of Fruit Tree Pests With Deep Learning on Embedded Drone to Achieve Accurate Pesticide Spraying. *IEEE Access*, v. 9, p. 21986–21997, 2021. ISSN 2169-3536. DOI: <<https://doi.org/10.1109/ACCESS.2021.3056082>>. Available from: <<https://ieeexplore.ieee.org/document/9343827/>>. Accessed on: 04 May 2026. Cited on page 16.

CHOVANCOVÁ, A. et al. Mathematical Modelling and Parameter Identification of Quadrotor (a survey). *Procedia Engineering*, v. 96, p. 172–181, 2014. ISSN 18777058. DOI: <<https://doi.org/10.1016/j.proeng.2014.12.139>>. Accessed on: 04 Sep. 2025. Cited on page 42.

COWLING, I. D. et al. A prototype of an autonomous controller for a quadrotor UAV. In: *2007 European Control Conference (ECC)*. Kos, Greece: IEEE, 2007. p. 4001–4008. ISBN 978-3-9524173-8-6. DOI: <<https://doi.org/10.23919/ECC.2007.7068316>>. Available from: <<https://ieeexplore.ieee.org/document/7068316/>>. Accessed on: 19 May 2026. Cited on page 31.

Drone Rich. *Cheerson CX-10 Review and Indoor Flight (Unboxing and Review)*. 2017. Available from: <<https://www.youtube.com/watch?v=NwXUb-p2rsg>>. Accessed on: 09 Feb. 2026. Cited on page 15.

FAA - Federal Aviation Administration. *Joint Air Operations*. [S.l.], 2019. Cited on page 14.

GERTLER, J.; ZORETICH, T. *World Civil Unmanned Aerial Systems: MARKET PROFILE & FORECAST 2022/2023*. Fairfax, Virginia, USA, 2022. Cited on page 15.

GUROCAK, H. *Industrial motion control: motor selection, drives, controller tuning, applications*. West Sussex: Wiley, 2016. ISBN 978-1-118-35081-2. DOI: <<https://doi.org/10.1002/9781118403211>>. Cited 2 times on page 29 and 30.

HAMZA, A.; MOHAMED, A. H.; EL-BADAWY, A. Robust H-infinity Control for a Quadrotor UAV. In: *AIAA SCITECH 2022 Forum*. San Diego, CA & Virtual: American Institute of Aeronautics and Astronautics, 2022. ISBN 978-1-62410-631-6. DOI: <<https://doi.org/10.2514/6.2022-2033>>. Available from: <<https://arc.aiaa.org/doi/10.2514/6.2022-2033>>. Accessed on: 03 May 2026. Cited on page 18.

HOFFMANN, F.; GODDEMEIER, N.; BERTRAM, T. Attitude estimation and control of a quadcopter. In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Taipei: IEEE, 2010. p. 1072–1077. ISBN 978-1-4244-6674-0. DOI: <<https://doi.org/10.1109/IROS.2010.5649111>>. Available from: <<http://ieeexplore.ieee.org/document/5649111/>>. Accessed on: 23 Sep. 2025. Cited on page 22.

HOFFMANN, G. et al. Quadrotor Helicopter Flight Dynamics and Control: Theory and Experiment. In: *AIAA Guidance, Navigation and Control Conference and Exhibit*. Hilton Head, South Carolina: American Institute of Aeronautics and Astronautics, 2007. ISBN 9781624100154. DOI: <<https://doi.org/10.2514/6.2007-6461>>. Available from: <<https://arc.aiaa.org/doi/10.2514/6.2007-6461>>. Accessed on: 24 Sep. 2025. Cited on page 22.

JOUKHADAR, A. et al. Trajectory Tracking Control of a Quadcopter UAV Using Nonlinear Control. In: RIZK, R.; AWAD, M. (Ed.). *Mechanism, Machine, Robotics and Mechatronics Sciences*. Cham: Springer International Publishing, 2019. v. 58, p. 271–285. ISBN 978-3-319-89910-7. Series Title: Mechanisms and Machine Science. DOI: https://doi.org/10.1007/978-3-319-89911-4_20. Available from: http://link.springer.com/10.1007/978-3-319-89911-4_20. Accessed on: 19 May 2026. Cited on page 31.

KARAM, K. et al. Quadcopters in Smart Agriculture: Applications and Modelling. *Applied Sciences*, v. 14, n. 19, p. 9132, out. 2024. ISSN 2076-3417. DOI: <https://doi.org/10.3390/app14199132>. Available from: <https://www.mdpi.com/2076-3417/14/19/9132>. Accessed on: 11 Feb. 2026. Cited 2 times on page 14 and 15.

KHALID, A. et al. Control Schemes for Quadrotor UAV: Taxonomy and Survey. *ACM Computing Surveys*, v. 56, n. 5, p. 1–32, maio 2024. ISSN 0360-0300, 1557-7341. DOI: <https://doi.org/10.1145/3617652>. Accessed on: 27 Apr. 2026. Cited on page 16.

KIM, O.-J. et al. In-Orbit Results and Attitude Analysis of the SNUGLITE Cube-Satellite. *Applied Sciences*, v. 10, n. 7, p. 2507, abr. 2020. ISSN 2076-3417. DOI: <https://doi.org/10.3390/app10072507>. Available from: <https://www.mdpi.com/2076-3417/10/7/2507>. Accessed on: 28 Apr. 2026. Cited on page 17.

KOUBAA, A. et al. Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey. *IEEE Access*, v. 7, p. 87658–87680, 2019. ISSN 2169-3536. DOI: <https://doi.org/10.1109/ACCESS.2019.2924410>. Accessed on: 06 Sep. 2025. Cited on page 61.

LEITE, H. *Empresa de São José dos Campos se lança no mercado de drones para o agronegócio*. 2024. Available from: <https://www.edrotacultural.com.br/empresa-de-sao-jose-dos-campos-se-lanca-no-mercado-de-drones-para-o-agronegocio/>. Accessed on: 09 Feb. 2026. Cited on page 15.

LI, J.; LI, Y. Dynamic analysis and PID control for a quadrotor. In: *2011 IEEE International Conference on Mechatronics and Automation*. Beijing, China: IEEE, 2011. p. 573–578. ISBN 978-1-4244-8113-2. DOI: <https://doi.org/10.1109/ICMA.2011.5985724>. Accessed on: 23 Sep. 2025. Cited on page 22.

LI, L.; SUN, L.; JIN, J. Survey of advances in control algorithms of quadrotor unmanned aerial vehicle. In: *2015 IEEE 16th International Conference on Communication Technology (ICCT)*. Hangzhou, China: IEEE, 2015. p. 107–111. ISBN 978-1-4673-7004-2. DOI: <https://doi.org/10.1109/ICCT.2015.7399803>. Accessed on: 04 Sep. 2025. Cited 2 times on page 16 and 21.

LI, P.; LIU, D.; BALDI, S. Plug-and-play adaptation in autopilot architectures for unmanned aerial vehicles. In: *IECON 2021 – 47th Annual Conference of the IEEE Industrial Electronics Society*. Toronto, ON, Canada: IEEE, 2021. p. 1–6. ISBN 978-1-6654-3554-3. DOI: <https://doi.org/10.1109/IECON48115.2021.9589106>. Available from: <https://ieeexplore.ieee.org/document/9589106/>. Accessed on: 28 Apr. 2026. Cited on page 18.

LIMA, G. *Modelagem dinâmica e controle para navegação de um veículo aéreo não tripulado do tipo quadricóptero*. Tese (Doutorado) — Universidade Federal de Uberlândia, ago. 2015. DOI: <https://doi.org/10.14393/ufu.di.2015.385>. Accessed on: 04 Sep. 2025. Cited 4 times on page 40, 49, 58, and 96.

LIMA, G. et al. Análise Dinâmica E Controle Pid De Um Quadricóptero. In: *XIII CEEL*. Uberlândia: [s.n.], 2015. Cited on page 22.

MAHONY, R.; KUMAR, V.; CORKE, P. Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor. *IEEE Robotics & Automation Magazine*, v. 19, n. 3, p. 20–32, set. 2012. ISSN 1070-9932, 1558-223X. DOI: <https://doi.org/10.1109/MRA.2012.2206474>. Accessed on: 29 Sep. 2025. Cited 2 times on page 38 and 39.

MARTINI, S. et al. Euler-Lagrange Modeling and Control of Quadrotor UAV with Aerodynamic Compensation. In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. Dubrovnik, Croatia: IEEE, 2022. p. 369–377. ISBN 978-1-6654-0593-5. DOI: <https://doi.org/10.1109/ICUAS54217.2022.9836215>. Available from: <https://ieeexplore.ieee.org/document/9836215/>. Accessed on: 08 Sep. 2025. Cited 2 times on page 44 and 48.

MARTINS, L.; CARDEIRA, C.; OLIVEIRA, P. Linear Quadratic Regulator for Trajectory Tracking of a Quadrotor. *IFAC-PapersOnLine*, v. 52, n. 12, p. 176–181, 2019. ISSN 24058963. DOI: <https://doi.org/10.1016/j.ifacol.2019.11.195>. Available from: <https://linkinghub.elsevier.com/retrieve/pii/S2405896319311450>. Accessed on: 18 May 2026. Cited on page 31.

MERIAM, J. L.; KRAIGE, L. G. *Dynamics: SI version*. 7. ed. ed. Singapore: Wiley, 2013. (Engineering mechanics / J. L. Meriam; L. G. Kraige, Vol. 2). ISBN 978-1-118-08345-1. Cited on page 44.

MUVVA, V. V. R. M. K. R. et al. Custom UAV with model predictive control for autonomous static and dynamic trajectory tracking in agricultural fields. *Frontiers in Robotics and AI*, v. 12, p. 1694952, dez. 2025. ISSN 2296-9144. DOI: <https://doi.org/10.3389/frobt.2025.1694952>. Available from: <https://www.frontiersin.org/articles/10.3389/frobt.2025.1694952/full>. Accessed on: 03 May 2026. Cited on page 18.

NAKOLA, S. *Rwanda and Chinese firms jointly launch Africa's first air taxi*. 2025. Available from: <https://global.chinadaily.com.cn/a/202509/05/WS68baeddea3108622abc9f275.html>. Accessed on: 09 Feb. 2026. Cited on page 15.

NGUYEN, K. et al. TinyMPC: Model-Predictive Control on Resource-Constrained Microcontrollers. In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, 2024. p. 1–7. ISBN 979-8-3503-8457-4. DOI: <https://doi.org/10.1109/ICRA57147.2024.10610987>. Available from: <https://ieeexplore.ieee.org/document/10610987/>. Accessed on: 28 Apr. 2026. Cited on page 18.

NIKU, S. B. *Introduction to robotics: analysis, control, applications*. Third edition. Hoboken, NJ Chichester, West Sussex: Wiley, 2020. ISBN 978-1-119-52762-6. Cited 2 times on page 40 and 41.

OKASHA, M.; KRALEV, J.; ISLAM, M. Design and Experimental Comparison of PID, LQR and MPC Stabilizing Controllers for Parrot Mambo Mini-Drone. *Aerospace*, v. 9, n. 6, p. 298, jun. 2022. ISSN 2226-4310. DOI: <https://doi.org/10.3390/aerospace9060298>. Available from: <https://www.mdpi.com/2226-4310/9/6/298>. Accessed on: 30 Sep. 2025. Cited on page 17.

OKASHA, M. et al. Gain scheduled linear quadratic control for quadcopter. *IOP Conference Series: Materials Science and Engineering*, v. 270, p. 012009, dez. 2017. ISSN 1757-8981,

1757-899X. DOI: <<https://doi.org/10.1088/1757-899X/270/1/012009>>. Available from: <<https://iopscience.iop.org/article/10.1088/1757-899X/270/1/012009>>. Accessed on: 18 May 2026. Cited on page 31.

PINTO, M. et al. Parameterization and Experimental Control of a Mini Quadcopter. In: *Proceedings of the 25th International Congress of Mechanical Engineering*. [S.l.]: ABCM, 2019. DOI: <<https://doi.org/10.26678/ABCM.COBEM2019.COB2019-2254>>. Accessed on: 08 Sep. 2025. Cited 2 times on page 22 and 49.

PX4 Development Team. *PX4 Guide*. 2025. Available from: <<https://docs.px4.io/main/en/>>. Accessed on: 14 Nov. 2025. Cited on page 21.

SHAKHATREH, H. et al. Unmanned Aerial Vehicles (UAVs): A Survey on Civil Applications and Key Research Challenges. *IEEE Access*, v. 7, p. 48572–48634, 2019. ISSN 2169-3536. DOI: <<https://doi.org/10.1109/ACCESS.2019.2909530>>. Available from: <<https://ieeexplore.ieee.org/document/8682048/>>. Accessed on: 23 Apr. 2026. Cited 2 times on page 14 and 16.

SVACHA, J.; MOHTA, K.; KUMAR, V. Improving quadrotor trajectory tracking by compensating for aerodynamic effects. In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*. Miami, FL, USA: IEEE, 2017. p. 860–866. ISBN 978-1-5090-4495-5. DOI: <<https://doi.org/10.1109/ICUAS.2017.7991501>>. Available from: <<http://ieeexplore.ieee.org/document/7991501/>>. Accessed on: 18 May 2026. Cited 2 times on page 47 and 54.

The MathWorks, Inc. *linmod - Extract continuous-time linear state-space model around operating point using block by block linearization algorithm - MATLAB*. 2026. Available from: <<https://www.mathworks.com/help/simulink/slref/linmod.html>>. Accessed on: 23 Jun. 2026. Cited on page 58.

VISIOLI, A. Anti-windup Strategies. In: *Practical PID Control*. [S.l.]: Springer London, 2006. p. 35–60. ISBN 978-1-84628-585-1. Series Title: Advances in Industrial Control. DOI: <https://doi.org/10.1007/1-84628-586-0_3>. Accessed on: 04 Sep. 2025. Cited on page 27.

VISIOLI, A. Basics of PID Control. In: *Practical PID Control*. Springer London, 2006. p. 1–18. ISBN 978-1-84628-585-1. Series Title: Advances in Industrial Control. DOI: <https://doi.org/10.1007/1-84628-586-0_1>. Available from: <http://link.springer.com/10.1007/1-84628-586-0_1>. Accessed on: 04 Sep. 2025. Cited on page 23.

VISIOLI, A. Control Structures. In: *Practical PID Control*. Springer London, 2006. p. 251–294. ISBN 978-1-84628-585-1. Series Title: Advances in Industrial Control. DOI: <https://doi.org/10.1007/1-84628-586-0_9>. Available from: <http://link.springer.com/10.1007/1-84628-586-0_9>. Accessed on: 04 Sep. 2025. Cited on page 29.

WIEPEN, D. *Army Warfighting Experiment 2017 – Testing the Next Generation of Technology*. 2017. Available from: <<https://www.defenceimagery.mod.uk>>. Accessed on: 09 Feb. 2026. Cited on page 15.

WILLIAMS, R. L.; LAWRENCE, D. A. *Linear state-space control systems*. Hoboken, N.J: John Wiley & Sons, 2007. ISBN 978-0-471-73555-7. DOI: <<https://doi.org/10.1002/9780470117873>>. Cited 2 times on page 35 and 58.

World Bank. *Drones for Development: The Current State and Future Prospects of UAS for Sustainable and Green Growth in Brazil*. Washington, DC: World Bank, 2025. DOI: <https://doi.org/10.1596/43102>. Available from: <https://hdl.handle.net/10986/43102>. Accessed on: 24 Apr. 2026. Cited on page 15.

ZHANG, H. et al. Review of the Approaches for Urban Underground Infrastructure Inspections Based on UAVs. In: *Proceedings of the 2025 2nd International Conference on Digital Society and Artificial Intelligence*. Wuhan China: ACM, 2025. p. 602–606. ISBN 979-8-4007-1433-7. DOI: <https://doi.org/10.1145/3748825.3748918>. Available from: <https://dl.acm.org/doi/10.1145/3748825.3748918>. Accessed on: 27 Apr. 2026. Cited on page 16.

ÅSTRÖM, K. J.; WITTENMARK, B. *Computer-controlled systems: theory and design*. 3. ed., unabridged, corrected republic. ed. Mineola, NY: Dover Publications, 2011. ISBN 978-0-486-48613-0. Cited 2 times on page 27 and 28.



Experimental Drone Configuration

This appendix shows, in Table 5, all the individual parts that make up the experimental drone built for this project. The constructed quadrotor has a total of 18 different parts, weighing approximately 1229g.

Table 5 – Table with the experimental drone’s configuration and the weight of the parts.

Part	Mass (g)	Quantity
F450 frame - arms	47	4
F450 frame - superior board	23	1
F450 frame - inferior board	42	1
F450 frame - landing skids	17	4
Vibration damper	27	1
PixHawk 2.4.8	38	1
LiPo Battery 5200mAH	322	1
Propellers	10	4
ESCs	23	4
GNSS/compass module	31	1
Telemetry radio	19	1
Controller radio	10	1
GPS module holder	21	1
BLDC motors	55	4
Power Module	34	1
PPM encoder	9	1
Buzzer/power switch module	4	1
Raspberry Pi 3 B+	42	1

Table 6 – Experimental drone parameters.

name	parameter	value	unit
mass	m	1.229	kg
inertia on x axis	I_{xx}	1.334985e-02	$kg \cdot m^2$
inertia on y axis	I_{yy}	1.414568e-02	$kg \cdot m^2$
inertia on z axis	I_{zz}	2.100660e-02	$kg \cdot m^2$
product of inertia xy	I_{xy}	-1.9854e-04	$kg \cdot m^2$
product of inertia yz	I_{yz}	5.4324e-04	$kg \cdot m^2$
product of inertia zx	I_{zx}	-5.4327e-04	$kg \cdot m^2$
rotor inertia	J_r	1.499495e-07	$kg \cdot m^2$
propeller thrust coefficient	a	1.487342e-08	kgf/rpm^2
propeller drag coefficient	b	2.203668e-10	$(kgf \cdot m)/rpm^2$
aerodynamic drag coefficient	k_d	1.5450e-04	$(N \cdot s^2)/(m \cdot rad)$
arm length	l	45	cm

B

Additional Mathematical Modeling

This appendix covers additional mathematical modeling omitted from the main text in order to reduce the amount of algebraic development not essential for understanding the dynamic model. However, this mathematical development provides equations used in the simulation model.

B.1 Attitude Planner

From the dynamic Equations 3.29, it is possible to see that the movement in the xy plane is coupled to the thrust input U_1 , which is responsible for the vertical movement. Therefore, we can say that there are horizontal components of U_1 , called u_x and u_y , which are responsible for the movements along the x and y axes. Thereby, the total thrust can be decomposed in terms responsible for vertical and horizontal accelerations. Firstly, the planner calculates the thrust input as follows:

$$U_1 = \frac{(\ddot{z} + g)m}{\cos \phi \cos \theta},$$

then, the horizontal components are obtained:

$$\begin{aligned} u_x &= \frac{m\ddot{x}}{U_1} \\ u_y &= \frac{m\ddot{y}}{U_1}. \end{aligned} \tag{B.1}$$

The control input U_1 is originated from the vertical throttle controller and is considered to be a time varying parameter. Each control cycle the throttle controller generates new u_x and u_y virtual inputs. Therefore, consequently, for each period of the position controller, it is possible to obtain ϕ_R and θ_R to be passed on to the rotational control of the quadcopter. As demonstrated in Lima (2015) the equations for the virtual inputs are:

$$u_x = \cos \psi \sin \theta \cos \phi - \sin \psi \sin \phi \quad (\text{B.2})$$

$$u_y = \sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi. \quad (\text{B.3})$$

The first step in isolating the angular setpoint variables ϕ_R and θ_R is to remove the yaw terms from the equations. To do this, one can apply a rotation of $-\psi$ to the u_x and u_y terms:

$$\begin{bmatrix} u'_x \\ u'_y \end{bmatrix} = \begin{bmatrix} \cos \psi & \sin \psi \\ -\sin \psi & \cos \psi \end{bmatrix} \begin{bmatrix} u_x \\ u_y \end{bmatrix},$$

which leads to de-rotated components

$$\begin{aligned} u'_x &= u_x \cos \psi + u_y \sin \psi \\ u'_y &= -u_x \sin \psi + u_y \cos \psi. \end{aligned}$$

Substituting u_x for B.2 and u_y for B.3 in the u'_x expression:

$$\begin{aligned} u'_x &= \cos \psi (\cos \psi \sin \theta \cos \phi - \sin \psi \sin \phi) + \sin \psi (\sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi) \\ u'_x &= \cos^2 \psi \sin \theta \cos \phi - \cos \psi \sin \psi \sin \phi + \sin^2 \psi \sin \theta \cos \phi - \sin \psi \cos \psi \sin \phi \\ u'_x &= \sin \theta \cos \phi (\cos^2 \psi + \sin^2 \psi) \\ u'_x &= \sin \theta \cos \phi \end{aligned}$$

Similarly, Substituting u_x for B.2 and u_y for B.3 in the u'_y expression:

$$\begin{aligned} u'_y &= -\sin \psi (\cos \psi \sin \theta \cos \phi - \sin \psi \sin \phi) + \cos \psi (\sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi) \\ u'_y &= -\sin \psi \cos \psi \sin \theta \cos \phi - \sin^2 \psi \sin \phi + \sin \psi \cos \psi \sin \theta \cos \phi - \cos^2 \psi \sin \phi \\ u'_y &= -\sin \phi (\cos^2 \psi + \sin^2 \psi) \\ u'_y &= -\sin \phi \end{aligned}$$

Finally, it is possible to isolate the roll and pitch angles by sequentially applying inverse trigonometric functions:

$$\begin{aligned} \phi &= \arcsin(-u'_y) \\ \theta &= \arcsin\left(\frac{u'_x}{\cos \phi}\right). \end{aligned}$$

Then, the expressions to calculate ϕ_R and θ_R are obtained by expanding the u'_x and u'_y terms:

$$\begin{aligned} \phi &= \arcsin(u_x \sin \psi - u_y \cos \psi) \\ \theta &= \arcsin\left(\frac{u_x \cos \psi + u_y \sin \psi}{\cos \phi}\right) \end{aligned} \quad (\text{B.4})$$

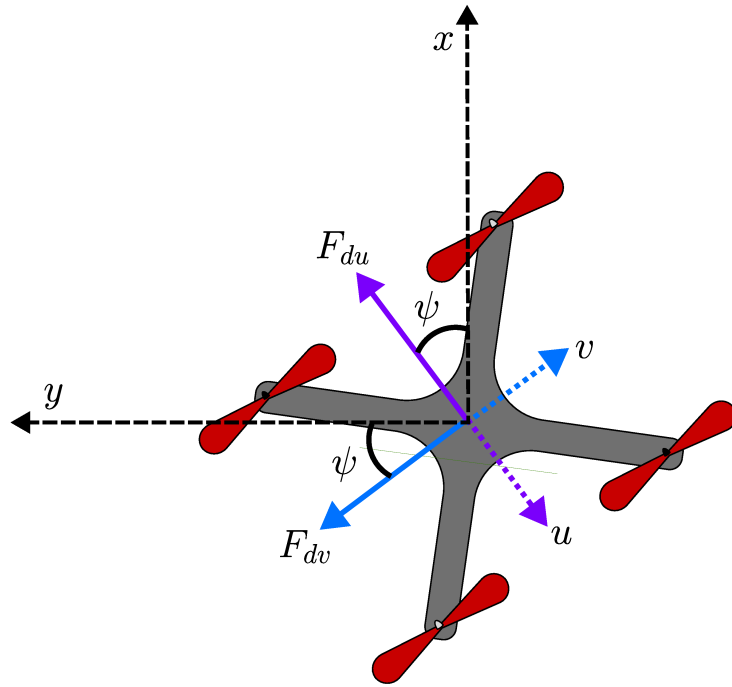
In this format, these equations can present problems during simulation. Very small roll and pitch values create division by zero problems during the calculation of U_1 . Furthermore, the

arcsine function has a defined domain in the closed interval $[-1,1]$. Trajectories that require very abrupt movements or the presence of excessively high gains in the position controllers can create situations where the drone needs roll and pitch angles greater than or equal to 90. Therefore, it is necessary to pin a minimum value of $1e-03$ to the roll and pitch angles before calculations and to clamp the arcsin function arguments to values between -1 and 1. Angles of exactly 90 degrees are also not allowed, as this would result in a gymball lock.

B.2 Aerodynamic Drag in the Earth Frame

As seen in the equation, the drag force is considered on the body frame. This is done because the drone is approximately radially symmetrical and we can consider the front part similar to its sides, and therefore only one value of k_d for velocities u and v . However, for the design of controllers, the dynamics must be considered in an inertial frame of reference, so transformations must be applied to this drag model.

Figure 38 – Free-body diagram of the UAV representing its horizontal velocities and drag forces on the body frame.



Source: The author (2026).

The drag forces in the forward (u) and left-side (v) in the body frame are decomposed in the x and y direction of the Earth frame,

$$\begin{aligned} F_{dx} &= F_{du} \cos \psi - F_{dv} \sin \psi \\ F_{dy} &= F_{du} \sin \psi - F_{dv} \cos \psi, \end{aligned}$$

and then it can be rewritten in a matrix form,

$$\begin{bmatrix} F_{dx} \\ F_{dy} \end{bmatrix} = \begin{bmatrix} \cos \psi & -\sin \psi \\ \sin \psi & \cos \psi \end{bmatrix} \begin{bmatrix} F_{du} \\ F_{dv} \end{bmatrix}.$$

Observing the matrix that pre-multiplies the forces decomposed in the body frame, it can be seen that it is a rotation transformation around the z-axis, or R_z . We can pre-multiply both sides by the inverse of the rotation in order to isolate the forces in the body frame:

$$\begin{bmatrix} \cos \psi & -\sin \psi \\ \sin \psi & \cos \psi \end{bmatrix}^{-1} \begin{bmatrix} F_{dx} \\ F_{dy} \end{bmatrix} = \begin{bmatrix} \cos \psi & -\sin \psi \\ \sin \psi & \cos \psi \end{bmatrix}^{-1} \begin{bmatrix} \cos \psi & -\sin \psi \\ \sin \psi & \cos \psi \end{bmatrix} \begin{bmatrix} F_{du} \\ F_{dv} \end{bmatrix},$$

The rotation matrix R_z is orthonormal, and then $R_z^{-1} = R_z^T$, hence

$$\begin{bmatrix} \cos \psi & \sin \psi \\ -\sin \psi & \cos \psi \end{bmatrix} \begin{bmatrix} F_{dx} \\ F_{dy} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} F_{du} \\ F_{dv} \end{bmatrix},$$

which leads to:

$$(\cos \psi - \sin \psi)F_{dx} + (\cos \psi + \sin \psi)F_{dy} = F_{du} + F_{dv}.$$

Finally, Finally, we can say that the drag force, described from the body frame and therefore in terms of $\dot{x}$ and $\dot{y}$, is

$$F_d = k_d \Omega (\cos \psi - \sin \psi) \dot{x} + k_d \Omega (\cos \psi + \sin \psi) \dot{y}.$$

From this expression, the terms $-k_d \Omega (\cos \psi - \sin \psi) \dot{x}$ and $-k_d \Omega (\cos \psi + \sin \psi) \dot{y}$, representing the contribution of the drag force to the system's dynamics, are taken and inserted into the differential equations that describe the horizontal linear movements in the body frame.

Simulink Model Diagrams

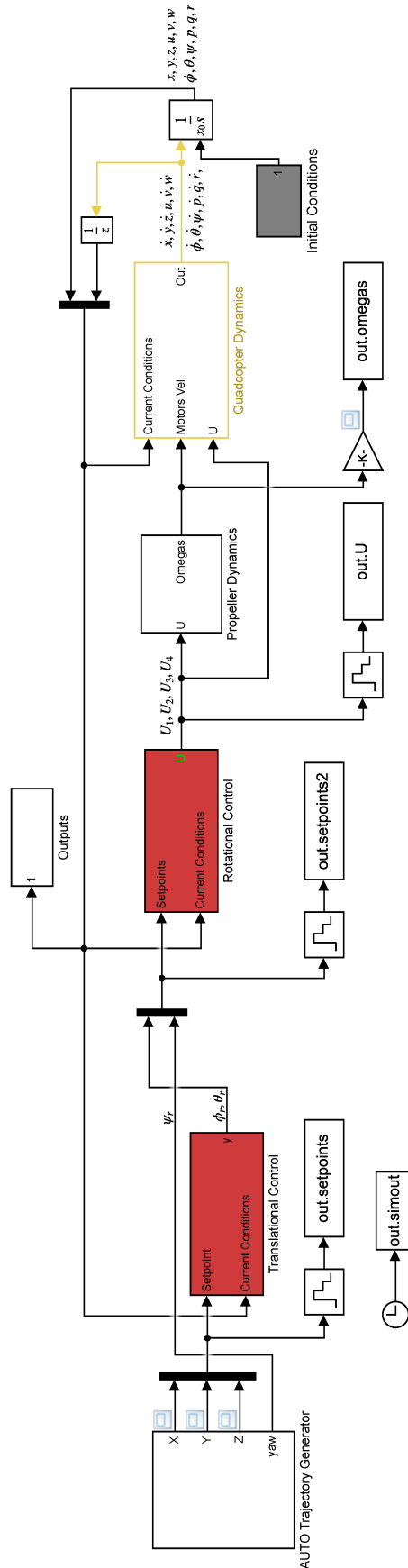
This appendix presents the diagrams developed in Simulink for simulating the dynamic model, the Ardupilot controllers, and the controller proposed in this work.

Figure 39 – Simulink diagram of the LQI attitude control



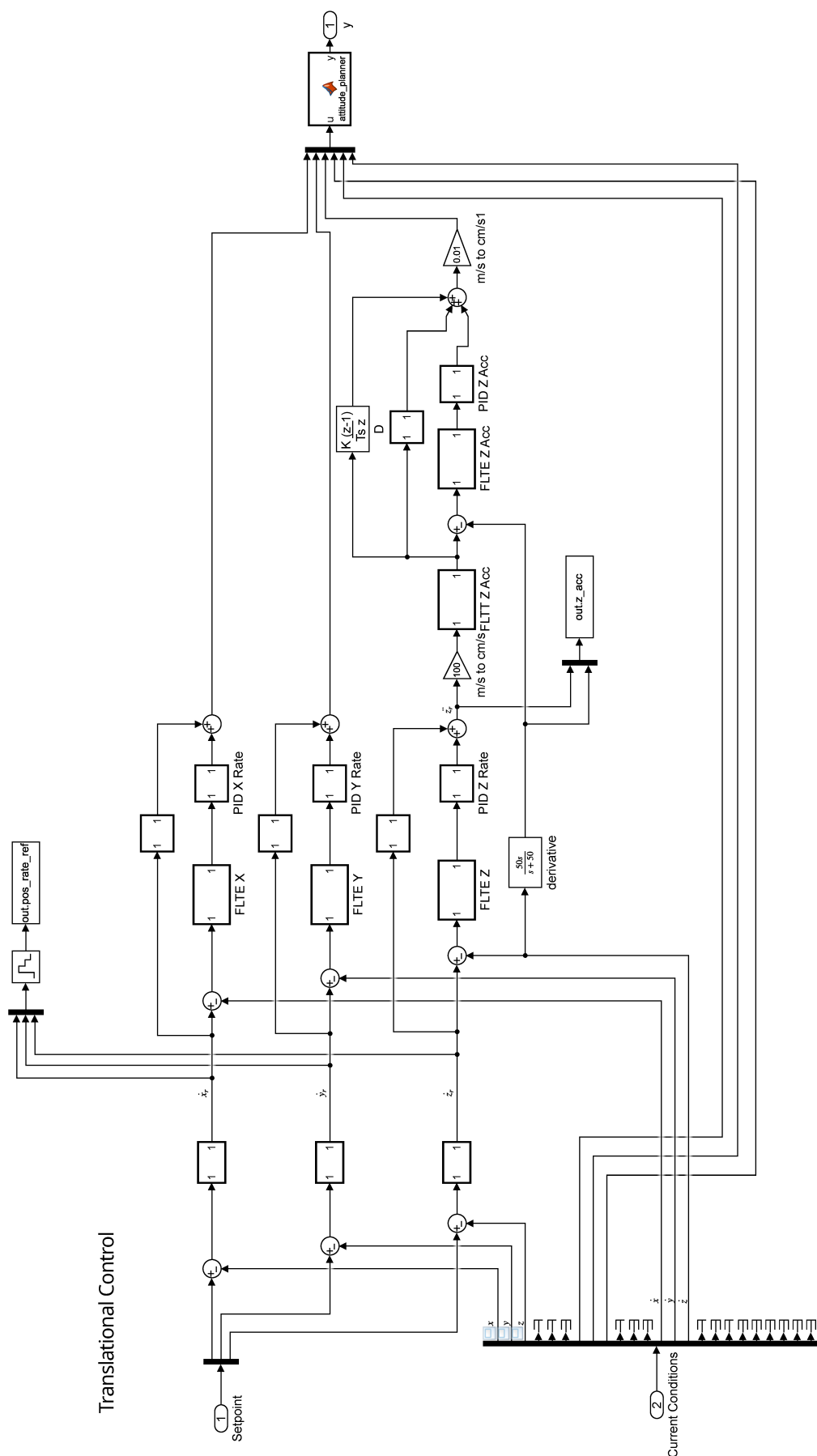
Source: The author (2026).

Figure 40 – Simulink diagram of the general cascade quadcopter control and dynamics.



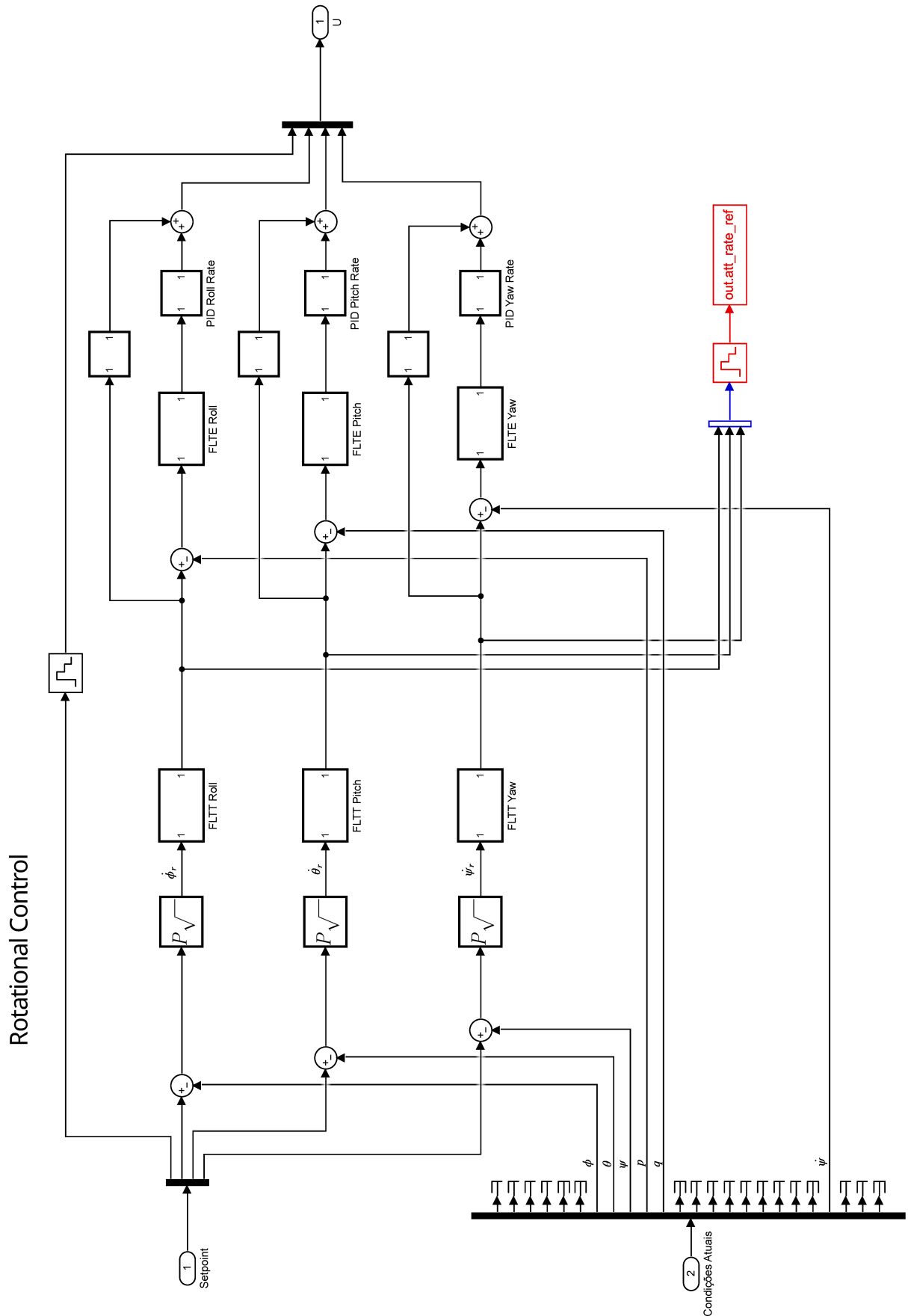
Source: The author (2026).

Figure 41 – Simulink diagram of the Ardupilot’s position controllers.



Source: The author (2026).

Figure 42 – Simulink diagram of the Ardupilot's attitude control.



Source: The author (2026).



Attitude Controller Gain Matrices

This appendix presents the gain matrices of the gain-scheduled LQI attitude controller.

$$K|_{r=-2,V=0} = \begin{bmatrix} 6.9672 & -0.5067 & 0 & 0.5069 & -0.0002 & 0 & -24.3319 & -1.7624 & 0 \\ 0.5289 & 7.0169 & 0 & -0.0003 & 0.5192 & 0 & 1.7641 & -24.3734 & 0 \\ 0 & 0 & 6.8046 & 0 & 0 & 1.0368 & 0 & 0 & -8.3791 \end{bmatrix}$$

$$K|_{r=-2,V=1.5} = \begin{bmatrix} 1.2141 & -2.8786 & 3.5205 & 0.1644 & -0.0827 & 1.0146 & -5.9580 & 6.7018 & -3.6814 \\ -0.6985 & 6.4947 & 1.1380 & -0.0780 & 0.4936 & 0.1006 & 5.8273 & -22.3290 & -1.2830 \\ -10.1415 & -2.4351 & 1.1867 & -0.4729 & -0.1005 & 0.2683 & 38.4405 & 12.6624 & -1.3293 \end{bmatrix}$$

$$K|_{r=-2,V=3} = \begin{bmatrix} 0.7445 & -5.4477 & 3.5203 & 0.1401 & -0.1251 & 1.6352 & -8.7695 & 10.5949 & -3.3126 \\ -1.1663 & 5.4898 & 1.9042 & -0.1178 & 0.4466 & 0.2492 & 6.7324 & -19.6103 & -2.0682 \\ -7.2079 & -2.9101 & 0.9444 & -0.3203 & -0.1122 & 0.2998 & 26.2603 & 12.6584 & -0.9891 \end{bmatrix}$$

$$K|_{r=-2,V=4.5} = \begin{bmatrix} 1.4019 & -7.8596 & 2.6749 & 0.2048 & -0.1815 & 2.0334 & -15.7744 & 12.8412 & -2.3038 \\ -1.9510 & 2.7784 & 2.9696 & -0.1703 & 0.3415 & 0.8799 & 5.7041 & -14.0231 & -3.0758 \\ -5.1587 & -4.1345 & 1.0081 & -0.2057 & -0.1555 & 0.4380 & 16.5096 & 14.6645 & -1.0225 \end{bmatrix}$$

$$K|_{r=-2,V=6} = \begin{bmatrix} 3.1084 & -8.3060 & 1.7899 & 0.3234 & -0.1864 & 1.8797 & -19.8908 & 11.5956 & -1.4560 \\ -2.2196 & 0.0500 & 3.5768 & -0.1749 & 0.2176 & 1.5991 & 3.3310 & -10.5911 & -3.5194 \\ -3.9291 & -5.0670 & 1.0580 & -0.1330 & -0.1937 & 0.5091 & 11.7651 & 16.0893 & -1.0604 \end{bmatrix}$$

$$K|_{r=0,V=0} = \begin{bmatrix} 6.9961 & 0 & 0 & 0.5076 & 0 & 0 & -24.3940 & 0 & 0 \\ 0 & 7.0483 & 0 & 0 & 0.5200 & 0 & 0 & -24.4353 & 0 \\ 0 & 0 & 6.8046 & 0 & 0 & 1.0368 & 0 & 0 & -8.3791 \end{bmatrix}$$

$$K|_{r=0,V=1.5} = \begin{bmatrix} 6.9927 & 0 & -0.1055 & 0.5075 & 0 & 0.0038 & -24.3813 & 0 & 0.1239 \\ 0 & 7.0483 & 0 & 0 & 0.5200 & 0 & 0 & -24.4353 & 0 \\ 0.4150 & 0 & 6.8002 & 0.0116 & 0 & 1.0371 & -1.6896 & 0 & -8.3748 \end{bmatrix}$$

$$K|_{r=0,V=3} = \begin{bmatrix} 6.9826 & 0 & -0.2109 & 0.5072 & 0 & 0.0075 & -24.3435 & 0 & 0.2476 \\ 0 & 7.0483 & 0 & 0 & 0.5200 & 0 & 0 & -24.4353 & 0 \\ 0.8293 & 0 & 6.7871 & 0.0233 & 0 & 1.0383 & -3.3772 & 0 & -8.3617 \end{bmatrix}$$

$$K|_{r=0,V=4.5} = \begin{bmatrix} 6.9658 & 0 & -0.3159 & 0.5066 & 0 & 0.0112 & -24.2804 & 0 & 0.3711 \\ 0 & 7.0483 & 0 & 0 & 0.5200 & 0 & 0 & -24.4353 & 0 \\ 1.2423 & 0 & 6.7653 & 0.0348 & 0 & 1.0401 & -5.0607 & 0 & -8.3400 \end{bmatrix}$$

$$K|_{r=0,V=6} = \begin{bmatrix} 6.9424 & 0 & -0.4204 & 0.5058 & 0 & 0.0150 & -24.1922 & 0 & 0.4941 \\ 0 & 7.0483 & 0 & 0 & 0.5200 & 0 & 0 & -24.4353 & 0 \\ 1.6535 & 0 & 6.7348 & 0.0463 & 0 & 1.0428 & -6.7382 & 0 & -8.3096 \end{bmatrix}$$

$$K|_{r=2,V=0} = \begin{bmatrix} 6.9672 & 0.5067 & 0 & 0.5069 & 0.0002 & 0 & -24.3319 & 1.7624 & 0 \\ -0.5289 & 7.0169 & 0 & 0.0003e & 0.5192 & 0 & -1.7641 & -24.3734 & 0 \\ 0 & 0 & 6.8046 & 0 & 0 & 1.0368 & 0 & 0 & -8.3791 \end{bmatrix}$$

$$K|_{r=2,V=1.5} = \begin{bmatrix} 1.2131 & 2.8766 & 3.5209 & 0.1643 & 0.0826 & 1.0147 & -5.9539 & -6.6952 & -3.6818 \\ 0.6964 & 6.4961 & -1.1366 & 0.0779 & 0.4936 & -0.1002 & -5.8211 & -22.3338 & 1.2815 \\ -10.1422 & 2.4313 & 1.1868 & -0.4730 & 0.1004 & 0.2683 & 38.4448 & -12.6480 & -1.3295 \end{bmatrix}$$

$$K|_{r=2,V=3} = \begin{bmatrix} 0.7055 & 5.4046 & 3.5382 & 0.1373 & 0.1215 & 1.6393 & -8.6512 & -10.4678 & -3.3312 \\ 1.1120 & 5.5557 & -1.8679 & 0.1145 & 0.4492 & -0.2306 & -6.6286 & -19.7763 & 2.0358 \\ -7.2232 & 2.8472 & 0.9489 & -0.3212 & 0.1084 & 0.3001 & 26.3341 & -12.4235 & -0.9961 \end{bmatrix}$$

$$K|_{r=2,V=4.5} = \begin{bmatrix} 0.9236 & 7.8487 & 2.7775 & 0.1738 & 0.1664 & 2.1458 & -15.6007 & -12.5699 & -2.3873 \\ 1.6842 & 3.2826 & -2.8094 & 0.1560 & 0.3674 & -0.7358 & -5.4590 & -14.8902 & 2.9738 \\ -5.1919 & 3.9173 & 1.0425 & -0.2094 & 0.1347 & 0.4579 & 16.1939 & -13.7160 & -1.0741 \end{bmatrix}$$

$$K|_{r=2,V=6} = \begin{bmatrix} 2.4671 & 8.5433 & 1.9246 & 0.2853 & 0.1915 & 2.0224 & -19.4958 & -11.9863 & -1.5448 \\ 2.2271 & 0.5243 & -3.4689 & 0.1795 & 0.2482 & -1.4669 & -3.3496 & -11.0834 & 3.4695 \\ -4.1021 & 4.8354 & 1.0426 & -0.1426 & 0.1782 & 0.5197 & 11.8989 & -15.2357 & -1.0485 \end{bmatrix}$$