



Politecnico
di Torino

ScuDo

Scuola di Dottorato - Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Electronics and Telecommunications Engineering
(36th cycle)

Towards Trustworthy Data-driven Modeling and Control of Unmanned Aerial Vehicles

By

Weibin Gu

Supervisor:

Prof. Alessandro Rizzo

Doctoral Examination Committee:

Prof. Marco Cognetti, LAAS-CNRS Toulouse and Université Toulouse III, France

Prof. Carlo Novara, Politecnico di Torino, Italy

Prof. Maurizio Porfiri, New York University Tandon School of Engineering, U.S.A.

Prof. Simona Sacone, University of Genova, Italy

Prof. Kimon P. Valavanis, University of Denver, U.S.A.

Politecnico di Torino

2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Weibin Gu

2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

I would like to dedicate this thesis to my loving parents and wife, whose unwavering support and encouragement have been my pillars along this transformative journey.

Acknowledgements

First and foremost, words cannot express my deepest gratitude to Prof. Alessandro Rizzo, my dear supervisor, for his unwavering patience and support along my Ph.D. journey, which were pivotal to me, especially during the first two years amidst the COVID-19 pandemic. His professionalism, mentorship, and attention to detail positively impacted both my academic and personal growth. My profound appreciation is also extended to the defense committee for providing valuable insights that enriched my dissertation. Undoubtedly, this endeavor would not have been possible without the generous support from MOST (The National Centre for Sustainable Mobility), FAIR (Future Artificial Intelligence Research), and Amazon Science, who offered essential funding for my research.

Special appreciation is reserved for Prof. Stefano Primatesta, who unceasingly provided useful comments and proofread my research papers. Thanks also go to Prof. Marina Indri, Prof. Michele Taragna, and Prof. Carlo Novara, for their instructive feedback on the yearly EECE Poster Days, and to Prof. Stefano Grivet-Talocia, the Ph.D. coordinator of the EECE Department, for his daily support throughout my doctoral program. I am also grateful to Prof. Kimon P. Valavanis from University of Denver, for his savvy advice at the onset of this journey.

Lastly, I would be remiss in not mentioning my family – my parents Qin Shi and Zhihao Gu, my wife Ziyang Wang, and my grandparents – for their unfaltering belief in me, which, along this transformative venture, served as a constant source of strength and motivation. My sincere gratitude also extends to my friends and colleagues for their emotional support, as well as numerous others, whose names I cannot list here, for their encouragement and assistance.

In some sense, this dissertation goes beyond just being the work of a sole author; it is a dedication to pushing scientific boundaries shaped by collective wisdom. Since the commencement of this journey, I feel truly fortunate to have

worked with and been supported by numerous like-minded individuals with brilliant minds and supportive organizations in carrying out this research. Thank you all again for being a part of my memorable journey!

Funding Acknowledgement: The research activities of this thesis have been partly carried out within the MOST – Sustainable Mobility National Research Center and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.4 – D.D. 1033 17/06/2022, CN00000023) and within the FAIR - Future Artificial Intelligence Research and received funding from the European Union Next-GenerationEU (PIANO NAZIONALE DI RIPRESA E RESILIENZA (PNRR) – MISSIONE 4 COMPONENTE 2, INVESTIMENTO 1.3 – D.D. 1555 11/10/2022, PE00000013). The views and opinions expressed in this work are solely those of the authors, and neither the European Union nor the European Commission can be held responsible for them. Additionally, Weibin Gu acknowledges partial support from the 2021 Amazon Research Award titled “Physics-Informed Machine Learning for Trustworthy Control of Autonomous Robots.”

Abstract

Unmanned Aerial Vehicles (UAVs) have gained significant attention and utility in various aspects of daily life, ranging from entertainment, like aerial photography, to meeting industrial demands such as delivery of goods and infrastructure inspection. As tasks and environments become more complex, advanced control algorithms are needed to ensure safe operations. While model-based control techniques have shown effectiveness in controller design, their performance heavily relies on accurate mathematical models, posing challenges in scenarios with uncertainties and disturbances. In response, data-driven methods, particularly learning-based approaches, have shown promise in accurate modeling due to their powerful approximation capabilities. End-to-end learning-based solutions have also emerged, mitigating the requirement for system knowledge in control synthesis. However, learning-based approaches, especially deep learning, are often regarded as black-box models, lacking interpretability and raising trust issues about their deployment in safety-critical systems like quadrotors. Moreover, many current studies on learning-based approaches are offline, trained before task execution, which introduces potential risks in real-world execution and compromises generalization to unseen scenarios.

This thesis aims to advance trustworthy data-driven modeling and control of quadrotors, focusing on two main parts: modeling and control design. In the modeling part, we propose a novel machine learning paradigm called Physics-informed Machine Learning (PIML) for quadrotor dynamical modeling. PIML integrates domain knowledge and empirical data to enhance the trustworthiness of the model, outperforming black-box and conventional mathematical models in terms of both modeling error and physical consistency. Our model also demonstrates improved learning capability with smaller data sets and provides interpretability through post-hoc visualization. In the control design part, we address control problems of a quadrotor subject to parametric and non-parametric

uncertainties. For parametric uncertainties, we develop a novel adaptive geometric controller, of which the synthesis is based on rotation matrix rather than Euler angles or quaternions, thereby avoiding issues such as gimbal lock and unwinding phenomena. Consequently, this controller enables aggressive maneuvers like 360° flips and elliptical helix trajectory tracking, even in the presence of uncertain mass and inertia matrix. To handle non-parametric uncertainties like wind gusts, we introduce a novel learning-based controller featuring online learning capability through the reservoir computing paradigm. We investigate the interpretability of our learning-based model through post-hoc analysis on model dynamics and parameters, providing valuable insights for understanding the model's behavior. Comparisons with offline solutions in the literature demonstrate the superior generalizability and tracking performance of our learning-based controller in facing unseen scenarios, thanks to its online learning ability. This, along with insights from post-hoc analysis, instills trust in our data-driven solution.

We anticipate a growing trend of data-driven solutions for robotics in the near future, driven by both market demands and technological advancements like Large Language Models (LLMs). Hence, infusing trust into these data-driven solutions is crucial, particularly for applications or products with potential safety implications. Although the research in trustworthy data-driven methods is still in its early stages, we acknowledge the collaborative efforts from various fields, such as robotics and machine learning, toward the common goal. These collective endeavors will ensure the successful deployment of intelligent systems in the real world, improving the lives of individuals and benefiting society as a whole. This thesis aims to contribute to this process by providing valuable insights and enlightening researchers in the field, serving as a small step in the larger journey towards advancing trustworthy data-driven modeling and control of quadrotors.

Contents

| | |
|--|-----------|
| List of Figures | xi |
| List of Tables | xv |
| 1 Background and Motivation | 1 |
| 1.1 Overview of Unmanned Aerial Vehicles | 1 |
| 1.2 The Synergy of Modeling and Control | 4 |
| 1.2.1 The Past: Model-based Methods | 5 |
| 1.2.2 The Present: Data-driven Methods | 7 |
| 1.2.3 The Future: Forging a Path to Trust | 9 |
| 1.3 Building Trust in Algorithms: Trustworthy Machine Learning | 10 |
| 1.3.1 Transparent Models | 11 |
| 1.3.2 Post-hoc Interpretability Techniques | 13 |
| 1.4 Thesis Outline and Contributions | 13 |
| 2 Preliminaries | 16 |
| 2.1 Notation | 16 |
| 2.2 Mathematical Model of Quadrotors | 17 |
| 2.3 Exponential and Logarithm Map | 18 |
| 2.4 Artificial Neural Networks | 20 |
| 2.4.1 Historical Perspective | 20 |
| 2.4.2 Mathematical Model of Artificial Neurons | 21 |

| | | |
|----------|---|-----------|
| 2.4.3 | Classifications of Neural Networks | 21 |
| 2.4.4 | Training Methods | 23 |
| 3 | Physics-informed Neural Modeling | 26 |
| 3.1 | Introduction | 26 |
| 3.2 | Related Work | 28 |
| 3.3 | Physics-informed Neural Network | 30 |
| 3.3.1 | Network Structure | 31 |
| 3.3.2 | Incorporation of Physics as Learning Bias | 32 |
| 3.3.3 | Cyclical Annealing Scheduler | 36 |
| 3.3.4 | Post-hoc Model Interpretability Visualization | 37 |
| 3.4 | Results and Discussion | 38 |
| 3.4.1 | Simulator | 38 |
| 3.4.2 | Network Training | 39 |
| 3.4.3 | Model Comparison and Ablation Study | 40 |
| 3.4.4 | Computational Complexity | 47 |
| 3.5 | Conclusions | 48 |
| 4 | Robust Adaptive Controller Design for Parametric Uncertainties | 50 |
| 4.1 | Introduction | 50 |
| 4.2 | Related Work | 52 |
| 4.3 | Problem Formulation | 55 |
| 4.4 | Control Synthesis for Position Tracking | 56 |
| 4.5 | Control Synthesis for Attitude Tracking | 59 |
| 4.6 | Results and Discussion | 66 |
| 4.6.1 | Simulator | 66 |
| 4.6.2 | Maneuver #1: Doing a 360° Flip | 67 |
| 4.6.3 | Maneuver #2: Tracking an Elliptical Helix Trajectory | 68 |

| | | |
|----------|--|------------|
| 4.6.4 | Maneuver #3: Tracking a Figure-8 Trajectory | 70 |
| 4.7 | Conclusions | 71 |
| 5 | Learning-based Controller Design for Non-Parametric Uncertainties | 73 |
| 5.1 | Introduction | 73 |
| 5.2 | Problem Formulation | 75 |
| 5.3 | Reservoir Computing Paradigm | 76 |
| 5.3.1 | Mathematical Model of Echo State Networks | 76 |
| 5.3.2 | Training Methods | 77 |
| 5.3.3 | Echo State Property | 77 |
| 5.4 | Learning-based Tracking Controller | 78 |
| 5.4.1 | Online Learning Module for Residual Dynamics | 78 |
| 5.4.2 | Tracking Control Laws | 80 |
| 5.5 | Post-hoc Analysis of Model Dynamics and Interpretability | 83 |
| 5.6 | Results and Discussion | 85 |
| 5.6.1 | Simulator | 86 |
| 5.6.2 | Network Selection and Training | 86 |
| 5.6.3 | Flight Control with Online Learning | 87 |
| 5.6.4 | Comparison with an Offline Learning-based Controller | 90 |
| 5.7 | Conclusions | 93 |
| 6 | Concluding Remarks and Future Work | 95 |
| | References | 98 |
| | Appendix A Multicopter Ground Effect Plugin | 108 |

List of Figures

| | | |
|-----|---|----|
| 1.1 | Hardware components of UAVs. | 2 |
| 1.2 | Software components of UAVs. | 2 |
| 1.3 | Classification of UAVs: Features and limitations. | 3 |
| 1.4 | Taxonomy of flight control methodologies. | 6 |
| 1.5 | Evolution of UAV control: Current trends and characteristics. | 10 |
| 1.6 | Explainable AI techniques. | 11 |
| 1.7 | Fundamental principles of PIML. | 12 |
| 1.8 | Post-hoc visualization workflow for a trained model. | 14 |
| 2.1 | Evolution of ANNs: Timeline and milestones. | 20 |
| 2.2 | Simplified neuron model and typical architecture of FNNs. | 21 |
| 3.1 | Main concept of PINN. | 31 |
| 3.2 | Correlation within training data series. (a), (b), (e), (f), (i), (j): The derivative of angular rate and the corresponding PWM signals. (c), (g), (k): Coherence plots for roll, pitch, and yaw motion, with PCCs indicated in gray boxes. (d), (h), (i): CCE plots for roll, pitch, and yaw motion, depicting data samples (blue dots) and 3σ CCEs (orange ellipses). | 34 |
| 3.3 | Illustration of cyclical annealing scheduler λ_{LM} in Eq. (3.4). | 37 |

| | | |
|-----|--|----|
| 3.4 | Data collection in the visual and physical simulator. (a) Manual flight of a quadrotor in the simulator. (b) User interface of QGround-Control. (c) Configuration of the employed quadrotor overlaid with a body-fixed reference frame. (d) Flight trajectory with a colorbar indicating the magnitude of linear velocity. | 39 |
| 3.5 | Simulated flight data with ground effect. (Top) Parametric model of quadrotor ground effect compared with simulator-collected data samples [†] . (Bottom) Training flight data, showing ground effect force along body z -axis ($F_{g,z}$) and quadrotor altitude (Z), with shaded areas indicating data partitioning (training/validation/test). | 41 |
| 3.6 | Simulated flight with periodic wind. The top three plots depict the composition of aerodynamic forces, including drag and wind. The bottom plot illustrates the profile of periodic wind in the forward direction with a speed of $v_{\text{wind}} = 2.5 \sin(\frac{\pi t}{5}) + 2.5$, overlaid with shaded areas indicating data partitioning (training/validation/test). | 44 |
| 3.7 | Test error comparison on \mathcal{D}_1 data set. The chart shows test errors for models M2 to M10 [†] . Light red bars represent models (M2-M4) trained on 60% of data without BN. Green bars represent models (M5-M6) trained on 60% of data with BN. Yellow and orange bars show models (M7-M10) with BN trained on 20% of the data and evaluated on 40% and 80% of the data, respectively. Error bars denote standard deviation. | 45 |
| 3.8 | Comparison of CCE between vanilla DNN and PINN. Targets and predictions are represented by dots and plus signs, respectively. Shaded ellipses illustrate 3σ CCEs, with dashed lines highlighting the slope. | 46 |
| 3.9 | Test error comparison on \mathcal{D}_2 and \mathcal{D}_3 data sets. All results are averaged over multiple seeds, with error bars representing the standard deviation. | 48 |
| 4.1 | Control scheme of the proposed robust adaptive geometric tracking controller on SE(3). | 55 |
| 4.2 | Numerical analysis of the bounds of $\ \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top}\ _{\text{F}}$ over $\mathcal{S} = \{\phi \mathbf{a} : \phi < \pi, \mathbf{a} \in \mathbb{S}^2\}$ | 65 |

| | | |
|-----|---|----|
| 4.3 | 360° flip maneuver. (a) Flip illustration in X-Z plane. (b) Quadrotor position \mathbf{p} . (c) Euler angles ϕ, θ, ψ . (d) Attitude error $\ \tilde{\mathbf{r}}\ _2$ and control inputs $f, \boldsymbol{\tau}$ | 67 |
| 4.4 | Elliptical helix trajectory tracking. (a) 3-D visualization [†] . (b) Comparisons under nominal and uncertain scenarios. (c) Mass estimation \bar{m} and inertia matrix estimation error $\ \tilde{\mathbf{J}}\ _2$. (d) Attitude error $\ \tilde{\mathbf{r}}\ _2$ and control inputs $f, \boldsymbol{\tau}$ | 69 |
| 4.5 | Figure-8 trajectory tracking. (a) 3-D visualization with a colorbar indicating the absolute value of the magnitude of linear velocity \mathbf{v} . (b) Quadrotor position \mathbf{p} . (c) Linear velocity \mathbf{v} and Euler angles ϕ, θ, ψ . (d) Motor speeds. | 70 |
| 5.1 | Control scheme of the proposed learning-based tracking controller. The online learning module for non-parametric uncertainty estimation (framed by an orange dashed box) takes as inputs the online measurements of quadrotor states $\mathbf{x} = [\mathbf{p}^\top, \mathbf{v}^\top, \boldsymbol{\eta}^\top, \boldsymbol{\omega}^\top]^\top$ and control inputs $\mathbf{u} = [\mathbf{f}_u^\top, \boldsymbol{\tau}_u^\top]^\top$, and generates uncertainty estimates $\hat{\Delta} = [\hat{\mathbf{f}}_a^\top, \hat{\boldsymbol{\tau}}_a^\top]^\top$ | 78 |
| 5.2 | Structure of online residual learning module: Deep ESN and data buffer. The data buffer of length N_b stores the feature and label data from flight history in a column-wise fashion (shown by green blocks), which are subsequently used for online training of readout weights of ESN (depicted in orange arrows). The ESN model adopts a hierarchical architecture for reservoir layers (shaded in blue) between which inter-layer connections exist (depicted in yellow arrows). . . | 81 |
| 5.3 | Asymptotic stable behavior of reservoir layers ensured by ESP reflects the goodness of reservoir dynamics. | 84 |
| 5.4 | Online learning performance over 50 simulation campaigns. (a) Predictions of quadrotor ground effect. (b) Predictions of periodic wind. (c) Training time of ESN for learning ground effect and periodic wind. | 88 |

| | | |
|-----|---|-----|
| 5.5 | Comparison of control behavior between our proposed learning-based controller and a nominal controller. (a) Tracking performance for executing a figure-8 maneuver (viewed in 2D). (b) Control input. (c) Position and velocity. | 89 |
| 5.6 | Comparison of control behavior between our proposed learning-based controller (ESN) and an offline learning-based controller (FNN). (a) Distribution shift in Euler angles (i.e., network inputs). (b) Distribution shift in network predictions and the learning performance of FNN. (c) Tracking performance for executing a figure-8 maneuver. (d) Position and velocity. | 92 |
| 5.7 | Post-hoc visualization using t-SNE. (a) Clustering of readout weight matrix. (b)-(f) Clustering of reservoir states. | 93 |
| A.1 | Architecture and components of our custom simulation environment. (a) Microsoft AirSim and UE4, (b) PX4 firmware, (c) Ground control station, QGroundControl, and (d) Overall architecture of the simulation environment enhanced with our ground effect implementation. | 111 |
| A.2 | Comparison of two ground effect models: Cheeseman-Bennett model vs. parametric model. | 115 |

List of Tables

| | | |
|-----|---|----|
| 1.1 | Comparison of advantages and disadvantages: Linear controllers (top) vs. Nonlinear controllers (bottom). | 8 |
| 2.1 | Classification of ANNs. | 23 |
| 3.1 | Quadrotor specifications. | 40 |
| 3.2 | Models for comparison and ablation studies. We analyzed: (i) model structure (2 nd – 4 th column), (ii) application of Batch Normalization (BN) (5 th – 6 th column), (iii) regularization hyperparameter settings (7 th – 8 th column), and (iv) training data set (9 th – 12 th column). Bullets (•) symbolizes the highlighted features present within the model. | 42 |
| 3.3 | Comparison results of dynamical modeling. Δy denotes the absolute value of prediction error between prediction \hat{y} and label y . Δm_i denotes the absolute difference of slope (or rotation angle) between prediction and label CCEs, where $i = \phi, \theta, \psi$ denotes the three degrees of rotation. $(\bar{\cdot})$ denotes averaging over multiple seeds and $\sigma(\cdot)$ denotes the corresponding standard deviation. Superscript † and ‡ denote linearized mathematical model (3.2) and vanilla DNN, respectively. The results of comparison and ablation studies are reported in groups, which are separated by horizontal lines, with the best performance indicated in bold. | 43 |

| | | |
|-----|--|----|
| 4.1 | Comparative analysis of selected literature on geometric control with applications to quadrotors. We compared the error function for rotational configuration (2 nd column), control methods for handling uncertainties (3 rd column), and obtained results (4 th column; <i>Num</i> : numerical, <i>Sim</i> : simulation, <i>Exp</i> : experimental). | 53 |
| 5.1 | ESN parameters. | 86 |
| 5.2 | Tracking errors of three different controllers over 20 simulation campaigns. Controllers include: (i) A nominal controller, (ii) a learning-based controller with offline-trained FNN, and (iii) our proposed learning-based controller with online residual learning using ESN. | 91 |

Chapter 1

Background and Motivation

1.1 Overview of Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAVs), colloquially referred to as drones, have emerged as versatile platforms with considerable applicability across diverse domains, owing to their capacity for autonomous or remote task execution. Typically, they represent a wide range of aerial vehicles operated without a human pilot onboard, controlled either autonomously by onboard computers or remotely by a human operator.

From a system-level viewpoint, UAVs exhibit a sophisticated interplay of hardware and software components. Hardware components include *airframe structures*, *propulsion mechanisms*, *avionic systems*, *payload modules*, *power units*, and *landing gears*; see, e.g., Fig. 1.1 for a detailed breakdown. The software constituents of UAVs comprise critical functionalities necessary for their autonomous operation. This includes the *flight control system*, responsible for stabilizing and maneuvering the UAV, the *navigation system*, which utilizes sensor measurements to determine its precise location, the *guidance system*, governing its trajectory towards predefined waypoints or objectives, and the *mission planning system*, orchestrating complex tasks and coordinating multiple UAVs in collaborative missions. Together, these software components form the backbone of UAV autonomy, enabling the execution of diverse missions with precision and efficiency. An overview of the software components of UAVs is illustrated in Fig. 1.2. It is worth noting that operators retain the authority to intervene and control the UAV using







| Component | Descriptions | Examples |
|-------------------|---|--|
| Airframe | The structure of the UAV, including wings, fuselage, and tail assembly. |  |
| Propulsion system | Engines or motors and propellers used for thrust generation. |  Motor Propeller |
| Avionics | Electronic systems for navigation, communication, and control, including sensors, microcontrollers, and radios. |  Arduino Radio GPS WiFi |
| Payload | Equipment carried by the UAV for specific missions such as sensors or cargo. |  Camera LiDAR |
| Power system | Batteries or fuel cells that provide energy to the propulsion and avionics systems. |  Battery |
| Landing gear | Mechanisms for takeoff and landing. |  |

Image source:

- <https://www.sparkfun.com/>
- <https://www.yangdaonline.com/>
- <https://www.dji.com/cn>
- https://www.amazon.com/ref=nav_logo

Fig. 1.1 Hardware components of UAVs.

remote controllers or ground control stations (referred to as “ground control” in Fig. 1.2) during tasks, serving as common tools for monitoring and supervision.

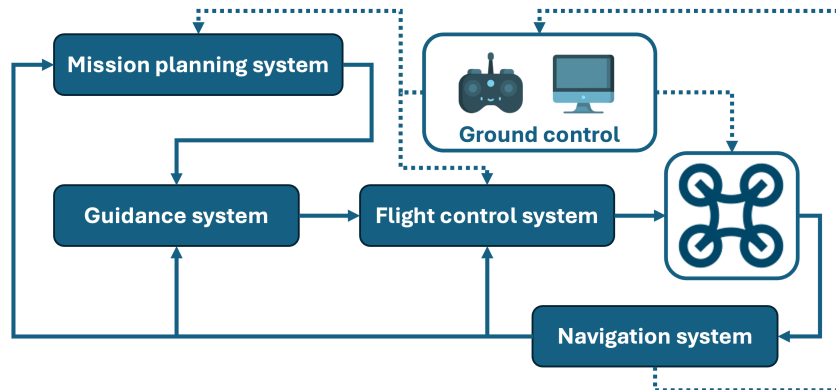


Fig. 1.2 Software components of UAVs.

From a structural and propulsion standpoint, UAVs can be systematically classified into distinct categories, which include *fixed-wing*, *rotary-wing* (encompassing both single-rotor or helicopter and multirotor configurations), *tilt-rotor*, and *tailsitter* UAVs [1]. In addition, there are morphing or bio-inspired UAVs (see, e.g., [2]) capable of dynamically altering their external configuration to suit varying mission environments during flight. However, these are not addressed here due to scope limitations. A concise overview of the features and operational constraints of various types of UAVs is presented in Fig. 1.3.






| Type | Advantages | Disadvantages | Examples |
|---------------------|--|--|---|
| Helicopter | <ul style="list-style-type: none"> ✓ VTOL capability ✓ Ability to hover | <ul style="list-style-type: none"> × High operational costs × Limited speed and range × Complex mechanical systems |  (a) |
| Fixed-wing aircraft | <ul style="list-style-type: none"> ✓ High speed and long range ✓ Low operational costs and fuel consumption | <ul style="list-style-type: none"> × Require runways for takeoff and landing × Limited maneuverability × Inability to hover |  (b) |
| Multicopter | <ul style="list-style-type: none"> ✓ VTOL ✓ Ability to hover ✓ Compact size and portability | <ul style="list-style-type: none"> × Short flight times × Limited speed and range |  (c) |
| Tilt-rotor aircraft | <ul style="list-style-type: none"> ✓ Combine VTOL capability with high speed and long range ✓ Versatile in operation | <ul style="list-style-type: none"> × Complex mechanical systems × Higher manufacturing and operational costs |  (d) |
| Tailsitter | <ul style="list-style-type: none"> ✓ Combine VTOL capability ✓ Compact design ✓ Versatile in operation | <ul style="list-style-type: none"> × Complex flight dynamics × Limited payload |  (e) |

Image source:

- (a) <https://www.unmannedsystemstechnology.com/company/steadicopter/>
(b) <https://www.unmannedsystemstechnology.com/company/fly-dragon-drone-tech/fdg30-vtol-fixed-wing-uav/>
(c) DJI Matrice 350 RTK
(d) <https://www.militarydrones.org.cn/ch-10-tiltrotor-drone-china-price-manufacturer-procurement-portal-p00148p1.html>
(e) <https://spectrum.ieee.org/tu-delft-tailsitter>

Fig. 1.3 Classification of UAVs: Features and limitations.

In this study, our focus lies on *quadrotors* as the system under control. Quadrotors are multicopter UAVs characterized by a four-rotor configuration, hence the name “quad”rotor. Over the past years, quadrotors have garnered substantial attention and demonstrated significant utility in various aspects of our daily lives, spanning from entertainment, like aerial photography, to meeting industrial demands such as delivery of goods and infrastructure inspection; see, e.g., [3–9]. This is attributed by and large to their high mobility, Vertical Takeoff and Landing (VTOL) capability, as well as low maintenance cost. Nonetheless, the design of controllers for quadrotors toward either trajectory tracking or attitude stabilization is non-trivial due to the inherent nonlinearities and strong coupling properties in quadrotor dynamics, let alone uncertainties and disturbances, which are ubiquitous in real-world flight operations [10]. The challenge further escalates when the quadrotor is assigned to perform aggressive maneuvers, particularly for specific tasks such as aerial acrobatics or rapid changes in direction [11, 12].

In summary, the rapid proliferation of UAVs in the 21st century has given rise to an unparalleled demand for control algorithms that not only ensure satisfactory tracking performance but also facilitate *safe* and *trustworthy* operations. This necessity becomes particularly pronounced in complex and densely populated

environments to enable real-world deployment while minimizing the risk of injury and property damage.

1.2 The Synergy of Modeling and Control

Modeling and control are core areas of UAV research, indispensable for understanding dynamic behavior and devising algorithms for stabilizing and maneuvering these aerial platforms. In the context of controller design, a broad spectrum of methodologies is available, spanning from classical Proportional-Integral-Derivative (PID) control to modern Model-Based Control (MBC) and learning-based control techniques. PID controllers, often deployed as model-free solutions for industrial applications, remain popular due to their simplicity and effectiveness in stabilizing quadrotors, particularly for fundamental tasks such as hovering and altitude control. However, they may struggle with more complex maneuvers or in the presence of disturbances.

To address these issues, modeling techniques come to the rescue, historically evolving from transfer functions derived from First Principle Models (FPMs), such as Ordinary Differential Equations (ODEs) rooted in Newton-Euler equations or Lagrangian mechanics [13], thus laying the groundwork for frequency domain controller design. Nowadays, state-space formulations have gained prevalence for time-domain control synthesis, characterizing the dynamics of the quadrotor in terms of position, velocity, orientation, and angular rate. More sophisticated models may incorporate additional complexities such as aerodynamic effects, motor dynamics, and environmental factors such as wind. Model-based control techniques [14], such as Feedback Linearization (FL), Backstepping Control (BSC), Sliding Mode Control (SMC), and adaptive control, offer distinct advantages in effectively managing the nonlinear dynamics of quadrotors, facilitating precise trajectory tracking and robustness against uncertainties and disturbances. Nonetheless, these techniques often require detailed knowledge of system dynamics and may entail considerable computational resources.

Recent trends in the literature have witnessed a surge in the adoption of machine learning and Artificial Intelligence (AI) methodologies for quadrotor control [10]. Notably, deep Reinforcement Learning (RL) has emerged as a promising approach for learning complex control policies directly from synthetic or real-

world data, empowering quadrotors to execute agile maneuvers and navigate dynamic environments adeptly [15]. Deep Neural Networks (DNNs) hold potential for building control-oriented models in a more effective way within model-based control frameworks [16]. While these learning-based approaches can potentially outperform traditional control methods in challenging scenarios, they typically require large amounts of training data and may lack interpretability compared to model-based control techniques.

Overall speaking, the choice of modeling and control techniques for quadrotors depends on the requirements of the specific application, the available computational resources, and the desired trade-offs between performance, robustness, and simplicity. Synergistic integration of multiple approaches, such as combining model-based models with data-driven methods, holds promise for enhancing the capabilities of quadrotor systems. In the subsequent discussion, we walk through the evolution of modeling and control techniques for UAVs: examining their historical development, current state, and potential future advancements. A categorization of control methodologies for UAV control is illustrated in Fig. 1.4.

1.2.1 The Past: Model-based Methods

MBC techniques have witnessed widespread adoption in recent decades, owing to their foundation in analytical formulations of system dynamics expressed through differential equations, thereby informing the controller design with assured performance. Model-based controllers can be further categorized into two classes: *linear* and *nonlinear*, as depicted in Fig. 1.4.

Linear controllers involve Single-Input Single-Output (SISO) and Multi-Input Multi-Output (MIMO) methods. Examples of SISO methods include PID control, while MIMO methods encompass Linear Quadratic Regulator (LQR), Linear Quadratic Gaussian (LQG), $\mathcal{H}_\infty/\mathcal{H}_2$ control, Gain Scheduling (GS), and Model Predictive Control (MPC), among others. Nonlinear controllers can be divided into linearized and fully nonlinear methods. Linearized methods start with a nonlinear model and proceed with linearization to derive a linear model for control design, including FL, SMC, MPC, adaptive control, and so forth. Fully nonlinear methods, such as BSC, can be applied directly to nonlinear dynamics without the need for linearization.

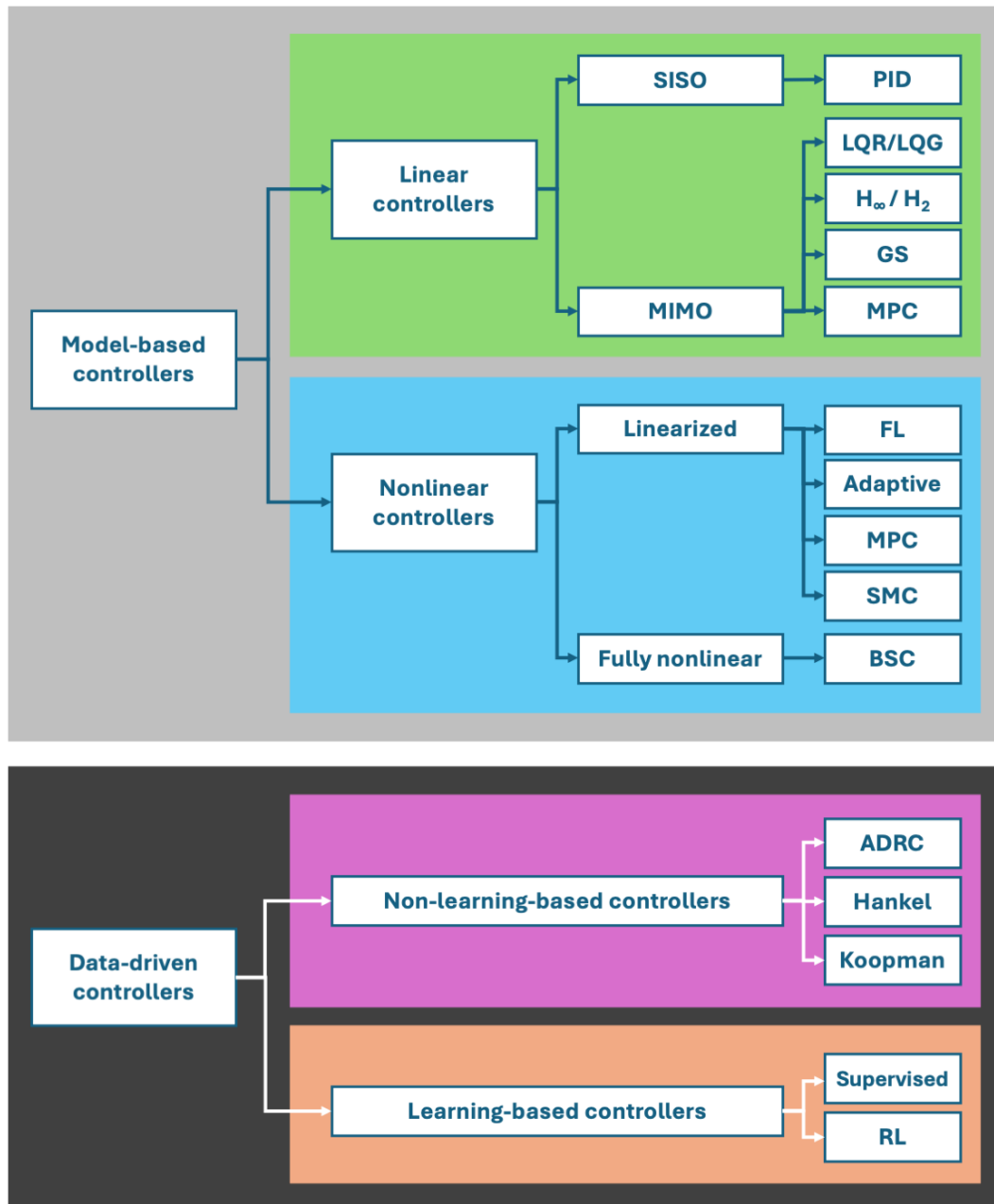


Fig. 1.4 Taxonomy of flight control methodologies.

A notable limitation of all MBC techniques lies in their dependence on the accuracy of the mathematical model describing the system under control [10]. Real-world flight scenarios introduce uncertainties and disturbances such as parametric variations and unmodeled dynamics, which pose challenges to the construction of accurate control-oriented models, ultimately impacting control

performance adversely. A major concern is the risk of system instability, which in the context of UAVs can lead to hazardous deviations from intended trajectories and potential accidents [17].

In response to modeling challenges exacerbated by uncertainties and disturbances, researchers in the control community have dedicated considerable effort to robust and adaptive control theory [18]. For example, adaptive backstepping controllers was developed to accommodate changes in the mass, inertia, and payload properties of the UAV [9, 19]. Additionally, an adaptive sliding backstepping control scheme was proposed to ensure attitude tracking in the presence of unmodeled dynamics [20]. However, many resulting control schemes suffer from mathematical complexity and the design very often leans toward conservative approaches tailored to worst-case scenarios, thereby constraining the ability to guarantee desired performance [18]. Table 1.1 summarizes the advantages and the limitations of linear and nonlinear controllers.

1.2.2 The Present: Data-driven Methods

In contrast to model-based controllers, data-driven controllers aim to learn the complex dynamics of the real system directly from empirical data sets, including uncertainties and disturbances. Broadly speaking, data-driven approaches can be categorized into *learning-based* methods and *non-learning-based* methods. The latter includes methods like Active Disturbance Rejection Control (ADRC) [21] and Hankel-matrix-based identification and control [22]. Despite ADRC's effectiveness and widespread recognition as a robust alternative to PID control, it requires tuning, and a rigorous stability proof seems lacking. While Hankel-matrix-based method exhibits promise for linear systems, its extension to nonlinear systems remains an open research area.

Learning-based methods, on the other hand, harness the powerful approximation capabilities of Artificial Neural Networks (ANNs) [23–25], leveraging recent advancements in big data and computer hardware. Numerous studies employing DNNs to learn various uncertainties have demonstrated the effectiveness of learning-based methods [26–28]. However, a prominent challenge with deep learning lies in its interpretability, primarily due to the black-box nature of DNNs. This may impede the deployment of DNN-based solutions in real-world safety-

| Controller | Advantages | Limitations |
|----------------------|--|---|
| PID | Easy implementation. Guaranteed stability margins for LQR. | (i) Lack of robustness; (ii) Neglect of dynamic coupling. (i) Limited applicability to specific operating conditions; (ii) Dependence on full state feedback for LQR; (iii) Degraded performance of Kalman filter in LQG; (iv) Disregard for output constraints. |
| \mathcal{H}_∞ | Ability to handle parametric uncertainties and unmodeled dynamics. | (i) Conservative design for addressing structural uncertainties; (ii) Prerequisite of comprehensive understanding of the system model. |
| GS | Capability to cover a broad spectrum of flight envelopes and operating conditions. | (i) Unsteady transition during switches; (ii) Complex and cumbersome design processes. |
| Adaptive | Capability to handle parametric uncertainties. | (i) Requirement for complex analysis and the understanding of system dynamics; (ii) Susceptible to parameter drift induced by process noise; (iii) Impractical implementation of large adaptation rates. |
| FL | Capability to handle nonlinearities. | (i) High computational complexity; (ii) Demanding requirement for model accuracy. |
| MPC | (i) Prediction capabilities; (ii) Explicit handling of constraints on control input. | (i) High demand for accuracy in prediction models; (ii) Time-intensive for online optimization procedures. |
| SMC | Insensitive control performance to modeling errors and parametric uncertainties. | (i) Chattering induced by frequent controller switches; (ii) Challenges in stability analysis due to highly nonlinear sliding mode surface. |
| BSC | Well-suited for underactuated systems. | (i) Requires a nonlinear model in lower triangular form (referred to as pure-feedback form); (ii) High computational complexity for computing derivatives of pseudo control inputs. |

Table 1.1 Comparison of advantages and disadvantages: Linear controllers (top) vs. Nonlinear controllers (bottom).

critical systems such as UAVs, as practitioners may lack trust in these models [29]. Another challenging aspect is the generalizability of the trained model to unseen data [10], particularly for offline-trained models, as their performance may degrade when confronted with Out-Of-Distribution (OOD) data samples.

1.2.3 The Future: Forging a Path to Trust

As the UAV industry tackles increasingly challenging tasks in demanding environments (see Fig 1.5), traditional model-based methods, discussed in Section 1.2.1, may fall short. On the other hand, the adoption of state-of-the-art learning-based approaches raises concerns about trustworthiness, particularly in emerging technologies like AI. Trustworthiness necessitates human understanding and appropriate trust, especially for novel techniques such as deep learning solutions [29], which are commonly black-box in nature. Performance metrics for DNNs typically rely on numerical deviations in input-output assessments, leading to poorly interpretable outcomes and limited generalization beyond the training dataset due to undesired spurious relationships among features and labels. As mentioned in Section 1.2.2, this limitation becomes particularly critical when DNNs are employed as control-oriented models or within control synthesis, potentially yielding effective yet inexplicable behaviors in response to external stimuli.

An intuitive solution to achieve *generalizable* and *interpretable* learning-based solutions is to integrate model-based and data-driven approaches [10], leveraging the strengths of both and hopefully resulting in “interpretable AI techniques”, as outlined in Fig 1.5. For example, domain knowledge can be used for constructing a nominal system model, while learning takes in charge of the residual dynamics that the model cannot reproduce. Yet, integrating knowledge and data remains challenging, particularly as the learning module for residuals lacks transparency and requires interpretation. Evidently, establishing trust has become an exigent requirement for the smooth deployment of AI solutions in real-world safety-critical systems. In the following, we will delve into ongoing endeavors in this direction.

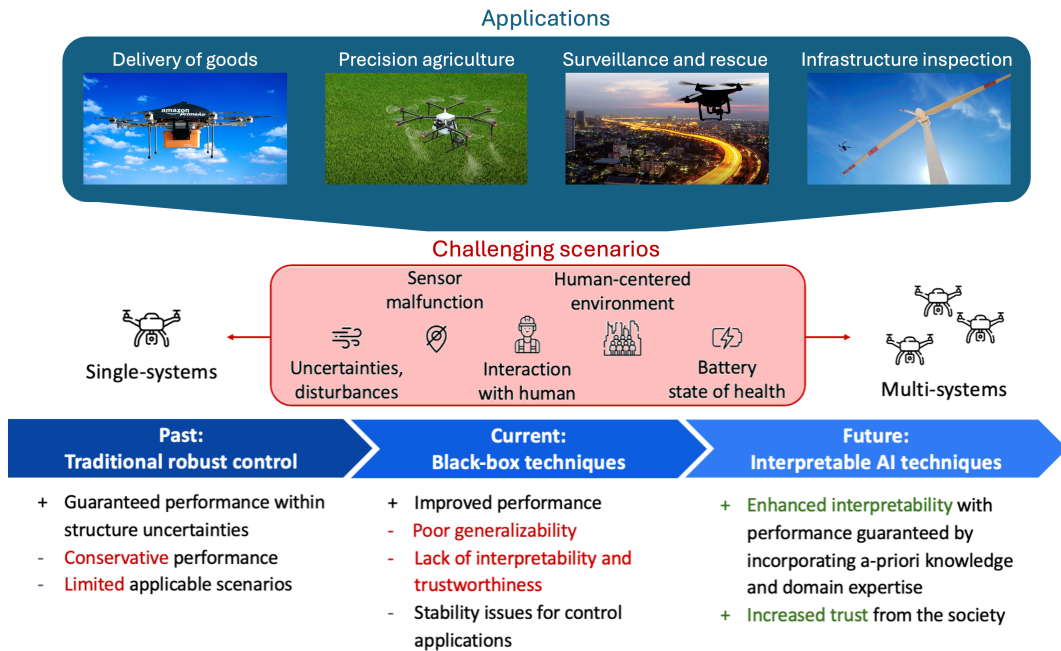


Fig. 1.5 Evolution of UAV control: Current trends and characteristics.

1.3 Building Trust in Algorithms: Trustworthy Machine Learning

While the pursuit of trustworthy AI is still nascent, collective and interdisciplinary efforts are already underway. Notably, *trust* is one of the crucial aspects of achieving a broader concept known as *explainable AI* [29], which also encompasses goals such as *fairness*, *privacy awareness*, and *causality* in the development of novel AI technologies.

Two primary techniques are employed for explainable AI: constructing *transparent models* and providing *post-hoc interpretability analysis*, as depicted in Fig. 1.6 (reproduced from [29]). Transparent models, in contrast to black-box models, inherently offer a certain degree of interpretability. They can be evaluated using qualitative metrics such as simulatability, decomposability, and algorithmic transparency. Simulatability serves as an indicator of model complexity; for example, a simple neural network (e.g., with one hidden layer and five hidden neurons) aligns with this characteristics, while a rule-based system with numerous rules does not. Decomposability refers to the ability to explain individual components of the model such as inputs and calculations, without relying on additional

tools like post-hoc interpretability techniques. Algorithmic transparency, in a broad sense, implies that the model can be comprehensively understood through mathematical analysis.

On the other side of the spectrum, post-hoc interpretability techniques are applied to models that lack inherent interpretability. These techniques aid humans in explaining systems and processes, which include text explanations, visual explanations, local explanations, explanations by example, explanations by simplification and feature relevance explanations techniques.

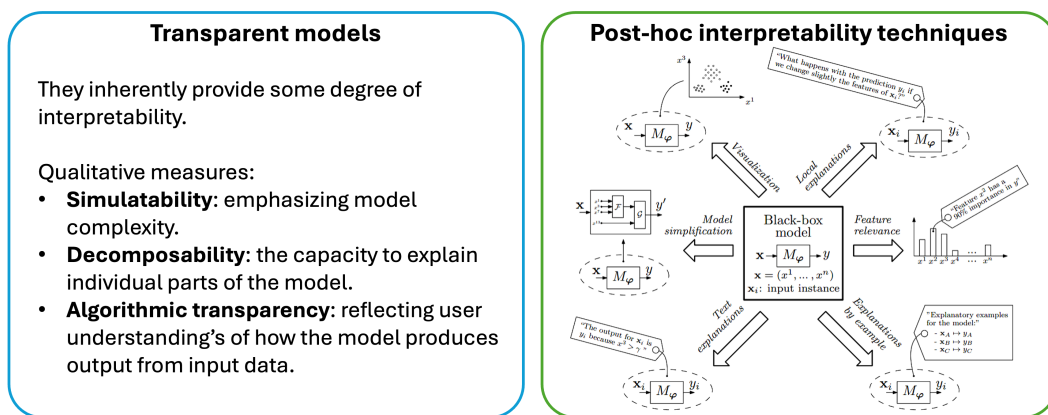


Fig. 1.6 Explainable AI techniques.

1.3.1 Transparent Models

Taking into account the qualitative measures mentioned earlier, namely simulatability, decomposability, and algorithmic transparency, certain machine learning models stand out for their transparency. These include linear/logistic regression, decision trees, K-Nearest Neighbors (KNN), rule-based learners, and Bayesian models; for detailed comparisons, please refer to Table 2 in [29]. However, it is worth noting that ANNs, which are among the most commonly used machine learning models today, lack transparency in most cases, except for shallow and simple structures. This has led to the emergence of a new paradigm known as *Physics-Informed Machine Learning (PIML)* [30].

The emerging paradigm of PIML demonstrates promising capabilities in aligning learning models with physical principles. Illustrated in Fig. 1.7(a), PIML models offer several advantages over data-driven models. Firstly, they exhibit the

ability to learn from limited data and generalize well beyond the training set by integrating domain knowledge. Secondly, with careful incorporation of this knowledge, PIML models can achieve certain degree of interpretability, distinguishing them from purely black-box approaches. In comparison to traditional FPMs, PIML models alleviate the stringent requirements for comprehensive domain expertise to construct accurate models.

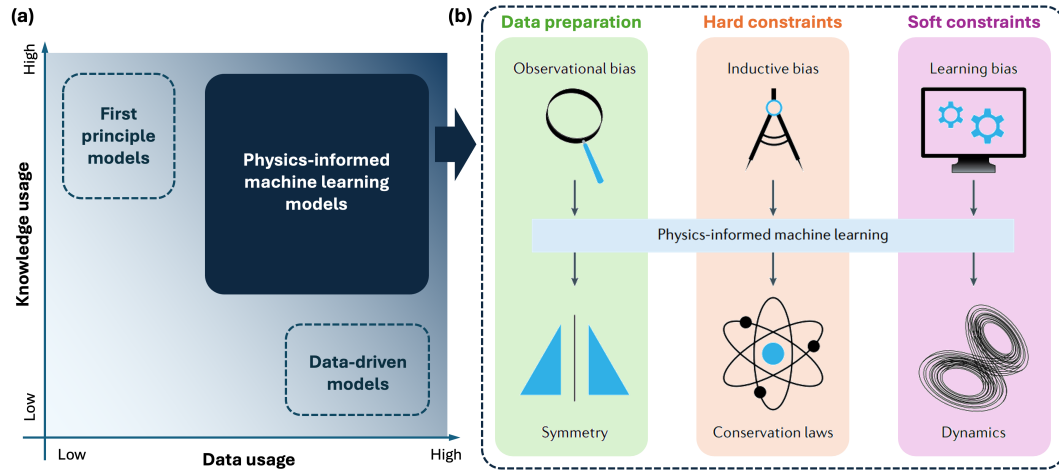


Fig. 1.7 Fundamental principles of PIML.

Developing a PIML solution involves introducing suitable *observational*, *inductive*, or *learning* biases, as illustrated in Fig. 1.7(b) (reproduced from [30]). Observational biases are introduced through data that encapsulate underlying physical laws or through meticulously designed data augmentation techniques. Inductive biases entail prior assumptions that are integrated into the model architecture, ensuring that the predictions adhere to a predefined set of physical laws expressed as hard constraints. Learning biases are introduced through the selection of appropriate loss functions, guiding the model in the training phase to converge towards solutions consistent with the underlying physics. While such soft constraints may only approximately satisfy physical laws, they offer a flexible framework for integrating a broad spectrum of learning biases, including integral and differential equations.

1.3.2 Post-hoc Interpretability Techniques

For non-transparent machine learning models like DNNs, various post-hoc analysis options are available, including *model simplification*, *feature relevance assessment*, and *visualization* [29].

Model simplification techniques reconstruct a new system based on the trained model, aiming to maintain similar performance while reducing complexity. Feature relevance methods indirectly explain the behavior of the model by calculating relevance scores for its variables, revealing their impact on model output.

Visualization techniques provide a straightforward assessment of a model's behavior. To address the curse of dimensionality in high-dimensional datasets, common practices involve using *dimension reduction* techniques such as Principal Component Analysis (PCA), t-distributed Stochastic Neighbor Embedding (t-SNE), and autoencoder. These techniques reduce the dataset's dimensionality before applying *clustering algorithms* for feature extraction on feature vectors. Furthermore, alternative visualization tools for interpreting models include methods like heatmap analysis. Figure 1.8 illustrates the procedural steps for post-hoc visualization of a trained model. It highlights that post-hoc visualization is an additional step performed after model training, making it suitable for transparent, partially transparent, or non-transparent models. Visualizations play a crucial role in enhancing the understanding of complex variable interactions within the model, even for users unfamiliar with machine learning.

1.4 Thesis Outline and Contributions

The remainder of this thesis is structured as follows. Chapter 2 lays the foundation for the subsequent chapters, including notations, mathematical formulations for quadrotor modeling, and background knowledge on ANNs. Chapter 3 explores interpretable neural modeling for quadrotor dynamics using Physics-Informed Neural Networks (PINNs). Thanks to the incorporation of domain knowledge into the machine learning framework, we demonstrate some promising results on the learning capability of PINNs in terms of modeling error and physical consistency, compared to conventional FPM and purely black-box DNN models. In addition, we have made an implementation of multirotor ground effect open source (see

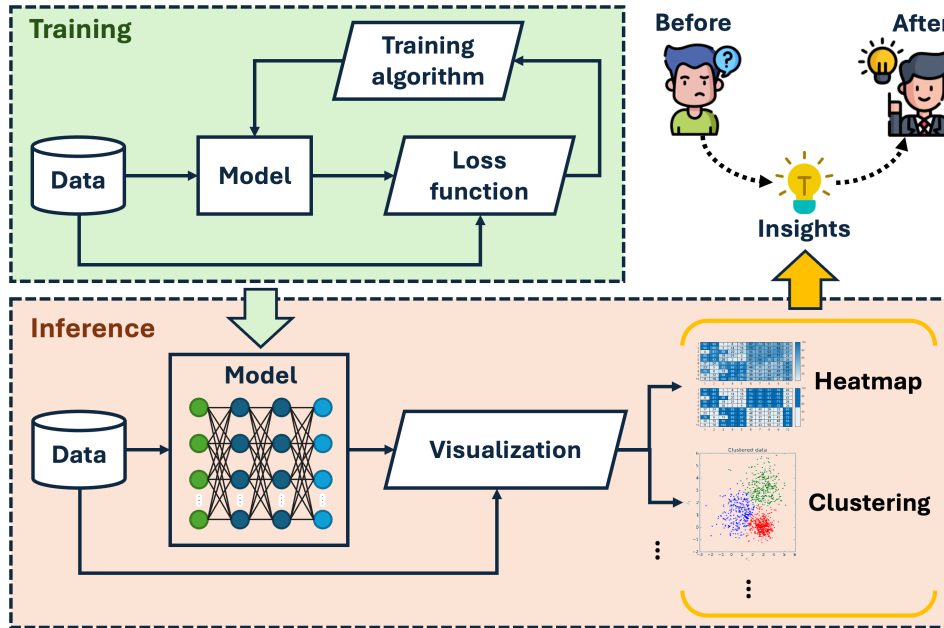


Fig. 1.8 Post-hoc visualization workflow for a trained model.

Appendix A), providing it as a plug-in for Microsoft AirSim simulator to facilitate UAV research endeavors. The majority of the contents therein have been previously published in the *Robotics and Autonomous Systems* journal, as referenced in [31].

Chapters 4 and 5 focus on controller design for quadrotors subject to uncertainties and disturbances. Specifically, Chapter 4 addresses parametric uncertainties in quadrotor mass and inertia matrix, leveraging MBC techniques. This approach is chosen due to the strengths of adaptive control in handling structural uncertainties and interpretability, compared to learning-based approaches. Differing from previous studies, we develop an adaptive geometric controller based on rotation matrices, which demonstrates the capability to execute aggressive maneuvers under uncertainties without suffering from gimbal lock or unwinding phenomena associated with other attitude representations such as Euler angles and quaternions. Projection operators are also introduced to enhance the robustness of the adaptive design against external disturbances or noise in the system. A manuscript containing the majority of this chapter is currently undergoing review by a journal, as cited in [32].

Chapter 5 explicitly handles non-parametric uncertainties such as wind gusts in the controller design. We propose a novel learning-based controller that learns the uncertainties within flight and incorporates the estimated uncertainties into the controller to compensate for them. This online learning capability is endowed with the Reservoir Computing (RC) paradigm. In addition to achieving satisfactory tracking performance, we enhance the interpretability of our learning-based model through post-hoc analysis, allowing for better understanding of model behavior by humans. We also demonstrate the benefits of online learning over offline learning on OOD flight data. A manuscript that includes the majority of this chapter has been accepted by a conference, as cited in [33].

In conclusion, this thesis has made the following main contributions:

- (i) A novel approach for quadrotor dynamical modeling is proposed, which strikes a balance between accuracy and interpretability by integrating knowledge and data.
- (ii) A novel adaptive geometric controller is developed, enabling quadrotors to perform aggressive maneuvers under parametric uncertainties.
- (iii) A novel learning-based controller is introduced to handle non-parametric uncertainties in quadrotors, featuring online learning capability and interpretability.
- (iv) An implementation of multirotor ground effect is presented as a plug-in to AirSim, providing a realistic simulation environment.

Chapter 2

Preliminaries

2.1 Notation

Henceforth, we use case-sensitive bold symbols to represent multidimensional variables, e.g., \mathbf{a} stands for a vector, while \mathbf{A} stands for a matrix. The n -dimension Euclidean space is denoted by \mathbb{R}^n with Euclidean norm $\|\cdot\|_2$. The transpose operator of a vector or a matrix is denoted by $(\cdot)^\top$. The trace, determinant, and eigenvalues of a matrix are denoted by $\text{tr}(\mathbf{A})$, $\det(\mathbf{A})$, and $\lambda(\mathbf{A})$, respectively. The Frobenius norm of a matrix is denoted by $\|\mathbf{A}\|_F = \sqrt{\text{tr}(\mathbf{A}\mathbf{A}^\top)}$. The positive and semi-positive definiteness of a matrix are denoted by $\mathbf{A} > 0$ and $\mathbf{A} \geq 0$, respectively. The symbol \mathbf{e}_3 denotes a vector pointing upwards in the unit 2-sphere $\mathbb{S}^2 = \{\mathbf{a} \in \mathbb{R}^3 : \|\mathbf{a}\|_2 = 1\}$ and \mathbf{I}_n denotes an $n \times n$ identity matrix. The hat operator is denoted by $(\cdot)^\wedge : \mathbb{R}^3 \rightarrow \mathfrak{so}(3)$ and inversely, the vee operator is denoted by $(\cdot)^\vee : \mathfrak{so}(3) \rightarrow \mathbb{R}^3$, where $\mathfrak{so}(3)$ represents the Lie algebra associated with a special orthogonal group $\text{SO}(3) = \{\mathbf{R} \in \mathbb{R}^{3 \times 3} : \mathbf{R}^\top \mathbf{R} = \mathbf{I}_3, \det(\mathbf{R}) = 1\}$. The symbol \mathcal{L}_∞ denotes the space of bounded functions and \mathcal{C}^k ($k = 0, 1, 2, \dots$) denotes the differentiability class of a function if its derivatives up to k^{th} order exist and are continuous.

2.2 Mathematical Model of Quadrotors

We consider the following mathematical model that in general describes the kinematics and the dynamics of a quadrotor

$$\dot{\mathbf{p}} = \mathbf{v}, \quad (2.1a)$$

$$m\dot{\mathbf{v}} = mg\mathbf{e}_3 + \mathbf{R}\mathbf{f}_u + \mathbf{f}_a, \quad (2.1b)$$

$$\dot{\mathbf{R}} = \mathbf{R}\boldsymbol{\omega}^\wedge, \quad (2.1c)$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = -\boldsymbol{\omega}^\wedge\mathbf{J}\boldsymbol{\omega} + \boldsymbol{\tau}_u + \boldsymbol{\tau}_a, \quad (2.1d)$$

where $\mathbf{p}, \mathbf{v} \in \mathbb{R}^3$ denote the position and linear velocity in inertial frame, $\boldsymbol{\omega} = [p, q, r]^\top \in \mathbb{R}^3$ denotes the body-fixed angular rate, and $\mathbf{R} \in \text{SO}(3)$ denotes the rotation matrix from body to inertial frame, constructed by Euler angles $\boldsymbol{\eta} = [\phi, \theta, \psi]^\top$ (roll, pitch, yaw) following “3-2-1” convention¹. Moreover, $\mathbf{f}_u = [0, 0, T]^\top \in \mathbb{R}^3$, $\boldsymbol{\tau}_u = [\tau_{u,x}, \tau_{u,y}, \tau_{u,z}]^\top \in \mathbb{R}^3$ denote the body wrench produced by the four rotors (i.e., control inputs) where $T \in \mathbb{R}$ is the total thrust. $\mathbf{f}_a \in \mathbb{R}^3$, $\boldsymbol{\tau}_a \in \mathbb{R}^3$ denote the lumped uncertainties including unmodeled dynamics and disturbances. $m \in \mathbb{R}$, $\mathbf{J} \in \mathbb{R}^{3 \times 3}$ denote the quadrotor mass and inertia matrix, $g \in \mathbb{R}$ denotes the gravitational constant, $\mathbf{e}_3 = [0, 0, 1]^\top$ is a vector in the unit 2-sphere \mathbb{S}^2 , and $(\cdot)^\wedge : \mathbb{R}^3 \rightarrow \mathfrak{so}(3)$ denotes the hat operator that converts real vectors into Lie algebras. Specifically, the hat operator applied to $\boldsymbol{\omega}$ in (2.1c) is equivalent to the skew-symmetric operator as

$$\boldsymbol{\omega}^\wedge = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \in \mathfrak{so}(3). \quad (2.2)$$

North-East-Down (NED) and Forward-Right-Down (FRD) conventions are adopted for (local) inertial and body(-fixed) reference frame, respectively.

It is important to note that (2.1c) is not the sole representation for rotational kinematics; alternative expressions using Euler angles and quaternions are also

¹The 3-2-1 rotation sequence is commonly used for aerospace applications to describe the orientation of aerospace vehicles from inertial frame to body frame. Specifically, it means rotations about z -axis, then y -axis, followed by x -axis. The right hand rule is adopted for all the rotations.

valid such as

$$\dot{\boldsymbol{\eta}} = \mathbf{T}\boldsymbol{\omega}, \quad \text{where } \mathbf{T} = \begin{bmatrix} 1 & \sin\phi \tan\theta & \cos\phi \tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi / \cos\theta & \cos\phi / \cos\theta \end{bmatrix}, \quad (2.3a)$$

$$\dot{\mathbf{q}} = \frac{1}{2} \mathbf{q} \otimes \boldsymbol{\omega}, \quad (2.3b)$$

where $\mathbf{q} \in \mathbb{R}^4$ denotes quaternions, and \otimes denotes the quaternion product.

By incorporating the rotor model, the generated thrust T and torque $\boldsymbol{\tau}_u$ can be further translated into the actual control command, i.e., the rotor angular speeds n_i ($i = 1, 2, 3, 4$) expressed in revolutions per second, as

$$\begin{bmatrix} T \\ \tau_{u,x} \\ \tau_{u,y} \\ \tau_{u,z} \end{bmatrix} = \underbrace{\begin{bmatrix} c_T & c_T & c_T & c_T \\ -c_T l & c_T l & c_T l & -c_T l \\ c_T l & -c_T l & c_T l & -c_T l \\ c_Q & c_Q & -c_Q & -c_Q \end{bmatrix}}_{:=\mathbf{M}} \begin{bmatrix} n_1^2 \\ n_2^2 \\ n_3^2 \\ n_4^2 \end{bmatrix}, \quad (2.4)$$

where c_T denotes the dimensional thrust coefficient, c_Q denotes the dimensional moment coefficient, and l denotes the moment arm (i.e., the distance from the rotor axis to the principal axis of the quadrotor). Note that \mathbf{M} (referred to as the mixer matrix) depends on the airframe layout. To prevent the need for special case handling and improve reusability in the core controllers, mixer logic is typically separated in the controller design². In other words, thrust and torques are generally treated as the control inputs rather than the actuator commands.

2.3 Exponential and Logarithm Map

The exponential map relates a matrix Lie group to its associated Lie algebra. For rotations, it can be computed through Rodrigues' rotation formula [34] as

$$\mathbf{R} = \exp(\boldsymbol{\phi}^\wedge) = \mathbf{I}_3 + \sin(\phi) \mathbf{a}^\wedge + (1 - \cos(\phi)) \mathbf{a}^\wedge \mathbf{a}^\wedge, \quad (2.5)$$

²A commonly adopted practice of separating mixer logic and controller can be found here: <https://docs.px4.io/v1.12/en/concept/mixing.html>.

where $\boldsymbol{\phi} = \phi \mathbf{a}$, $\mathbf{a} \in \mathbb{S}^2$ is the rotation axis, and $\phi \in \mathbb{R}$ is the rotation angle³.

The inverse of the exponential map is called ‘‘logarithmic map’’ [35] denoted as

$$\boldsymbol{\phi} = \log(\mathbf{R})^\vee, \quad (2.6)$$

which can be computed from

$$\phi = \arccos\left(\frac{\text{tr}(\mathbf{R}) - 1}{2}\right), \quad (2.7)$$

$$\boldsymbol{\phi} = \phi \mathbf{a} = \frac{\phi}{2 \sin(\phi)} (\mathbf{R} - \mathbf{R}^\top)^\vee. \quad (2.8)$$

Remark. *The exponential map from $\mathfrak{so}(3)$ to $\text{SO}(3)$ is surjective-only, meaning that there exist multiple candidates of $\mathfrak{so}(3)$ that yield the same element of $\text{SO}(3)$. This can be revealed by the fact that $\mathbf{R} = \exp((\phi + 2\pi n)\mathbf{a}^\wedge)$, with n being an arbitrary integer and the ambiguity in the sign of ϕ due to the even function $\cos(\phi)$. Nonetheless, we can confine the map such that it is bijective by: (i) limiting $|\phi| < \pi$, and (ii) determining the correct sign of ϕ through verifying if the rotation matrix generated by such ϕ using (2.5) is correct; if not, reversing the sign of ϕ and recalculating the rotation axis. \square*

Moreover, the left Jacobian of $\text{SO}(3)$ [35] is defined as

$$\mathbf{J}_l(\boldsymbol{\phi}) = \int_0^1 \exp(\boldsymbol{\phi}^\wedge)^\alpha d\alpha \quad (2.9)$$

$$= \frac{\sin(\phi)}{\phi} \mathbf{I}_3 + \left(1 - \frac{\sin(\phi)}{\phi}\right) \mathbf{a} \mathbf{a}^\top + \frac{1 - \cos(\phi)}{\phi} \mathbf{a}^\wedge, \quad (2.10)$$

and its inverse is given by

$$\mathbf{J}_l(\boldsymbol{\phi})^{-1} = \frac{\phi}{2} \cot\left(\frac{\phi}{2}\right) \mathbf{I}_3 + \left(1 - \frac{\phi}{2} \cot\left(\frac{\phi}{2}\right)\right) \mathbf{a} \mathbf{a}^\top - \frac{\phi}{2} \mathbf{a}^\wedge. \quad (2.11)$$

Remark. *There exist singularities associated with \mathbf{J}_l using the above equations due to the appearance of $\cot(\phi/2)$ at $\phi = 2\pi n$ with n being an arbitrary integer. To address this, we can use the following approximations: $\mathbf{J}_l(\boldsymbol{\phi}) \approx \mathbf{I}_3 + \frac{1}{2} \boldsymbol{\phi}^\wedge$ and $\mathbf{J}_l(\boldsymbol{\phi})^{-1} \approx \mathbf{I}_3 - \frac{1}{2} \boldsymbol{\phi}^\wedge$ [35]. \square*

³Please note that there is a slight abuse of notation here. The symbol ϕ now denotes the angle of rotation in view of axis-angle representation and has nothing to do with the roll angle previously defined for describing the orientation of the quadrotor.

2.4 Artificial Neural Networks

2.4.1 Historical Perspective

The smallest element in ANNs is named as neuron by analogy with neuro-physiology, whose simplified model was firstly studied by McCulloch and Pitts in the 1940s [36]. Few years later, the very first simple model of ANN, namely perceptron, was proposed by Rosenblatt [37], which was a supervised learning model receiving wide attention. However, subsequent research progress was somehow impeded at that time until Back-Propagation (BP) algorithm appeared in 1986 [38], a powerful tool for training Feedforward Neural Networks (FNNs) proposed by Rumelhart et al. Coupled with Multi-Layer Perceptrons (MLPs), a number of problems on classification and regression could be practically solved with satisfactory results. Nonetheless, the thriving development of Support Vector Machines (SVMs) starting from 1992 witnessed AI winter for the next decade. In 2006, Hinton et al. proposed the concept of deep learning as opposed to shallow learning [39]. Given many more layers, Deep Neural Networks (DNNs) are endowed with much powerful potential, especially in pattern recognition and object detection [40]. Motivated by deep learning, an increasing number of ANNs with more complicated architecture have been popping up since then. Figure 2.1 (reprinted from [10]) depicts the brief history of ANNs indicated with big events happened.

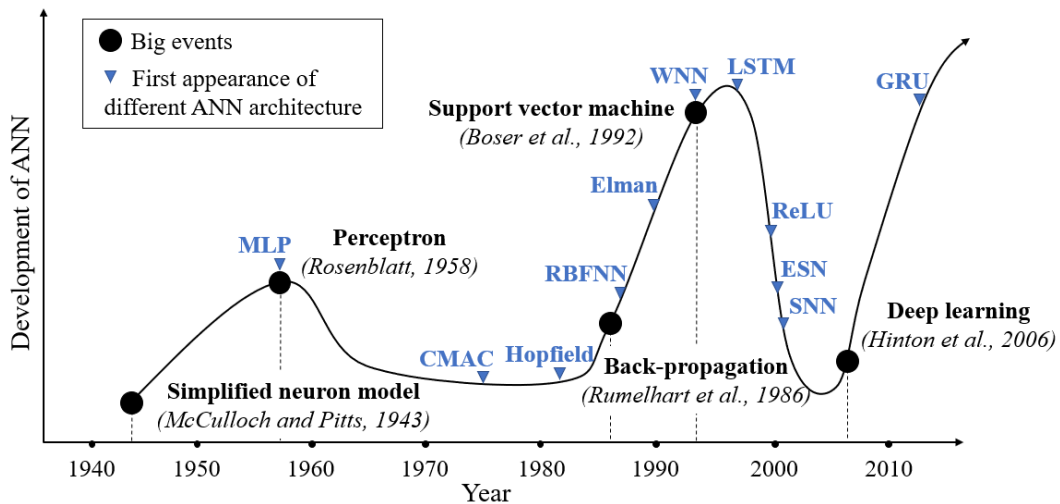


Fig. 2.1 Evolution of ANNs: Timeline and milestones.

2.4.2 Mathematical Model of Artificial Neurons

An artificial neuron (or simply, a neuron) behaves like a function in principle. The simple model of a neuron generally consists of a scalar-valued activation function $f : \mathbb{R} \rightarrow \mathbb{R}$ and two training parameters, namely input weight matrix $\mathbf{W} \in \mathbb{R}^{1 \times N}$ and bias weight $b \in \mathbb{R}$, where N is the number of elements in the input vector $\mathbf{p} \in \mathbb{R}^{N \times 1}$. The output $a \in \mathbb{R}$ is generated through $a = f(\mathbf{W}\mathbf{p} + b)$ where the engaged activation function $f(\cdot)$ could be hard-limit, linear, log-sigmoid or tan-sigmoid function [41], chosen by the designers considering the requirements of network performance. Built by neurons, an ANN can be considered as a typical network, which is made up of a single input layer, single or multiple hidden layers, and a single output layer; see, e.g., Fig. 2.2 (reprinted from [10]). Each layer (except the input) comprises a number of neurons, possibly governed by different activation functions.

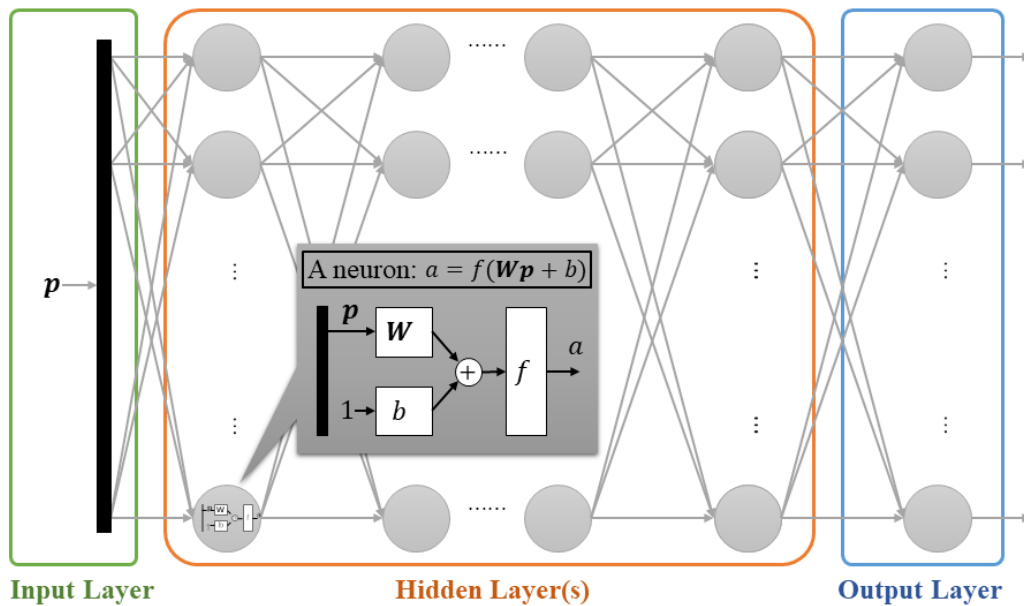


Fig. 2.2 Simplified neuron model and typical architecture of FNNs.

2.4.3 Classifications of Neural Networks

In terms of neuron connections, ANNs can be categorized into one of the following two types: *feed-forward* and *recurrent*. Neurons in an FNN only receives

input from previous layer (see, e.g., Fig. 2.2), whereas feedback connections exist in a Recurrent Neural Network (RNN). Due to these additional connections, RNNs are endowed with large dynamical memory which FNNs generally do not possess, however, at the cost of increasing complexity.

In terms of dynamic characteristics, RNNs may also be named as *dynamic* networks as opposed to *static* networks referring to FNNs. Note that with the use of Tapped Delay Lines (TDLs), FNNs can also be considered as dynamic networks. For example, Focused Time-Delay Neural Network (FTDNN)⁴ (as part of a general class of dynamic networks, namely focused networks) refers to FNNs with tapped delay lines only at the input.

In Table 2.1, some typical and widespread ANNs are presented and classified based on neuron connections. These representatives are also marked in Fig. 2.1 for visualization purposes. In the sequel, we provide a brief introduction to these ANNs, recalling their origin and key features.

MLPs [37] and Radial Basis Function Neural Networks (RBFNNs) [42] were the two well-studied architecture of FNNs originated in the late 1900s, mainly differing in the underlying activation functions. Subsequently, many variations had appeared with different functions taking on the role of activation function. For example, Wavelet Neural Network (WNN) was proposed by Zhang et al. in 1992 to decrease the number of nodes required in the network by using Mexican hat wavelet as activation function [43]; Rectified-Linear Unit (ReLU) was firstly introduced to a dynamical network by Hahnloser et al. in 2000 to solve gradient vanishing issues [44]. On the other hand, motivated by cerebellum neurophysiological model, Cerebellar Model Articulation Controller or Cerebellar Model Arithmetic Computer (CMAC) was initially proposed by Albus for robotic manipulator control in 1975 [45].

Under the category of RNNs, Hopfield networks served as content-addressable memory systems with saturated linear transfer functions, popularized by Hopfield in 1982 [46]. Later in 1990, Elman networks were proposed which are normally two-layer RNNs having a feedback connection from the output of the hidden layer to its input [47]. Due to this additional connection, Elman networks have the capabilities of detecting and generating time-varying patterns. Echo State Net-

⁴More details can be referred to the MathWorks Documentation on “Design Time Series Time-Delay Neural Networks”: <https://ww2.mathworks.cn/help/deeplearning/ug/design-time-series-time-delay-neural-networks.html>.

works (ESNs) are a special type of RNNs, initially introduced by Jaeger in 2001 [48]. Thanks to the concept of RC, the training of such RNNs became conceptually simple and computationally inexpensive. Long Short-Term Memory (LSTM) networks are a typical type of RNNs, firstly proposed by Hochreiter and Schmidhuber in 1997 [49], capable of learning long-term dependencies by solving the problem of vanishing gradients [50]. Similar to LSTM units, Gated Recurrent Units (GRUs) are a gating mechanism particularly used in RNNs while involving fewer parameters, firstly introduced by Cho et al. in 2014 [51].

At last, we stress the importance of a series of biologically plausible neuron models, namely spiking neurons, which describe the properties of biological neurons to communicate via voltage spikes. The earliest spiking neuron model is called Hodgkin-Huxley model [52] while the Leaky Integrate-and-Fire (LIF) model [53] is more preferable in the controller design due to its simpler structure. Such bio-inspired neural networks are known as Spiking Neural Networks (SNNs).

| Type of connection | Neural network model |
|--------------------|---|
| Feed-forward | Multi-Layer Perceptron (MLP) Radial Basis Function Neural Network (RBFNN) Rectified-Linear Unit (ReLU) network Wavelet Neural Network (WNN) Cerebellar Model Articulation Controller (CMAC) |
| Recurrent | Elman network Hopfield network Echo State Network (ESN) Long Short-Term Memory (LSTM) network Gated Recurrent Unit (GRU) network Spiking Neural Network (SNN) |

Table 2.1 Classification of ANNs.

2.4.4 Training Methods

Broadly speaking, machine learning encompasses three main paradigms: *supervised*, *unsupervised*, and *reinforcement* learning. In supervised learning, algorithms learn from labeled data, while unsupervised learning discovers patterns

from unlabeled data without explicit supervision. Reinforcement learning, in contrast, enables agents to learn by interacting with an environment, receiving feedback through rewards or penalties.

In the context of supervised learning, the BP algorithm stands out as a widely adopted techniques for training ANNs. Rooted in gradient descent, the BP algorithm iteratively adjusts network parameters – namely, weights and biases. The procedures for updating these parameters through the BP algorithm are as follows:

- (i) Take a batch of training data.
- (ii) Perform forward-propagation to calculate the loss (e.g., the difference between predicted and labeled output).
- (iii) Perform back-propagation on the loss to derive gradients with respect to network parameters.
- (iv) Update network parameters through gradient descent.

However, the BP algorithm works only for FNNs and requires modifications to be successfully applied to RNNs. For example, Back-Propagation Through Time (BPTT) [54] and Real Time Recurrent Learning (RTRL) [55] are the two alternatives for training RNNs.

Besides gradient-based approaches, linear regression proves to be an efficient and effective method for certain ANN architectures (e.g., ESNs) due to its one-shot training process. Moreover, training ANNs extends beyond supervised learning methods and includes approaches like RL or computational methods such as Particle Swarm Optimization (PSO). In the context of controller design, network parameters can also be adjusted using Lyapunov stability theory, providing the essential mathematical proof of stability.

Depending on the mode of network training, ANNs can be further classified as either *offline* or *online* networks. Offline networks refer to those with pre-trained and fixed network parameters. For applications where offline training time is not critical, such networks are prevalent as they offer, by and large, higher accuracy models. On the contrary, online networks refer to those with online adaptation of network parameters. Due to this unique learning ability, online networks are of great interest to applications where online learning is desirable. For example,

constructing the dynamic model of a UAV using an online network would be more ideal, allowing for the automatic capture of all dynamic changes by adjusting the network weights in real-time. However, since online networks involves online adaptation, the resulting learning process can be computationally expensive and may suffer from divergence owing to a paucity of data.

Chapter 3

Physics-informed Neural Modeling

3.1 Introduction

As partially discussed in Section 1.2, the 21st century has witnessed an explosive growth of civil applications of UAVs, ranging from the inspection of industrial infrastructures such as power line and wind turbine to the operations in human-interactive environments such as delivery of goods, to name a few [56]. Such a scenario unprecedentedly calls for control algorithms that not only provide sufficient tracking performance, but also enable *safe* and *trustworthy* operations, especially in complex and populated environments, to facilitate real-world deployment as well as to avoid injury and property damage by all means.

MBC techniques have found great applicability in the last decades thanks to the underlying analytical formulation of system dynamics in terms of differential equations, which typically informs the design of *performance*-guaranteed control techniques [10]. However, a remarkable drawback of MBC techniques is their heavy reliance on the accuracy of the mathematical model of the system under control [18]. *Uncertainties* and *disturbances* such as parametric uncertainty and unmodeled dynamics are ubiquitous in real-world flight, which may set back the derivation of such control-oriented models, with a clear adverse impact on the control performance. A typical example is the loss of system stability, which, for UAVs, results in dangerous and uncontrollable deviations from the planned path, eventually incurring accidents [17].

To address the modeling issues induced by uncertainties and disturbances, many efforts in the control community have been put forward in robust and adaptive control theory [18]. For example, adaptive backstepping controllers were designed to account for the changes in mass and inertia matrix of UAV [19], and in payload [9]; another adaptive sliding backstepping control scheme was proposed to guarantee attitude tracking under unmodeled dynamics [20]. Nonetheless, most of the resulting controllers are usually mathematically complex and the design very often leans toward conservative approaches, tuned on the demanding, yet unlikely, worst-case scenario, thus limiting the possibilities of guaranteeing desired performance [18].

On the other side of the spectrum, data-driven approaches have been pursued with the aim to learn the real system dynamics in its entire complexity from empirical data sets, including uncertainties and disturbances, which could be subsequently used for MBC design. ANNs, as *universal approximators* with powerful learning capability, have probably raised the most interest for UAV dynamical modeling due to the advances in computer hardware in recent years; see, e.g., [26–28, 57, 58]. As they started to be applied to the growing UAV industry, *trustworthiness* becomes an exigent demand, which by definition [29], requires human to understand, appropriately trust, especially for new techniques. Due to the *black-box* (a.k.a., model-agnostic) nature of ANNs, however, the performance metrics mainly rely on the assessment of input-output numerical deviations of the available data. Hence, the outcome of the learning process is typically *poorly interpretable*, often implying a lack of generalization capability outside the training data set due to undesirably learned spurious relationships among features and labels. This issue becomes particularly harmful when ANNs are used as a control-oriented model to design control actions, since such systems respond to external stimuli with behaviors that are maybe effective yet inexplicable. While nowadays practitioners and society are far from being concerned just by performance, such risk evidently reduces the overall trustworthiness of the control system [29].

To meet the expectations of both *accurate* and *interpretable* neural modeling, here we propose a novel approach for dynamical modeling of quadrotors, inspired by PIML [30], an emerging machine learning paradigm that aims at achieving better generalization capability by incorporating a-priori system knowledge into the learning process. Such paradigm has been successfully applied for modeling in many scientific and engineering disciplines such as acoustic field of a

quadrotor [59–61], robotic manipulator [62–64], lake temperature [65], pandemic spread [66], just to list a few. In our context, we seamlessly embed the law of conservation of momentum into the training of a DNN in the form of *soft constraints* (or *learning bias*) to model fast, high-dimensional, and highly nonlinear dynamics. Comparison and ablation studies have been carried out over multiple seeds in a visual and physical simulator that we customized on top of AirSim [67] on several data sets including aerodynamics such as drag, ground effect, and periodic wind. It has been consistently shown that our proposed PINN outperforms both linearized mathematical model and purely black-box approach such as vanilla DNN [58] in terms of test error (generalizability on unseen data) and exhibits better learning capability of underlying relationships than vanilla DNN by means of Covariance Confidence Ellipse (CCE) [68], a *post-hoc* model interpretability technique introduced to reveal *physical consistency* of the learned model.

3.2 Related Work

Recent endeavors that are closely aligned with our work stem from two distinct communities: *UAV* and *machine learning*. However, each community has different research focuses, which are outlined as follows:

(i) Learning-based dynamical modeling of UAVs

Previous studies in the UAV community have extensively explored the effectiveness of deep learning techniques for dynamical modeling of UAVs, owing to their ability to capture intricate features hidden within the data. Notable examples include the successful adoption of DNN for learning the dynamics of helicopters [57] and quadrotors [58] from real flight data. Besides full dynamics modeling, recent research has demonstrated the feasibility of utilizing DNN to learn residual forces and torques resulting from advanced aerodynamics such as quadrotor ground effect [26] and aerodynamic interactions between multicopters [27]. However, despite the promising results achieved in these studies, vanilla DNN architectures are typically employed, which are notorious for their purely black-box nature and may result in learning physically inconsistent relationships and reduced generalization capability [30].

In regard to this issue, only a few attempts have been made to enhance the interpretability of learned models. For example, in [69], a hybrid approach combining analytical and empirical techniques was proposed for linear velocity estimation of a quadrotor. Similarly, [16] adopted a hybridization strategy for quadrotor dynamical modeling, combining Blade-Element-Momentum (BEM) theory with a neural network to compensate for residual dynamics using real flight data of agile maneuvers. However, the structure and training process of these networks remain opaque. More recently, Domain Adversarially Invariant Meta-Learning (DAIML) was proposed to learn the shared representation of winds offline and then update wind-specific linear coefficients online [28]. While DAIML offers interpretability by visualizing clustered linear coefficients associated with similar wind speeds, it necessitates comprehensive data collection across various wind conditions, which can be labor-intensive and time-consuming.

(ii) **Learning dynamics from trajectory data using PINNs**

The recent success of using PINN to solve Partial Differential Equations (PDEs) [70] has sparked a surge of interest among machine learning researchers in learning dynamics from trajectory data while incorporating physical insights. In this context, research efforts can broadly be categorized into two streams: *structured* and *unstructured* learning.

Structured learning frameworks such as Lagrangian neural networks [62–64] and Hamiltonian neural networks [71, 72] leverage Lagrangian and Hamiltonian mechanics to inform the structure of neural ODE of the system. These approaches offer the advantage of guaranteeing energy conservation, provided the system does not involve non-conservative forces such as friction. While this could hold true for the dynamical modeling of robotic manipulators, the primary application of research within this stream, the presence of aerodynamic effects can substantially impact quadrotor flight dynamics, thereby posing challenges to applying these frameworks in such scenarios.

On the other hand, unstructured learning incorporates a priori domain knowledge into the loss function as learning bias to achieve more physically consistent predictions. Sometimes referred to as Theory-Guided Data Science (TGDS) [73], the crux of unstructured learning involves embedding knowledge from the outset [29] and assessing its compliance with the

knowledge eventually learned by the model. Ongoing research in various scientific and engineering domains has demonstrated the effectiveness of this approach, including acoustic field modeling of quadrotors [59–61], lake temperature modeling [65], and pandemic spread prediction [66].

3.3 Physics-informed Neural Network

In this section, we detail our proposed PINN for modeling the inverse dynamics of the quadrotor. Following the unstructured learning strategy, our primary goal is to identify proper domain knowledge and incorporate it into the network training process. This enables the resulting network to generate predictions that align more closely with physical principles. In other words, this process can also be seen as the development of a customized physics-informed loss function, which guides the network – whether it takes the form of a FNN or RNN – to learn the underlying relationships within the data.

The main concept is illustrated in Fig. 3.1. The PINN is essentially a vanilla DNN trained using a physics-informed loss function, integrating prior domain knowledge. This differs from the conventional approaches that use a model-agnostic loss function. In addition to assessing prediction accuracy, the performance of the network will be further evaluated through post-hoc visualization. This analysis aims to understand the capability of the network to effectively learn prior knowledge, which can subsequently inform controller design.

In the sequel, we begin by providing the details of the chosen network architecture. Then, we delve into the rationale behind and the methodology of integrating physics principles into network training through a customized physics-informed loss function. Furthermore, we shed light on the post-hoc interpretability visualization, which serves to validate the proficiency of the network in extracting underlying knowledge. Additionally, we introduce a practical technique for fine-tuning regularization parameters to enhance training performance specifically tailored to our application.

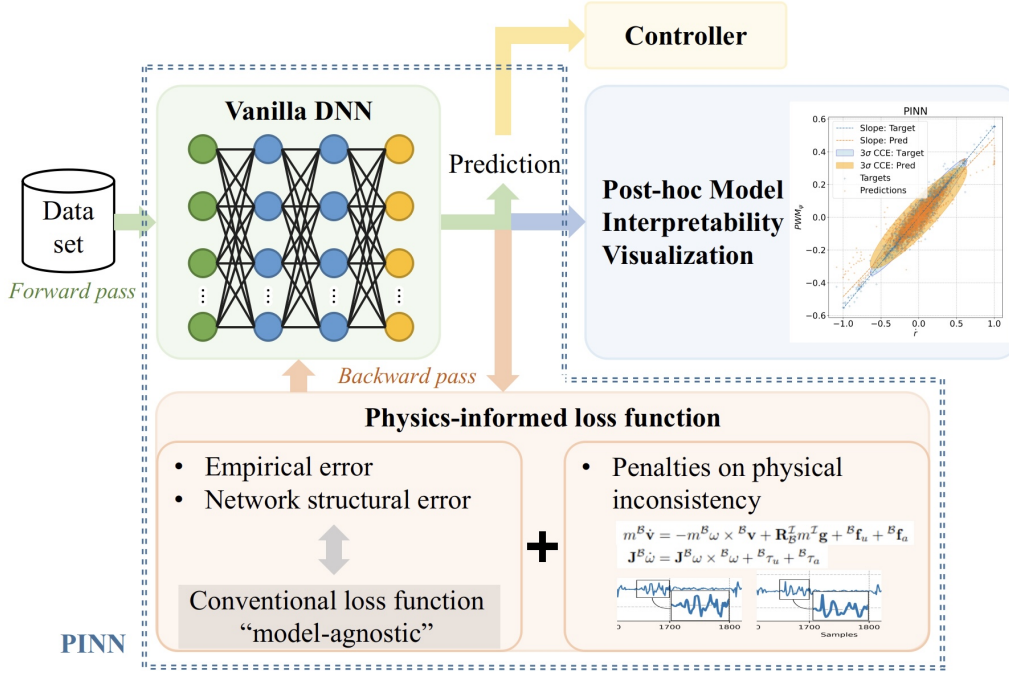


Fig. 3.1 Main concept of PINN.

3.3.1 Network Structure

The proposed PINN, which approximates the inverse of quadrotor dynamics (2.1), can be expressed as $\hat{\mathbf{y}} = \text{PINN}(\mathbf{x}; \Theta)$, where \mathbf{x} denotes the network input in the form of a vector consisting of quadrotor states and accelerations, $\hat{\mathbf{y}}$ denotes the network predictions on control inputs, and Θ denotes the trainable network parameters. Since yaw angle may suffer from abrupt changes from 0 to 2π , sine and cosine operators are used for embedding to ensure the continuity of the signal. Besides, we use the dimensionless form of Pulse-Width Modulation (PWM) of rotors as control inputs due to the signal property from the visual and physical simulator that we set up for data collection as will be discussed in Section 3.4.1. Therefore, the input (or feature) and the output (or prediction, denoted with $\hat{\cdot}$) of the network are formulated as follows:

- $\mathbf{x} := (\hat{\mathbf{v}}^\top, \hat{\boldsymbol{\omega}}^\top, \mathbf{v}^\top, \boldsymbol{\omega}^\top, \phi, \theta, \sin \psi, \cos \psi)^\top \in \mathbb{R}^{15}$,
- $\hat{\mathbf{y}} := (\hat{u}_1, \hat{u}_2, \hat{u}_3, \hat{u}_4)^\top \in \mathbb{R}^4$, where u_i is the PWM signal of each rotor.

We opt for a DNN as the foundational architecture for our network. Hence, our PINN can be re-reformulated as

$$\text{PINN}(\mathbf{x}, \Theta) := f \circ \underbrace{g \circ \dots \circ f \circ g}_{N_l \text{ hidden layers}}(\mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}), \quad (3.1)$$

where $\{\mathbf{W}^{[1]}, \mathbf{b}^{[1]}\}$ is the pair of network parameters of the first layer in Θ , $g(\cdot) = \text{ReLU}(\cdot)$ is the activation function, and $f(\beta) = \mathbf{W}^{[i]}\beta + \mathbf{b}^{[i]}$, with $i = 1, 2, \dots, N_l$. Note that if the model described in Eq. (3.1) is trained using a conventional loss function such as the Mean Square Error (MSE) between \mathbf{y} and $\hat{\mathbf{y}}$, it essentially constitutes a vanilla DNN, as demonstrated in prior works (e.g., see [58]). However, in the subsequent discussion, we elaborate on our customized physics-informed loss function. This function is meticulously designed with the specific objective of guiding the network to capture the intrinsic relationships within the input-output data pair, thereby facilitating generalization capability to unseen data sets.

3.3.2 Incorporation of Physics as Learning Bias

The absence of universally accepted mathematical representations or empirical domain expertise in advanced aerodynamics presents a big challenge when attempting to incorporate domain knowledge into the learning process. Consequently, we pivot towards harnessing fundamental physical principles to steer the network training. For quadrotors, the law of conservation of momentum emerges as an effective guide, as it inherently governs the dynamical equations and embodies the fundamental property of the system. Thus, the crux of the issue boils down to integrating this physical law into the network training process; or in other words, effectively embedding physics into loss terms that the network can learn.

The embedding of the law of conservation of momentum is motivated by the observed correlation between accelerations and control inputs. Using our collected data, which includes agile maneuvers with drag, as shown in Fig. 3.2, we can easily assess the Pearson Correlation Coefficient (PCC) between the derivative of roll angular rate \dot{p} and the PWM signals utilized to generate such acceleration. This correlation coefficient is calculated at 0.67, indicating a significant correlation between these two quantities. Graphically, the data series exhibit a remarkable

similarity in pattern, as evident in Figs. 3.2 (a)&(b). This strong correlation also holds true for pitch and yaw motion, as illustrated in Figs. 3.2 (e)&(f), (i)&(j). For better visualization, we present the sorted samples with CCE [68], which represents the enclosure of 98.9% of data, as depicted in Figs. 3.2 (d)&(h)&(i). The positive correlation can be visually implied by the slope (or, more precisely, the rotation angle) of the CCEs. From a mathematical standpoint, it becomes straightforward that $\dot{\omega}$ and τ_u are collinear under conditions where nonlinearities and aerodynamics are marginal; see, e.g., the subsequent linearized quadrotor dynamics derived from (2.1b) and (2.1d)

$$\begin{aligned} \dot{v}_x &= -\frac{1}{m}\theta T, & \dot{v}_y &= \frac{1}{m}\phi T, & \dot{v}_z &= -\frac{1}{m}T + g, \\ \dot{p} &= \frac{1}{J_{xx}}\tau_{u,x}, & \dot{q} &= \frac{1}{J_{yy}}\tau_{u,y}, & \dot{r} &= \frac{1}{J_{zz}}\tau_{u,z}, \end{aligned} \quad (3.2)$$

where J_{xx}, J_{yy}, J_{zz} denote the moment of inertia along the principal axes.

Based on the aforementioned empirical findings, we devised a local monotonicity loss term to embed the law of conservation of momentum into the loss function for network training. This term penalizes the deviations from consistent patterns between two data series. More specifically, if one data series increases while another decreases, the network is penalized by incorporating the local monotonicity loss term into the overall loss function during training. The detailed implementation of the local monotonicity loss term is outlined in Algorithm 1, where we utilize the differentiable approximation of the sign function, $\tanh(\cdot)$, to assess if two data series share the same pattern. Additionally, we apply the local monotonicity loss term to all three degrees of rotation, resulting in the final physics-informed loss function for network training as

$$\mathcal{L} = \mathcal{L}_{\text{MSE}} + \sum_i \lambda_{\text{LM}}^i \mathcal{L}_{\text{LM}}^i, \quad i = \{\phi, \theta, \psi\}, \quad (3.3)$$

where λ_{LM} is the hyperparameter and \mathcal{L}_{MSE} is the (conventional) MSE between targets and predictions.

Note that we chose not to directly incorporate the PCC into the composition of the final loss function. This decision stems from the fact that PCC is dependent on the assumptions on the data distribution and may result in poor training performance. However, it serves as an effective metric for assessing the correlation

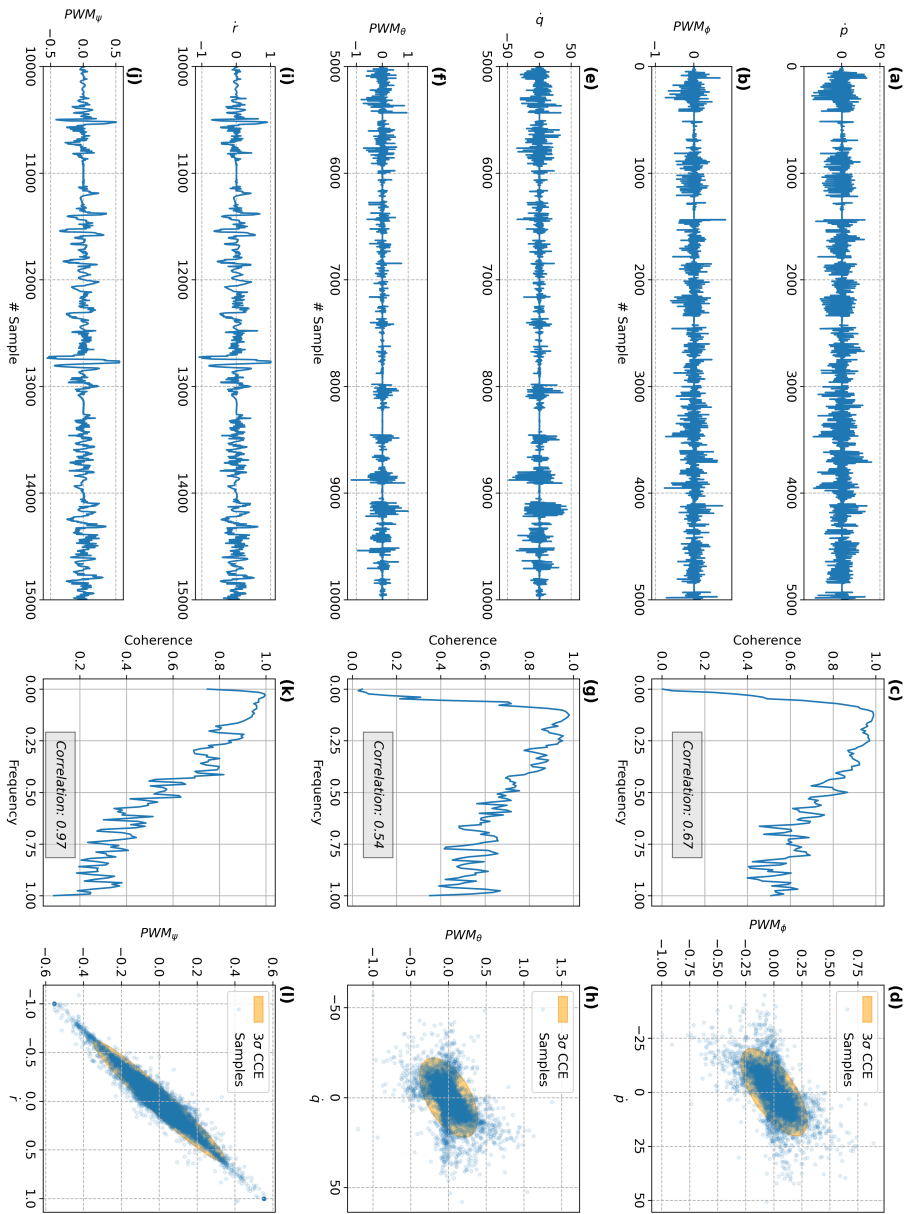


Fig. 3.2 **Correlation within training data series.** (a), (b), (e), (f), (i), (j): The derivative of angular rate and the corresponding PWM signals. (c), (g), (k): Coherence plots for roll, pitch, and yaw motion, with PCCs indicated in gray boxes. (d), (h), (l): CCE plots for roll, pitch, and yaw motion, depicting data samples (blue dots) and 3 σ CCEs (orange ellipses).

between variables once the data vectors are available. Notably, it may happen that two data series with different patterns might have the same PCC value¹. Hence, using PCC for training would steer the network parameters toward similar values even in presence of different input patterns, clearly leading to ill-posed optimization problems.

Furthermore, it is important to acknowledge that the observation of shared patterns between data series may not always hold true, and hence, neither does the devised local monotonicity loss term. This is because our proposed embedding (i.e., local monotonicity loss term) is an approximation rather than a precise mathematical equation. However, this discrepancy typically arises only when the system dynamics are dominated by the highly nonlinear aerodynamic terms. To address this challenge, we treat the local monotonicity loss as an auxiliary learning bias – a soft constraint that can be violated – to enhance network learning through the introduction of a hyperparameter λ_{LM} , which governs the strength of the local monotonicity loss term. Additionally, the value of λ_{LM} is dynamically adjusted using the cyclical annealing method [74] as explained next.

Algorithm 1: Calculation of physics-informed loss function

Input: Training data \mathcal{D} , network parameter Θ
Parameter: Batch size N_b , hyperparameter λ_{LM}
Output: Physics-informed loss \mathcal{L}

- 1 **Function** LocalMonotonicityLoss(m, n):
- 2 $\text{LM}_m, \text{LM}_n \leftarrow \tanh(m[2, N_b] - m[1, N_b - 1]), \tanh(n[2, N_b] - n[1, N_b - 1])$
- 3 $\text{loss} \leftarrow \frac{1}{N_b} (1 - \text{LM}_m \times \text{LM}_n)$ // element-wise multiplication
- 4 **return** loss
- 5 $X_{\text{batch}}, Y_{\text{batch}} \leftarrow \text{DataLoader}(\mathcal{D}, N_b)$ // X is feature, Y is label
- 6 $\hat{Y}_{\text{batch}} \leftarrow \text{PINN}(X_{\text{batch}}, \Theta)$ // forward propagation
- 7 $\mathcal{L}_{\text{MSE}} \leftarrow \frac{1}{N_b} (\hat{Y}_{\text{batch}} - Y_{\text{batch}})^2$ // conventional MSE
- 8 $\mathcal{L}_{\text{LM}} \leftarrow \text{LocalMonotonicityLoss}(\hat{Y}_{\text{batch}}, X_{\text{batch}})$
- 9 $\mathcal{L} \leftarrow \mathcal{L}_{\text{MSE}} + \lambda_{\text{LM}} \mathcal{L}_{\text{LM}}$
- 10 **return** \mathcal{L}

¹An explanatory example: Let the first pair of data series be $x_1 = [1, 2, 3, 4, 5]$, $y_1 = [1, -2, -3, 4, 5]$ and the second pair of data series be $x_2 = [1, 2, 3, 4, 5]$, $y_2 = [1, 2, 3, 1.9455, 2.5]$. Although both pairs have similar PCC values of approximately 0.626, the increasing/decreasing pattern between x_i and y_i , where $i = 1, 2$, significantly differs. On the other hand, the proposed local monotonicity loss can tell the difference in patterns, with value of 1.0 for $i = 1$ and of 0.5 for $i = 2$.

3.3.3 Cyclical Annealing Scheduler

When aerodynamics dominate over the body wrench generated by the four rotors, the two loss terms in Eq. (3.3) may contradict each other, leading to ineffective guidance for the network in learning the real aerodynamic effects. To mitigate this issue, we employ a practical strategy known as cyclical annealing scheduler. Originally introduced to address the Kullback–Leibler (KL) vanishing problem in Variational Autoencoders (VAEs) [74], this technique cyclically adjusts the hyperparameter of the KL regularizer. Similar to our physics-informed loss function, a VAE loss function comprises MSE (i.e., reconstruction error) and the KL regularizer. By scheduling the hyperparameter of the KL regularizer in a cyclical fashion, it was empirically found that the network can leverage latent codes learned in the previous cycle as warm re-starts, thereby progressively improving the performance.

In our approach, we apply a similar concept, but with inverse scheduling as

$$\lambda_{\text{LM}} = \begin{cases} \frac{1-\beta}{1-R} \lambda_{\text{max}} & \text{if } \beta > R, \\ \lambda_{\text{max}} & \text{if } \beta \leq R, \end{cases} \quad \text{with } \beta = \frac{\text{mod}(k, \lceil T/M \rceil)}{\lceil T/M \rceil}, \quad (3.4)$$

where k denotes the current epoch index, T denotes the maximum number of epochs, λ_{max} denotes the maximum value to which λ_{LM} will be annealed, M denotes the number of cycles, R denotes the proportion used to maintain λ_{max} , and $\lceil \cdot \rceil$ denotes the ceiling function that maps the argument to the smallest integer greater than or equal to its value. As such, the training process is split into M cycles, each starting from λ_{max} , where the network predominantly adheres to local monotonicity, and gradually decreasing to 0. At this point, our physics-informed loss function transitions towards resembling the conventional loss function, aiming solely to minimize the MSE. This cyclical progression is visualized in Fig. 3.3 with parameters: $(M, R, \lambda_{\text{max}}, T) = (5, 0.5, 0.1, 300)$. Empirical evidence suggests that this scheduling strategy facilitates improved performance over successive cycles.

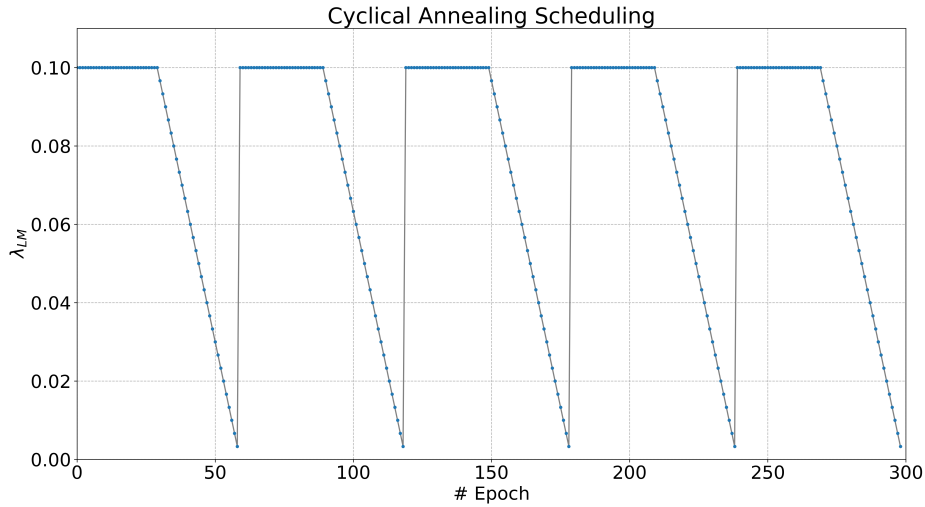


Fig. 3.3 Illustration of cyclical annealing scheduler λ_{LM} in Eq. (3.4).

3.3.4 Post-hoc Model Interpretability Visualization

To cultivate trustworthy AI, it is essential not only to develop transparent models such as our proposed PINN, but also to employ post-hoc interpretability techniques. These techniques include explanations by example, text explanations, and feature relevance, among others [29].

In this context, we utilize post-hoc visualization methods to enhance the interpretability of the learned model. Unlike the approach in [28], where dimensionality reduction technique was employed to cluster different wind conditions, we use CCE, as discussed earlier. CCE provides insights into both PCC and MSE, making it a valuable tool for assessing the model's behavior during training or deployment. Indeed, the rotation angle of the CCE allows for a comprehension of the PCC, while the height and width of the CCE offer information about the MSE. This dual representation enables a comprehensive understanding of the model's performance, facilitating interpretation for users who may not be familiar with the intricacies of deep learning.

3.4 Results and Discussion

3.4.1 Simulator

To streamline data collection, our work employs a simulator built upon the visual and physical simulator AirSim [67], Unreal Engine 4 (UE4)², PX4 [75], and QGroundControl³; see Figs. 3.4 (a) and (b) for illustration. Compared with other open-source simulator alternatives such as Gazebo [76], X-Plane⁴ and MathWorks quadcopter project⁵, AirSim offers a more complex and realistic environment. Leveraging advanced rendering techniques from UE4 and incorporating various sensors and aerodynamics features such as drag force, AirSim provides a sophisticated simulation platform with a physical engine at its core. Additionally, its exceptional extensibility allows for seamless integration with different hardware platforms and software protocols, while also enabling customization of personalized Application Programming Interfaces (APIs) and functionalities using C++ and Python.

In our experiments, alongside data collection coding, we developed a custom C++ class integrated with AirSim's source code to simulate the quadrotor ground effect, thereby further enhancing the fidelity of our simulator. Specifically, we incorporated two ground effect models: a simple Cheeseman-Bennett model [77] and a parametric model derived from polynomial fitting using empirical data [78], as depicted in Fig.3.5. Moreover, we provide user-level APIs in both C++ and Python, namely `simSetGroundEffect`, to facilitate the research efforts of other UAV community researchers. Our implementation complements the advanced aerodynamics already available in AirSim such as drag, enabling users to safely collect representative flight data under ground effect conditions by flying the quadrotor very close to the ground in the simulator. For more detailed design information and usage instructions regarding our ground effect implementation, readers are kindly directed to either Appendix A or the GitLab repository. The GitLab repository contains a self-explanatory README file that provides comprehensive

²An advanced real-time 3D creation tool: <https://www.unrealengine.com/en-US>.

³A ground control station for the MAVLink protocol: <http://qgroundcontrol.com/>.

⁴X-Plane, a flight simulator: <https://www.x-plane.com/>.

⁵MathWorks quadcopter project based on the Parrot[®] series of mini-drones: <https://www.mathworks.com/help/aeroblks/quadcopter-project.html>.

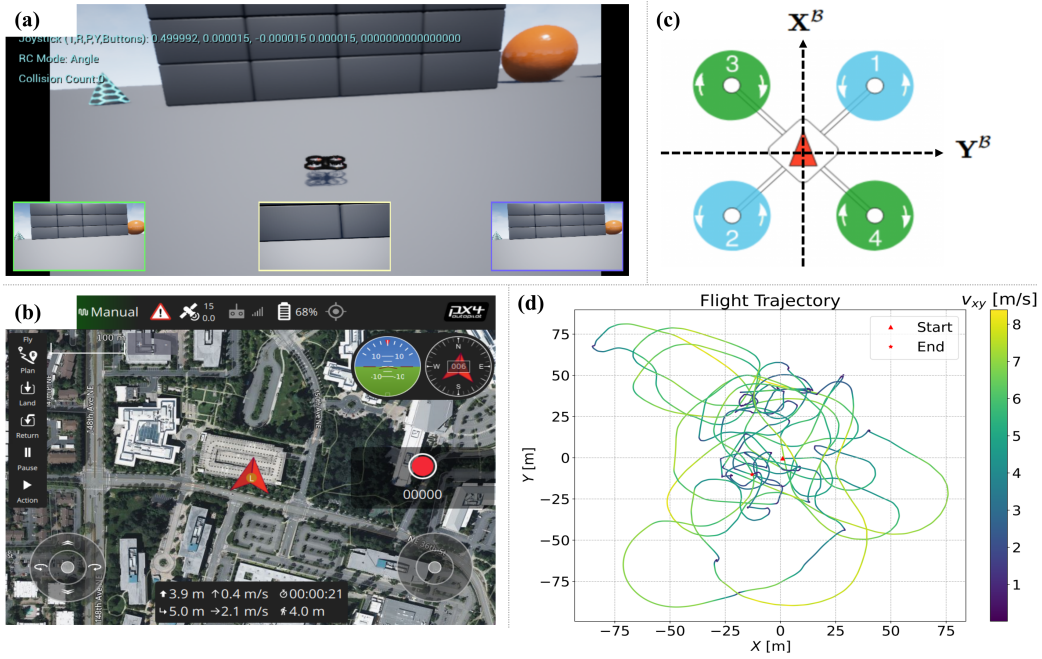


Fig. 3.4 **Data collection in the visual and physical simulator.** (a) Manual flight of a quadrotor in the simulator. (b) User interface of QGroundControl. (c) Configuration of the employed quadrotor overlaid with a body-fixed reference frame. (d) Flight trajectory with a colorbar indicating the magnitude of linear velocity.

information about the project. One can access the repository and the README file at the following URL: <https://gitlab.com/PoliToComplexSystemLab/AirSim-GE>.

3.4.2 Network Training

Thanks to the visual and physical simulator described in Section 3.4.1, training data sets can be easily and safely collected through manual flight of the quadrotor. The quadrotor specifications⁶ are given in Table 3.1. Various maneuvers, including aerodynamics and other uncertainties, can be performed using a joystick or gamepad. In total, we collected three data sets for our experiments:

- (i) \mathcal{D}_1 : Agile maneuver with maximum linear speed up to 8 m s^{-1} and drag force, resulting in approximately 15k data samples (see Fig. 3.4 (d)),
- (ii) \mathcal{D}_2 : Near-ground flight incorporating both drag force and ground effect [78], resulting in approximately 6k data samples (see Fig. 3.5),

⁶The thrust and power constants reported here are dimensionless.

| Parameter | Notation | Selected Value |
|--------------------|----------|--------------------------|
| Mass | m | 1.5kg |
| Inertia | J_{xx} | $1.469e-2 \text{ kgm}^2$ |
| | J_{yy} | $1.686e-2 \text{ kgm}^2$ |
| | J_{zz} | $3.093e-2 \text{ kgm}^2$ |
| Thrust constant | C_T | $1.099e-1$ |
| Power constant | C_P | $4.016e-2$ |
| Propeller diameter | D | 0.2286m |
| Propeller spacing | $2l$ | 0.690m |

Table 3.1 Quadrotor specifications.

- (iii) \mathcal{D}_3 : Agile maneuver with both drag force and periodic wind, resulting in approximately 5k data samples (see Fig. 3.6).

All three data sets comprise the physical quantities of the quadrotor necessary for network training and ablation studies, with a fixed sampling time of 0.05 sec.

After a trial-and-error procedure, we determined that using 10 hidden layers, each with 25 hidden neurons, resulted in neural networks with satisfactory performance after training. Prior to training, we initialized all network parameters (i.e., Θ) using the Xavier initialization method, and we selected a batch size of 64 data samples. We employed the Adam optimizer with weight decay (an alternative to network structural error) during training to mitigate the risk of overfitting.

3.4.3 Model Comparison and Ablation Study

To demonstrate the effectiveness of the proposed PINN, we performed extensive comparisons among a variety of models, which can be categorized into:

- (i) linearized mathematical model,
- (ii) purely black-box model (i.e., vanilla DNN similar to [58]),
- (iii) PINN model (with same structural settings as the vanilla one).

Additionally, an ablation study was carried out to investigate the influence of using Batch Normalization (BN) and the cyclical annealing scheduler alongside

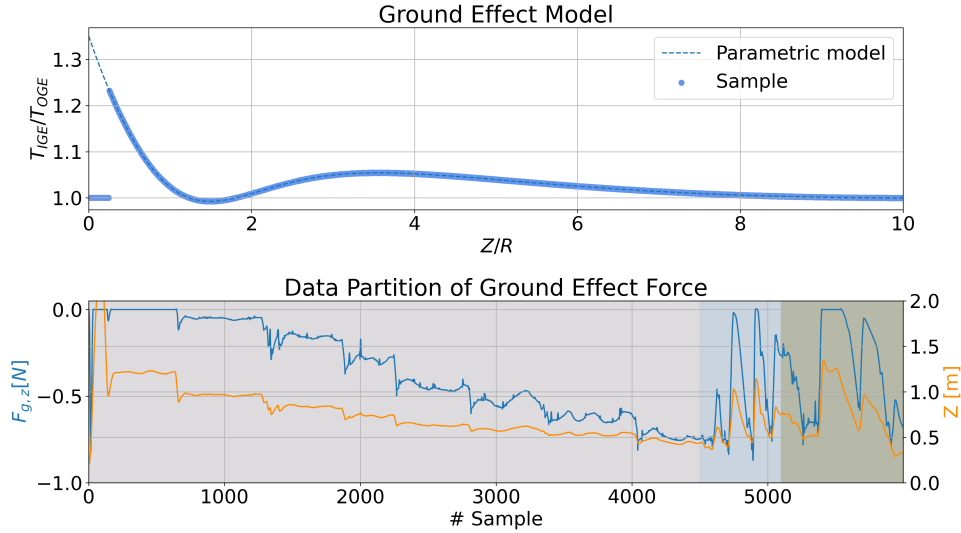


Fig. 3.5 **Simulated flight data with ground effect.** (Top) Parametric model of quadrotor ground effect compared with simulator-collected data samples[†]. (Bottom) Training flight data, showing ground effect force along body z -axis ($F_{g,z}$) and quadrotor altitude (Z), with shaded areas indicating data partitioning (training/validation/test).

[†] The parametric model is derived in [78]. In the top figure, x -axis represents the ratio between altitude above the ground, Z , and propeller radius, R , and y -axis represents the ratio between the lift experienced while hovering In-Ground Effect (IGE) and Out-of-Ground Effect (OGE).

our PINN. A brief overview of all the models considered in our comparison and ablation studies is reported in Table 3.2.

All three test data sets were used for evaluation, taking into account drag, ground effect, and periodic wind. In our assessment trials, all the considered models were evaluated under a single setting with one selected test data set for 20 trials, and the results were averaged over multiple seeds. These averaged results, including both prediction error ($\Delta\mathbf{y} := \mathbf{y} - \hat{\mathbf{y}}$) and the difference in slope (or rotation angle) between prediction and label CCEs, denoted by Δm , are reported in Table 3.3. Note that Δm serves as an indicator of how well the network learns the prior knowledge, in our case, the conservation law of momentum.

Linearized mathematical model as baseline. We first examined the performance of the linearized mathematical model given by (3.2) (referred to as “Model 1” in Table 3.3) compared with a vanilla DNN (“Model 2”) and PINNs (“Model

Table 3.2 **Models for comparison and ablation studies.** We analyzed: (i) model structure (2nd – 4th column), (ii) application of Batch Normalization (BN) (5th – 6th column), (iii) regularization hyperparameter settings (7th – 8th column), and (iv) training data set (9th – 12th column). Bullets (•) symbolizes the highlighted features present within the model.

| Model ID | Structure | | | BN | | Hyperparameter | | Data set | | | |
|----------|------------------|-------------|------|-----|-----|----------------|--------------------|-----------------|-----------------|-----------------|------------------|
| | Linearized model | Vanilla DNN | PINN | w/o | w/ | Constant | Cyclical annealing | \mathcal{D}_1 | \mathcal{D}_2 | \mathcal{D}_3 | Reduced data set |
| 1 | • | ○ | ○ | n/a | n/a | n/a | n/a | • | ○ | ○ | ○ |
| 2 | ○ | • | ○ | • | ○ | n/a | n/a | • | ○ | ○ | ○ |
| 3 | ○ | ○ | • | • | ○ | • | ○ | • | ○ | ○ | ○ |
| 4 | ○ | ○ | ○ | • | ○ | ○ | • | • | ○ | ○ | ○ |
| 5 | ○ | • | ○ | ○ | • | n/a | n/a | • | ○ | ○ | ○ |
| 6 | ○ | ○ | ○ | ○ | • | ○ | • | • | ○ | ○ | ○ |
| 7 | ○ | • | ○ | ○ | • | n/a | n/a | • | ○ | ○ | • |
| 8 | ○ | ○ | • | ○ | • | ○ | • | • | ○ | ○ | • |
| 9 | ○ | • | ○ | ○ | • | n/a | n/a | • | ○ | ○ | • |
| 10 | ○ | ○ | • | ○ | • | ○ | • | • | ○ | ○ | • |
| 11 | ○ | • | ○ | ○ | • | n/a | n/a | ○ | • | ○ | ○ |
| 12 | ○ | ○ | • | ○ | • | ○ | • | ○ | • | ○ | ○ |
| 13 | ○ | • | ○ | ○ | • | n/a | n/a | ○ | ○ | • | ○ |
| 14 | ○ | ○ | • | ○ | • | ○ | • | ○ | ○ | • | ○ |

Table 3.3 **Comparison results of dynamical modeling.** $\Delta \mathbf{y}$ denotes the absolute value of prediction error between prediction $\hat{\mathbf{y}}$ and label \mathbf{y} . Δm_i denotes the absolute difference of slope (or rotation angle) between prediction and label CCEs, where $i = \phi, \theta, \psi$ denotes the three degrees of rotation. $\bar{(\cdot)}$ denotes averaging over multiple seeds and $\sigma(\cdot)$ denotes the corresponding standard deviation. Superscript † and ‡ denote linearized mathematical model (3.2) and vanilla DNN, respectively. The results of comparison and ablation studies are reported in groups, which are separated by horizontal lines, with the best performance indicated in bold.

| Data set | Model | $\overline{\Delta \mathbf{y}}$ | $\sigma(\Delta \mathbf{y})$ | $\overline{\Delta m_\phi}$ | $\sigma(\Delta m)_\phi$ | $\overline{\Delta m_\theta}$ | $\sigma(\Delta m)_\theta$ | $\overline{\Delta m_\psi}$ | $\sigma(\Delta m)_\psi$ |
|-----------------|-----------------|--------------------------------|-----------------------------|----------------------------|-------------------------|------------------------------|---------------------------|----------------------------|-------------------------|
| \mathcal{D}_1 | 1 [†] | 1.023e-1 | n/a | 1.321 | n/a | 6.327 | n/a | 9.200e-2 | n/a |
| | 2 [†] | 3.867e-3 | 2.424e-4 | 22.669 | 4.603 | 25.309 | 4.626 | 1.402 | 0.466 |
| | 3 | 3.775e-3 | 3.525e-4 | 12.507 | 1.666 | 1.706 | 1.498 | 0.118 | 9.611e-2 |
| | 4 | 3.493e-3 | 3.296e-4 | 16.550 | 2.517 | 23.543 | 4.238 | 0.463 | 0.204 |
| | 5 [†] | 2.625e-3 | 1.160e-4 | 26.409 | 4.746 | 28.281 | 5.491 | 0.499 | 0.118 |
| | 6 | 2.480e-3 | 1.030e-4 | 24.337 | 3.555 | 28.604 | 4.618 | 0.273 | 0.0677 |
| | 7 [†] | 4.874e-3 | 6.764e-4 | 33.717 | 4.994 | 22.997 | 4.762 | 0.347 | 0.142 |
| | 8 | 3.793e-3 | 3.444e-4 | 6.550 | 1.546 | 6.472 | 1.478 | 0.136 | 4.227e-2 |
| | 9 [†] | 5.981e-3 | 1.190e-3 | 40.489 | 7.631 | 23.196 | 6.782 | 0.269 | 0.141 |
| | 10 | 4.405e-3 | 6.185e-4 | 12.053 | 1.176 | 1.171 | 1.383 | 0.316 | 3.021e-2 |
| \mathcal{D}_2 | 11 [†] | 2.747e-4 | 4.570e-5 | 4.582 | 1.829 | 1.615 | 0.796 | 57.040 | 11.849 |
| | 12 | 2.580e-4 | 4.290e-5 | 5.662 | 2.031 | 1.724 | 0.631 | 23.399 | 5.521 |
| \mathcal{D}_3 | 13 [†] | 4.771e-3 | 3.661e-4 | 9.526 | 4.686 | 18.904 | 5.065 | 6.954 | 0.997 |
| | 14 | 4.525e-3 | 3.087e-4 | 5.754 | 3.315 | 16.587 | 3.202 | 1.729 | 0.716 |

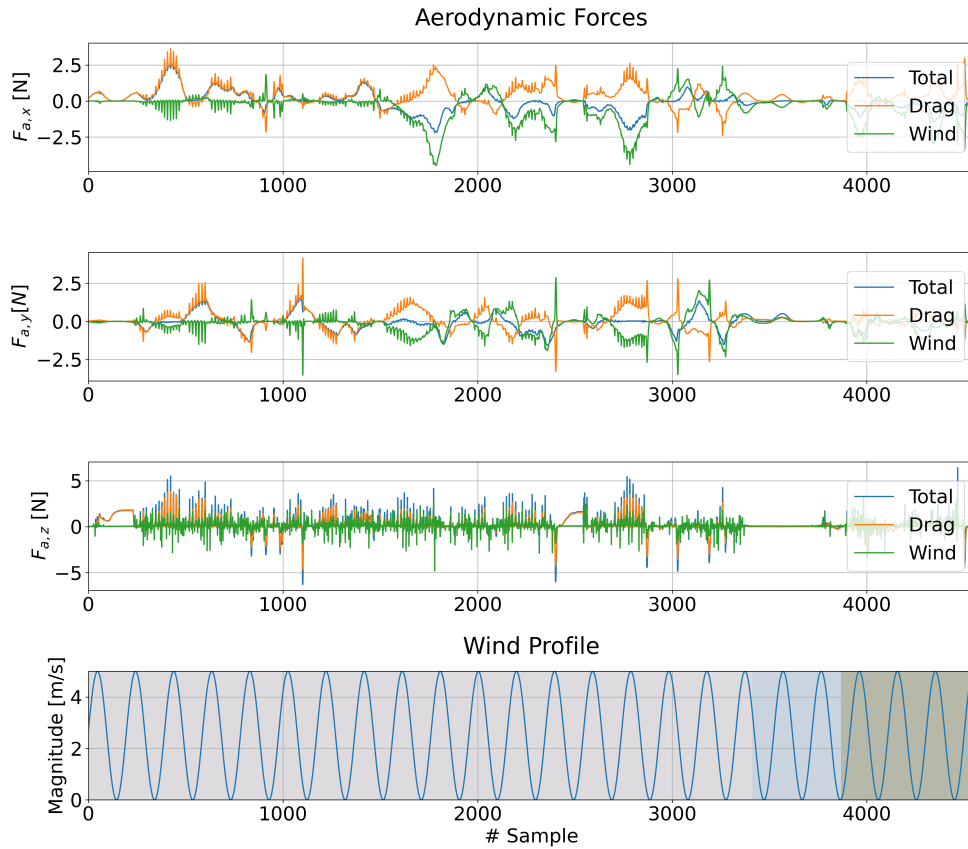


Fig. 3.6 **Simulated flight with periodic wind.** The top three plots depict the composition of aerodynamic forces, including drag and wind. The bottom plot illustrates the profile of periodic wind in the forward direction with a speed of $v_{\text{wind}} = 2.5 \sin(\frac{\pi t}{5}) + 2.5$, overlaid with shaded areas indicating data partitioning (training/validation/test).

3 and 4”) using the \mathcal{D}_1 data set. It was observed that both the vanilla DNN and PINN, regardless of the settings, significantly outperformed the linearized model in terms of prediction error by at least one order of magnitude. This result was expected since drag, included during data collection, becomes non-negligible for agile maneuvers, thereby degrading the performance of the linearized model. Interestingly, the linearized model exhibited relatively small values for Δm , implying that it is “generally” more physically consistent compared with those learning-based models. Despite sound somewhat contradictory, it can be understood because CCE extracts linearity from the data to reveal the global trend, which is more robust to aerodynamic-dominant data samples than MSE. In other words,

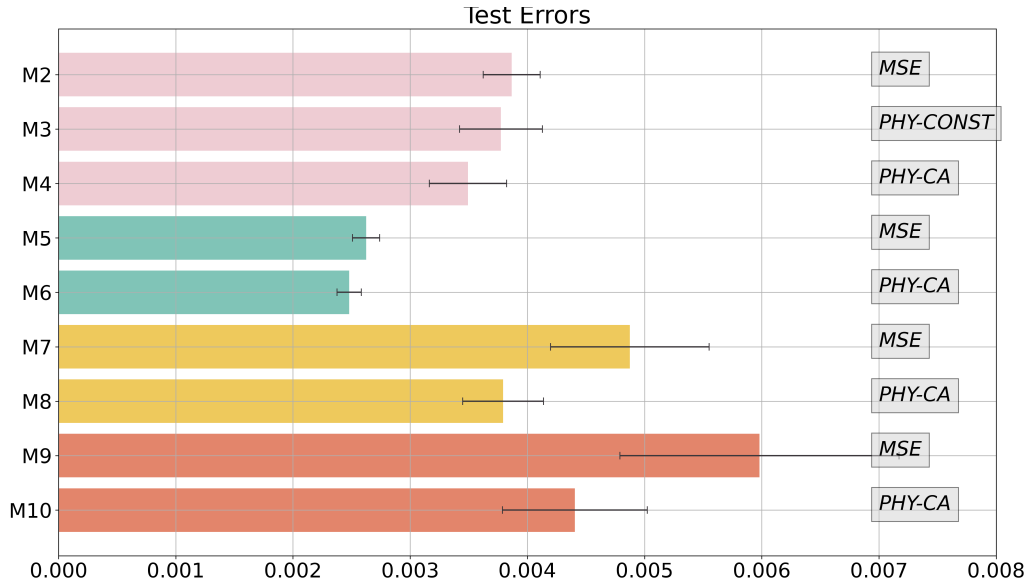


Fig. 3.7 **Test error comparison on \mathcal{D}_1 data set.** The chart shows test errors for models M2 to M10[†]. Light red bars represent models (M2-M4) trained on 60% of data without BN. Green bars represent models (M5-M6) trained on 60% of data with BN. Yellow and orange bars show models (M7-M10) with BN trained on 20% of the data and evaluated on 40% and 80% of the data, respectively. Error bars denote standard deviation.

[†] The abbreviations of the employed loss function for different models are indicated in gray boxes on the right: “MSE” denotes the vanilla DNN, “PHY-CONST” denotes the PINN with a constant hyperparameter ($\lambda_{LM} = \lambda_{max}$), and “PHY-CA” denotes the PINN with cyclical annealing scheduler.

although the linearized model has a larger “bias” in prediction, it achieves a higher “variance” in terms of the slope of CCE.

Effectiveness of PINN and cyclical annealing scheduler. We then found that PINNs (“Model 3 and 4”) exhibited superior performance compared with the vanilla DNN (“Model 2”) in terms of prediction error (for better visualization, see Fig. 3.7). Moreover, the use of cyclical annealing scheduler (“Model 4” with settings from Fig. 3.3) further enhanced the performance compared to the one using a constant value of regularizer (“Model 3”). To provide more insights, we plot 3σ CCE, a post-hoc model interpretability technique, for the predictions generated by both the vanilla DNN and the PINN for a single seed. Despite competitive test error performance (3.322×10^{-3} for the former while 3.061×10^{-3} for the latter), Fig. 3.8 illustrates that the prediction CCE of the vanilla DNN has a larger

deviation of slope (or rotation angle) compared with the target CCE than the PINN does. This implies that the PINN is more capable of learning prior knowledge and maintaining physical consistency in the learned model.

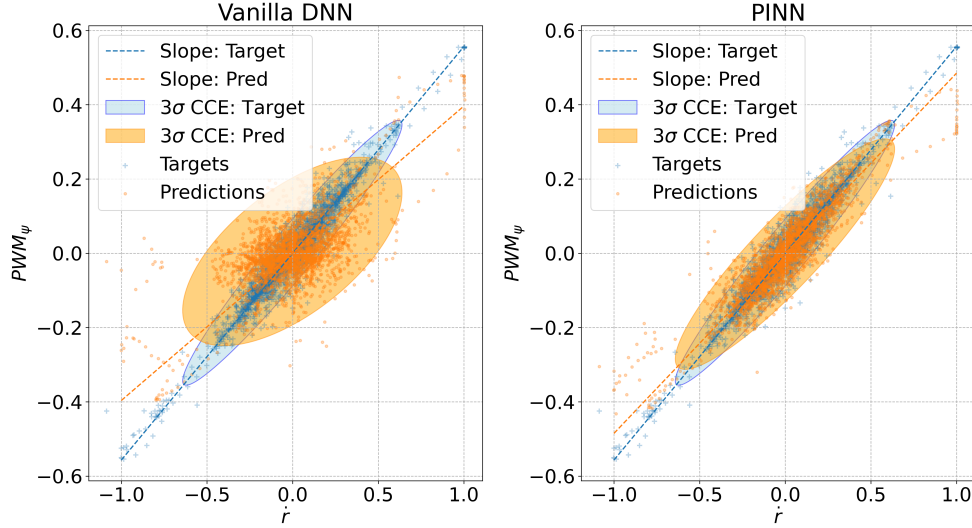


Fig. 3.8 **Comparison of CCE between vanilla DNN and PINN.** Targets and predictions are represented by dots and plus signs, respectively. Shaded ellipses illustrate 3σ CCEs, with dashed lines highlighting the slope.

Benefitting from BN. As an effective practice for enhancing network optimization by eliminating internal covariate shifts within each layer, we investigated if PINN could take advantage of BN (note that “Model 2 to 4” do not use BN). From Table 3.3 or Fig. 3.7, it can be seen that we achieved the best model (“Model 6”) for \mathcal{D}_1 data set using PINN with both cyclical annealing scheduler and BN. It is worth noting that the vanilla DNN also reached competitive performance with BN (“Model 5”), which we believe both models in this scenario are nearing performance saturation due to the large data set and the powerful learning potential provided by DNN and practical techniques. However, as we will see next, PINN establishes a noticeable superiority over vanilla DNN when facing relatively smaller data sets.

Performance on reduced data set. To assess the advantage of the PINN in terms of generalization capability on unseen data enriched by informed physics, we re-trained both the vanilla DNN and the PINN on a reduced data comprising only 20% (approximately $\sim 3k$ samples) of \mathcal{D}_1 , as opposed to 60% (approximately

~ 9 k samples). In this circumstance, PINNs (“Model 8 and 10”) demonstrated notable performance superiority over vanilla DNNs (“Model 7 and 9”) in terms of both prediction error and physical consistency. However, it is worth noting that both networks experienced performance degradation compared to their counterparts trained on the original (full) data set. This outcome underscores the better generalization capability of PINNs in the small data regime. Our proposed method thus offers an effective means to training DNN-based controllers with small data sets while maintaining acceptable generalization capability – an aspect that is critical in numerous robotics and control applications [79]. Additionally, it lays the groundwork for future research directions such as online modeling using real flight.

Evaluation on data sets with ground effect and periodic wind. Lastly, we assessed the performance of the vanilla DNN and PINN on \mathcal{D}_2 (with added ground effect) and \mathcal{D}_3 (with periodic wind). The network structures remained the same, except for the addition of altitude as an extra input feature for both networks trained with \mathcal{D}_2 , as altitude is known to be critical for ground effect [78, 80]. From Fig. 3.9 and Table 3.3, it is straightforward to observe that the PINN demonstrates lower prediction error on both data sets and exhibits higher physical consistency in terms of CCEs. Therefore, despite being guided by the conservation law of momentum instead of direct knowledge on aerodynamics, the resulting PINN still shows better generalization capability compared with the vanilla DNN, even in the presence of various or combined aerodynamic effects.

3.4.4 Computational Complexity

Compared to the vanilla DNN, the additional computational complexity of our proposed PINN mainly arises from the inclusion of the local monotonicity loss (Algorithm 1), which involves $\tanh(\cdot)$ and other basic arithmetic operators. Leveraging automatic differentiation by PyTorch (e.g., `torch.autograd`), back-propagation using the chain rule of known gradients can be easily performed, as Algorithm 1 is implemented entirely using tensors.

In our training trials, conducted over 20 runs with random seeds, using a training data set of size ~ 12 k (~ 9 k for training, ~ 3 k for validation) and running for 300 epochs, training a vanilla DNN (“Model 2” in Table 3.3) took ~ 234 sec

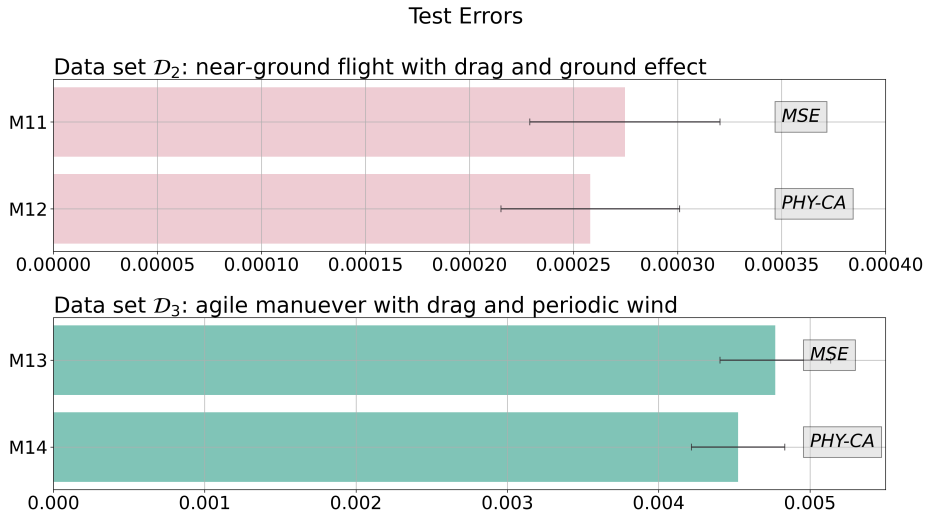


Fig. 3.9 **Test error comparison on \mathcal{D}_2 and \mathcal{D}_3 data sets.** All results are averaged over multiple seeds, with error bars representing the standard deviation.

(~ 3.9 min), while training a proposed PINN (“Model 4” in Table 3.3) took ~ 373 sec (~ 6.2 min) using CUDA on a laptop with an AMD Ryzen 7 5800h and NVIDIA GeForce RTX 3060. This increase in training time is partially offset by the fact that PINNs generally achieve better performance to vanilla DNNs in small data regimes [30]. While online implementation may still be impractical for both networks, their usage is generally acceptable for non-time-critical situations where offline training is feasible.

3.5 Conclusions

Learning-based approaches such as DNNs have garnered unprecedented attention for modeling and control for robotics in recent years. Nonetheless, their black-box nature raises concerns about interpretability, particularly in safety-critical robotic applications. In this chapter, we address this challenge by introducing a PINN for learning quadrotor dynamics, which integrates prior domain knowledge into a DNN architecture. We elaborate on how to seamlessly embed the conservation law of momentum into the training loss function using a cyclical annealing scheduler, following an unstructured learning strategy. Additionally, we adopt CCEs as a post-hoc model interpretability visualization tool to evaluate and understand the behavior of the learned model, thus enhancing its trustworthiness.

To facilitate data collection, we set up a visual and physical simulator based on AirSim, incorporating a customized implementation of quadrotor ground effect. We provide user-level APIs in both C++ and Python, which are publicly available for further research in the UAV community. Through extensive simulation campaigns over multiple seeds, our proposed PINN consistently demonstrates superior generalization capability compared with both linearized mathematical model and vanilla DNN, evaluated on complete and reduced data sets including various aerodynamic effects. We also conduct ablation studies and analyze the additional computational complexity of our proposed approach compared with vanilla DNNs.

In our ongoing efforts, we aspire to delve deeper into the potential of PINN in online learning, leveraging real flight data for enhanced model refinement and adaptation. Integrating the proposed PINN as a control-oriented model into the controller design is another potential research direction, which allow us to thoroughly assess its impact on closed-loop performance of the system under control, potentially paving the way for more robust and efficient UAV control systems. Last but not least, we are eager to explore new paradigms of PINN that seamlessly combine structured and unstructured learning methodologies, harnessing their complementary strengths to advance the development of UAV modeling and control even further.

Chapter 4

Robust Adaptive Controller Design for Parametric Uncertainties

4.1 Introduction

The design of controllers for quadrotors toward either trajectory tracking or attitude stabilization is non-trivial due to the inherent nonlinearities and strong coupling properties in quadrotor dynamics, let alone uncertainties and disturbances, which are ubiquitous in real-world flight operations [10]. The challenge further escalates when the quadrotor is assigned to perform aggressive maneuvers, particularly for specific tasks such as aerial acrobatics or rapid changes in direction [11, 12].

For decades, a number of nonlinear control methodologies such as BSC [81], SMC [82], MPC [83], and Disturbance-Observer-Based Control (DOBC) [84] have been studied to guarantee stability and performance of systems under control. Most of these designs were based on *local coordinates* such as Euler angles and quaternions to describe the rotational kinematics and dynamics of quadrotors. Despite merits, such as quaternions being the minimal representation without singular points[85], and Euler angles providing an intuitive representation for 3-D rotations, local coordinates come with inherent limitations that hinder practical applications, especially for aggressive maneuvers [86]. Specifically, Gimbal lock (or singularity), arisen from Euler angles, results in the loss of one degree of freedom in 3-D orientation systems [87]; unwinding phenomena with quaternion

representations can cause rotations through large angles before stabilizing at the desired attitude, even with a close initial state [88]. These issues are particularly undesirable in aerospace applications, as they may induce catastrophic instability.

Geometric control techniques have been lately proven effective in tackling these issues, especially in the execution of aggressive maneuvers; see, e.g., [89–95]. Instead of using local coordinates, geometric controllers are designed directly on the *special orthogonal group* $SO(3)$, which is a nonlinear manifold on which the configuration space of attitude dynamics evolves. For example, one of the first applications of geometric control to quadrotors was [89], where a controller was proposed on the nonlinear configuration Lie group. As such, through the intrinsic characterization of the geometric properties of nonlinear manifolds, singularities and ambiguities associated with local coordinates can be circumvented, thereby enabling the execution of large-angle rotational maneuvers.

While notable advancements have been made in geometric control for quadrotors, crucial gaps persist, particularly in addressing uncertainties and disturbances as well as in selecting configuration error functions. Like other model-based control approaches, the effectiveness of the control law derived from $SO(3)$ hinges on the precise knowledge of system parameters. Additionally, the exploration of configuration error functions for control synthesis remains an open topic in geometric control, sometimes chosen without careful consideration. While some studies draw on the results in [89] (see, e.g., [4, 6, 91, 93, 96]), it was noted that the chosen configuration error vector can lead to degraded performance with increasing initial attitude error, prompting a new error function proportional to the rotation angle for consistent tracking performance [90]. Another alternative for the configuration error function is the logarithmic map of $SO(3)$ [35]. Thanks to the inherent ability to transform geodesics on $SO(3)$ into straight lines in its Lie algebra $\mathfrak{so}(3)$, the logarithmic map establishes a proportional relationship between the magnitude of the attitude error vector and the rotation angle, with a higher constant of proportionality than that used in [90], hence facilitating accelerated convergence rates for tracking errors [95, 97]. However, existing studies in this direction focus only on the nominal case, neglecting uncertainties and disturbances, which could potentially compromise controller performance or even lead to instability.

In this chapter, we present a novel robust adaptive geometric controller for quadrotor aggressive maneuvers in the presence of parametric uncertainties, namely quadrotor mass and inertia matrix. Specifically, the control problem on $SE(3)$ is decomposed into two distinct sub-tasks, i.e., position tracking on \mathbb{R}^3 and attitude tracking on $SO(3)$ [98]. As illustrated in Fig. 4.1, our proposed method utilizes BSC for thrust determination and the logarithmic map of $SO(3)$ to represent the attitude error for torque determination. In contrast to the existing approaches in [94] and [95] that employ the logarithmic map for nominal conditions, we present a fully nonlinear control synthesis on $SE(3)$ capable of addressing uncertainties, thereby extending its applicability beyond linearized and uncertainty-free scenarios. Two adaptive laws are derived through Lyapunov analysis, aimed at dynamically compensating for uncertainties in the mass and inertia matrix. Along with the use of projection operators [18], we show asymptotically stable tracking on $SE(3)$ and the boundedness of all signals in the closed-loop system even when there exist nonparametric uncertainties such as sensor noise [99]. The main outcomes from this chapter are summarized as follows:

- (i) We developed a novel adaptive geometric controller, utilizing the logarithmic map of $SO(3)$ for attitude tracking and the BSC for position tracking, for executing aggressive maneuvers without requiring precise knowledge of the quadrotor mass and inertia matrix.
- (ii) We proved almost globally asymptotically stable tracking and the boundedness of all signals in the closed-loop system through Lyapunov analysis.
- (iii) We enhanced the robustness of adaptive laws by applying projection operators to prevent system from parameter drift in the presence of nonparametric uncertainties.

4.2 Related Work

Efforts in the field of geometric control for quadrotors can be generally classified into two classes: one focuses on addressing uncertainties and disturbances, while the other centers around the selection of configuration error functions. A comparative analysis of prior studies is summarized in Table 4.1.

| Rotational error | Citation (Year) | Robustness and adaptability | Results |
|--|-----------------|---|----------|
| | [89] (2010) | n/a | Num |
| | [91] (2013) | Integral control for handling constant disturbances | Num, Exp |
| | [93] (2014) | Adaptive laws for unmodeled dynamics | Num, Exp |
| $\frac{1}{2} \text{tr}(\mathbf{I} - \mathbf{R}_b^d)$ | [4] (2019) | Control of a quadrotor-pulley-load system | Num |
| | [6] (2021) | Control of quadrotor load transporting via elastic cables | Num |
| | [96] (2018) | Control of quadrotor load transporting via rigid cables | Num |
| $2 - \sqrt{1 + \text{tr}(\mathbf{R}_b^d)}$ | [90] (2010) | Stabilization under an unknown inertia matrix | Num |
| $\frac{1}{2} \text{tr}[\mathbf{G}(\mathbf{I} - \mathbf{R}_b^d)], \mathbf{G} > 0$ | [92] (2013) | A robust adaptive law for the inertia matrix and bounded disturbances | Exp |
| | [100] (2020) | A robust adaptive law for center of gravity | Num |
| $\frac{1}{2} \text{tr}[\mathbf{K}(\mathbf{I} - \mathbf{R}\mathbf{R}_d^T)], \mathbf{K} > 0$ | [101] (2017) | Control of a tiltrotor | Num |
| | [94] (2015) | n/a | Exp |
| | [102] (2017) | n/a | Sim |
| $\ \log(\mathbf{R}_b^d)^\vee\ $ | [95] (2021) | n/a | Sim, Exp |
| | [97] (2022) | n/a | Num |
| Hybrid error | [103] (2016) | n/a | Sim, Exp |

Table 4.1 **Comparative analysis of selected literature on geometric control with applications to quadrotors.** We compared the error function for rotational configuration (2nd column), control methods for handling uncertainties (3rd column), and obtained results (4th column; *Num*: numerical, *Sim*: simulation, *Exp*: experimental).

One line of research aims at extending the results in [89] by taking into account uncertainties and disturbances. For example, [91] adopted integral control terms to guarantee almost global asymptotic stability when there exist fixed disturbances in both translational and rotational dynamics; [92] proposed a robust adaptive tracking controller without the knowledge of the inertia matrix and guaranteed the boundedness of tracking errors in the presence of unstructured disturbances; [93] developed adaptive control laws that guarantees asymptotic convergence of tracking errors for modeling error and uncertainties in dynamical equations; and [100] proposed an adaptive law for the geometric controller to estimate the center of gravity of the quadrotor, which differs from the geometric center. Other applications to tiltrotor [101] and load transportation quadrotor [4, 6, 96] can also be considered to fall into this class.

Another avenue of research contributes to exploring configuration error functions, an ongoing subject in geometric control. These functions continue to be extensively examined and, at times, selected without meticulous consideration [90]. For example, the configuration error functions chosen in [89] may result in diminished performance as the initial attitude error tends to grow. To counter this issue, several alternatives have been proposed to ensure consistent tracking performance by constructing a proportional relationship to the rotation angle. [90] adopted a revised version of [89] for the stabilization of a quadrotor subject to unknown inertia matrix; [97] designed a quadratic cost function in the Lie algebra through its gradient for the control on Lie groups thanks to their symmetry structure such that faster error convergence can be achieved; [94] proposed a PID controller directly on $\mathfrak{so}(3)$, the Lie algebra associated with $SO(3)$, with rotation modeled using exponential coordinates to perform complex acrobatic maneuvers; [102] employed a homogeneous method to address the finite-time control problem of a quadrotor on a Lie group, utilizing exponential and logarithmic maps; [95] presented a geometric tracking controller based on the logarithmic map of $SO(3)$, achieving faster convergence speed of tracking error; [104] studied the performance of several attitude error vectors for the control of a quadrotor subject to rotor failures; and [103] proposed a hybrid attitude controller which guarantees global exponential stability and overcomes the topological obstacles of global control on $SO(3)$ [86]. Nonetheless, almost none of these studies incorporate robust or adaptive designs to handle uncertainties and disturbances.

4.3 Problem Formulation

Our control objective is to achieve asymptotically stable tracking for a quadrotor that satisfies (2.1) of a given, possibly aggressive, reference trajectory $\xi_{\text{ref}}(t)$ in the presence of uncertain mass and inertia matrix while ensuring that all signals in the closed-loop system remain bounded.

To this end, we present a robust adaptive geometric tracking controller on $\text{SE}(3)$ as illustrated in Fig. 4.1, which consists of: (i) a backstepping controller for thrust determination and providing attitude reference (in terms of body z -axis) to the inner-loop attitude controller, (ii) a geometric controller based on the logarithmic map of $\text{SO}(3)$ to generate commanded torques for attitude tracking, and (iii) two adaptive laws for handling uncertainties in mass and inertia matrix, robustified by projection operators. For the control synthesis discussed in Section 4.4 and Section 4.5, we make the following assumptions necessary for the subsequent control synthesis and simplify our notation by dropping the time-dependent components.

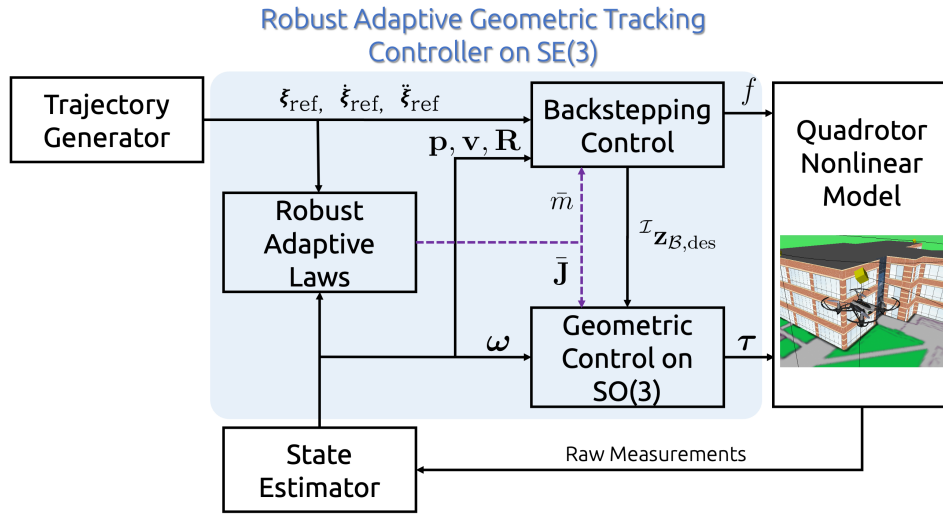


Fig. 4.1 Control scheme of the proposed robust adaptive geometric tracking controller on $\text{SE}(3)$.

Assumption 4.3.1. *The reference trajectory is given as $\xi_{\text{ref}}(t) = [\mathbf{p}_{\text{ref}}^T(t) \psi_{\text{ref}}(t)]^T \in \mathbb{R}^4$ (i.e., position and yaw angle) using the differential flatness property of quadrotors [105]. Moreover, it satisfies $\xi_{\text{ref}}(t) \in \mathcal{C}^3$, i.e., the derivatives up to $\xi_{\text{ref}}^{(3)}(t)$ exist and are continuous.*

Assumption 4.3.2. *The derivatives up to the 3rd order of the reference trajectory are bounded, i.e., $\xi_{ref}^{(k)}(t) \in \mathcal{L}_\infty$ ($k = 0, 1, 2, 3$).*

Assumption 4.3.3. *The quadrotor mass m and inertia matrix \mathbf{J} are uncertain but slowly time-varying, with their variations bounded, i.e., $\dot{m}, \dot{\mathbf{J}} \in \mathcal{L}_\infty$.*

4.4 Control Synthesis for Position Tracking

In this section, we present the primary results for achieving globally asymptotically stable tracking of position reference in the presence of uncertain mass through backstepping formulation. Additionally, to enhance the robustness of the adaptive law under nonparametric uncertainties such as sensor noise, a projection operator is applied to guarantee the boundedness of estimated parameters and avoid sudden instability.

Theorem 4.4.1 (Adaptive Tracking Control for Translational Dynamics). *Consider the translational dynamics given by (2.1a), (2.1b), and for a given tracking command $\mathbf{p}_{ref} \in \mathcal{C}^3$, we define a control input (4.1) and an adaptive law for mass estimation (4.2) as*

$$\mathbf{f} = \bar{m}(-\mathbf{K}_v \tilde{\mathbf{v}} - \tilde{\mathbf{p}} - \mathbf{g}\mathbf{e}_3 + \dot{\mathbf{v}}_r), \quad (4.1)$$

$$\dot{\hat{m}} = \gamma_m \tilde{\mathbf{v}}^\top (\mathbf{K}_v \tilde{\mathbf{v}} + \tilde{\mathbf{p}} + \mathbf{g}\mathbf{e}_3 - \dot{\mathbf{v}}_r), \quad (4.2)$$

$$\mathbf{v}_r = \dot{\mathbf{p}}_{ref} - \mathbf{K}_p \tilde{\mathbf{p}}, \quad (4.3)$$

where $\gamma_m \in \mathbb{R}$ is a positive constant, and $\mathbf{K}_p, \mathbf{K}_v \in \mathbb{R}^{3 \times 3}$ are positive definite gain matrices. The estimate of m is given by \hat{m}^1 , with the estimation error defined as $\tilde{m} := m - \hat{m}$. The virtual control input $\mathbf{v}_r \in \mathbb{R}^3$ in (4.3) is derived from backstepping formulation. Then, the zero equilibrium of translational tracking errors $(\tilde{\mathbf{r}}, \tilde{\mathbf{v}}) = (\mathbf{0}, \mathbf{0})$ is globally asymptotically stable, and furthermore the mass estimation error \tilde{m} is uniformly bounded.

Proof. *To show that Theorem 4.4.1 holds, we start with designing the certainty-equivalent controller by assuming perfect knowledge on quadrotor mass (m). Subsequently, we relax this assumption, accounting for the unknown mass, resulting in the final control law with adaptation.*

¹Throughout Chapter 4, we opt for the notation $(\hat{\cdot})$ to represent estimated variables, as opposed to the conventional $(\hat{\cdot})$. This choice is made to prevent confusion with the hat operator, defined as $(\cdot)^\wedge : \mathbb{R}^3 \rightarrow \mathfrak{so}(3)$.

Defining the position tracking error as

$$\tilde{\mathbf{p}} := \mathbf{p} - \mathbf{p}_{ref}, \quad (4.4)$$

it is straightforward to design a virtual control input (4.3) such that if \mathbf{v} perfectly tracks \mathbf{v}_r . Hence, the following two Lyapunov conditions are met:

$$\begin{aligned} V_1(\tilde{\mathbf{p}}) &= \frac{1}{2} \tilde{\mathbf{p}}^\top \tilde{\mathbf{p}} > 0, \\ \dot{V}_1(\tilde{\mathbf{p}}) &= \tilde{\mathbf{p}}^\top \dot{\tilde{\mathbf{p}}} = \tilde{\mathbf{p}}^\top (\mathbf{v} - \dot{\mathbf{p}}_{ref}) < 0. \end{aligned}$$

Next, define the velocity tracking error as

$$\tilde{\mathbf{v}} := \mathbf{v} - \mathbf{v}_r, \quad (4.5)$$

and construct the composite Lyapunov candidate function

$$V_2(\tilde{\mathbf{p}}, \tilde{\mathbf{v}}) = V_1(\tilde{\mathbf{p}}) + \frac{1}{2} \tilde{\mathbf{v}}^\top \tilde{\mathbf{v}} > 0.$$

We can easily derive the desired (nominal) thrust vector

$$\mathbf{f} = m(-\mathbf{K}_v \tilde{\mathbf{v}} - \tilde{\mathbf{p}} - g\mathbf{e}_3 + \dot{\mathbf{v}}_r) \quad (4.6)$$

using backstepping technique [106] along with translation dynamics (2.1b), such that

$$\begin{aligned} \dot{V}_2(\tilde{\mathbf{p}}, \tilde{\mathbf{v}}) &= \tilde{\mathbf{p}}^\top \dot{\tilde{\mathbf{p}}} + \tilde{\mathbf{v}}^\top \dot{\tilde{\mathbf{v}}} \\ &= \tilde{\mathbf{p}}^\top (\tilde{\mathbf{v}} - \mathbf{K}_p \tilde{\mathbf{p}}) + \tilde{\mathbf{v}}^\top (g\mathbf{e}_3 - \frac{1}{m} f \mathbf{R} \mathbf{e}_3 - \dot{\mathbf{v}}_r) \\ &= -\tilde{\mathbf{p}}^\top \mathbf{K}_p \tilde{\mathbf{p}} + \tilde{\mathbf{v}}^\top (\tilde{\mathbf{p}} + g\mathbf{e}_3 - \frac{1}{m} f \mathbf{R} \mathbf{e}_3 - \dot{\mathbf{v}}_r) \\ &= -\tilde{\mathbf{p}}^\top \mathbf{K}_p \tilde{\mathbf{p}} - \tilde{\mathbf{v}}^\top \mathbf{K}_v \tilde{\mathbf{v}} < 0. \end{aligned}$$

That being said, under nominal conditions (i.e., m is perfectly known), if we choose the desired thrust and the desired body z -axis, which shall be opposite to the thrust

direction, as

$$f = \|\mathbf{f}\|, \quad (4.7)$$

$$\mathbf{z}_{\mathcal{B},des} = \frac{\mathbf{f}}{\|\mathbf{f}\|} \in \mathbb{S}^2, \quad (4.8)$$

where \mathcal{B} denotes quantities expressed in the body frame. Then, $\tilde{\mathbf{p}}, \tilde{\mathbf{v}}$ will globally asymptotically go to zero for $t \rightarrow \infty$.

Now, we assume m to be unknown and rewrite the certainty equivalent control law (4.6) by replacing m with its estimate \tilde{m} , yielding (4.1). To derive the adaptive law for \tilde{m} , we construct a composite Lyapunov candidate function and take the time derivative as

$$V_3(\tilde{\mathbf{p}}, \tilde{\mathbf{v}}, \tilde{m}) = V_2(\tilde{\mathbf{p}}, \tilde{\mathbf{v}}) + \frac{1}{2\gamma_m m} \tilde{m}^2 > 0.$$

Hence,

$$\begin{aligned} \dot{V}_3(\tilde{\mathbf{p}}, \tilde{\mathbf{v}}, \tilde{m}) &= \tilde{\mathbf{p}}^\top \dot{\tilde{\mathbf{p}}} + \tilde{\mathbf{v}}^\top \dot{\tilde{\mathbf{v}}} + \frac{1}{\gamma_m m} \tilde{m} \dot{\tilde{m}} \\ &\stackrel{(4.1)}{=} -\tilde{\mathbf{p}}^\top \mathbf{K}_p \tilde{\mathbf{p}} + \tilde{\mathbf{v}}^\top \left(\tilde{\mathbf{p}} + g\mathbf{e}_3 + \frac{\tilde{m}}{m} (-\mathbf{K}_v \tilde{\mathbf{v}} - \tilde{\mathbf{p}} - g\mathbf{e}_3 + \dot{\mathbf{v}}_r) - \dot{\mathbf{v}}_r \right) + \frac{1}{\gamma_m m} \tilde{m} \dot{\tilde{m}}. \end{aligned} \quad (4.9)$$

Recalling Assumption 4.3.3, the following equalities hold:

$$\begin{aligned} \frac{\tilde{m}}{m} &= 1 - \frac{\tilde{m}}{m}, \\ \dot{\tilde{m}} &= -\dot{\tilde{m}}, \end{aligned}$$

which allow us to further organize (4.9) as

$$\dot{V}_3(\tilde{\mathbf{p}}, \tilde{\mathbf{v}}, \tilde{m}) = -\tilde{\mathbf{p}}^\top \mathbf{K}_p \tilde{\mathbf{p}} - \tilde{\mathbf{v}}^\top \mathbf{K}_v \tilde{\mathbf{v}} + \frac{\tilde{m}}{\gamma_m m} \left(-\dot{\tilde{m}} + \gamma_m \tilde{\mathbf{v}}^\top (\mathbf{K}_v \tilde{\mathbf{v}} + \tilde{\mathbf{p}} + g\mathbf{e}_3 - \dot{\mathbf{v}}_r) \right). \quad (4.10)$$

By selecting the adaptive law for mass estimation as in (4.2), it can be proved that $\dot{V}_3(\tilde{\mathbf{p}}, \tilde{\mathbf{v}}, \tilde{m}) \leq 0$, which implies $\tilde{\mathbf{p}}, \tilde{\mathbf{v}}, \tilde{m} \in \mathcal{L}_\infty$, but with no guarantees on asymptotic stability. Nonetheless, we have $\dot{\tilde{\mathbf{p}}} = \tilde{\mathbf{v}} - \mathbf{K}_p \tilde{\mathbf{p}} \in \mathcal{L}_\infty$, $\tilde{m} \in \mathcal{L}_\infty$ from Assumption 4.3.3, and $\dot{\mathbf{v}}_r \in \mathcal{L}_\infty$ from (4.3) together with Assumption 4.3.2. Hence, it can be proved that $\dot{\mathbf{v}} = g\mathbf{e}_3 + \frac{1}{m} \tilde{m} (-\mathbf{K}_v \tilde{\mathbf{v}} - \tilde{\mathbf{p}} - g\mathbf{e}_3 + \dot{\mathbf{v}}_r) \in \mathcal{L}_\infty$. From (4.5), we get $\dot{\tilde{\mathbf{v}}} = \dot{\mathbf{v}} - \ddot{\mathbf{p}}_{ref} + \mathbf{K}_p \dot{\tilde{\mathbf{p}}} \in \mathcal{L}_\infty$. Therefore, $\ddot{V}_3 = -2\tilde{\mathbf{p}}^\top \mathbf{K}_p \dot{\tilde{\mathbf{p}}} - 2\tilde{\mathbf{v}}^\top \mathbf{K}_v \dot{\tilde{\mathbf{v}}} \in \mathcal{L}_\infty$, implying that \dot{V}_3 is uniformly continu-

ous. By invoking Barbalat's Lemma, it can be concluded that $\lim_{t \rightarrow \infty} \dot{V}_3(t) = 0$, i.e., $\tilde{\mathbf{p}}, \tilde{\mathbf{v}} \rightarrow 0$ for $t \rightarrow \infty$. \square

Proposition 4.4.1 (Robustification of Adaptive Law using Projection Operator). *To improve the robustness of estimation algorithms (such that parameter drift can be avoided when sensor noise exists in the system), the adaptive law of mass estimation (4.2) is adjusted as*

$$\dot{\bar{m}} = \text{Proj}_{\gamma_m}(\bar{m}, \tilde{\mathbf{v}}^\top (\tilde{\mathbf{p}} + \mathbf{g}\mathbf{e}_3 - \dot{\mathbf{v}}_r + \mathbf{K}_v \tilde{\mathbf{v}}), h),$$

where $\text{Proj}_{(\cdot)}$ denotes Γ -projection [18], and the continuously differentiable convex function $h: \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ is chosen as

$$h(\bar{m}, m_0, \epsilon_m) = \frac{\bar{m}^2 - m_0^2}{2\epsilon_m m_0 + \epsilon_m^2},$$

where m_0 and ϵ_m are two constant scalar quantities. If $\bar{m}(t=0) \in \Omega_m = \{\bar{m} \in \mathbb{R} : h(\bar{m}, m_0, \epsilon_m) \leq 1\}$, then we can conclude $\bar{m}(t) \in \Omega_m$, or equivalently, $\|\bar{m}(t)\| \leq m_0 + \epsilon_m, \forall t \geq 0$, guaranteeing the boundedness of the estimated parameter.

Proof. The readers interested in the proof of projection operator are kindly referred to [18, 107] for details. \square

4.5 Control Synthesis for Attitude Tracking

In this section, we first introduce the attitude and angular velocity error dynamics based on the logarithmic map of $\text{SO}(3)$. Then, we present the main results on achieving almost globally asymptotically stable tracking of attitude reference in the presence of uncertain inertia matrix through geometric control synthesis. Lastly, in the same vein as in Section 4.4, we also ensure the boundedness of estimation parameters by projection operator even when nonparametric uncertainties exist.

The desired orientation for attitude control can be expressed in terms of rotation matrix, which can be derived by using heading angle reference and the

desired body z -axis (4.8) from the backstepping controller as

$$\begin{aligned} \mathbf{y}_c &= [-\sin(\psi_{\text{ref}}), \cos(\psi_{\text{ref}}), 0]^\top, \\ \mathbf{x}_{\mathcal{B},\text{des}} &= \mathbf{y}_c^\wedge \mathbf{z}_{\mathcal{B},\text{des}} / \|\mathbf{y}_c^\wedge \mathbf{z}_{\mathcal{B},\text{des}}\|, \\ \mathbf{y}_{\mathcal{B},\text{des}} &= \mathbf{z}_{\mathcal{B},\text{des}}^\wedge \mathbf{x}_{\mathcal{B},\text{des}}, \\ \mathbf{R}_d^i &= [\mathbf{x}_{\mathcal{B},\text{des}}, \mathbf{y}_{\mathcal{B},\text{des}}, \mathbf{z}_{\mathcal{B},\text{des}}], \end{aligned}$$

where \mathbf{R}_d^i represents the rotation matrix from desired frame (denoted by \mathcal{D}) to inertial frame (denoted by \mathcal{I}). Such formulation is similar to that in [105], with a slight difference due to the rotation convention used (here we use “3-2-1”, whereas in [105] they use “3-1-2”). Since $\mathbf{R}_d^i \in \text{SO}(3)$, we can therefore write

$$\dot{\mathbf{R}}_d^i = \mathbf{R}_d^i \mathcal{D} \boldsymbol{\omega}_{d/i}^\wedge,$$

where the subscripts describe the relation of the rotational motion, e.g., $\boldsymbol{\omega}_{d/i}$ indicates the rotational motion of desired frame with respect to inertial frame, and the left superscripts indicate the reference frames.

Next, we introduce the definition of attitude and angular velocity error vector based on the logarithmic map of $\text{SO}(3)$.

Definition 4.5.1 (Attitude and Angular Velocity Error Vector). *For a given tracking command $(\mathbf{R}_d^i, \mathcal{D} \boldsymbol{\omega}_{d/i})$, and current attitude and angular velocity $(\mathbf{R}_b^i, \mathcal{B} \boldsymbol{\omega}_{b/i})$, we define an attitude error vector $\tilde{\mathbf{r}} : \text{SO}(3) \times \text{SO}(3) \rightarrow \mathbb{R}^3$ and an angular velocity error vector $\tilde{\boldsymbol{\omega}} : \text{SO}(3) \times \mathbb{R}^3 \times \text{SO}(3) \times \mathbb{R}^3 \rightarrow \mathbb{R}^3$ as*

$$\tilde{\mathbf{r}}(\mathbf{R}_b^i, \mathbf{R}_d^i) := \log((\mathbf{R}_b^i)^\top \mathbf{R}_d^i)^\vee = \log(\mathbf{R}_d^b)^\vee, \quad (4.11)$$

$$\tilde{\boldsymbol{\omega}}(\mathbf{R}_b^i, \mathcal{B} \boldsymbol{\omega}_{b/i}, \mathbf{R}_d^i, \mathcal{D} \boldsymbol{\omega}_{d/i}) := \mathcal{B} \boldsymbol{\omega}_{d/b} = \mathbf{R}_d^b \mathcal{D} \boldsymbol{\omega}_{d/i} - \mathcal{B} \boldsymbol{\omega}_{b/i}. \quad (4.12)$$

By Definition 4.5.1, we can derive the error dynamics of attitude and angular velocity as stated in Proposition 4.5.1. Since we are going to formulate our main results in the body frame, $\boldsymbol{\omega}$'s in the sequel are by default expressed in the body frame, hence left superscripts \mathcal{B} are omitted unless ambiguities must be resolved.

Proposition 4.5.1 (Attitude and Angular Velocity Error Dynamics). *The dynamics of $\tilde{\mathbf{r}}$ and $\tilde{\boldsymbol{\omega}}$ satisfy*

$$\dot{\tilde{\mathbf{r}}} = \mathbf{J}_l(\tilde{\mathbf{r}})^{-1} \tilde{\boldsymbol{\omega}}, \quad (4.13)$$

$$\mathbf{J} \dot{\tilde{\boldsymbol{\omega}}} = \mathbf{J} \dot{\boldsymbol{\omega}}_{d/i} - \mathbf{J} \boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \mathbf{J} \boldsymbol{\omega}_{b/i} - \boldsymbol{\tau}, \quad (4.14)$$

where $\mathbf{J}_l(\cdot)^{-1}$ is the inverse of the left Jacobian as in (2.11).

Proof. *The attitude error dynamics can be derived from (2.5) by using the time derivative of matrix exponential*

$$\begin{aligned} \dot{\mathbf{R}}_d^b &= \frac{d}{dt} \exp(\tilde{\mathbf{r}}^\wedge) \\ &= \int_0^1 \exp(\alpha \tilde{\mathbf{r}}^\wedge) \dot{\tilde{\mathbf{r}}}^\wedge \exp((1-\alpha)\tilde{\mathbf{r}}^\wedge) d\alpha \\ &= \left(\int_0^1 (\mathbf{R}_d^b)^{\alpha \tilde{\mathbf{r}}^\wedge} (\mathbf{R}_d^b)^{-\alpha} d\alpha \right) \mathbf{R}_d^b. \end{aligned}$$

Using the definition of left Jacobian of SO(3) as given in (2.9) and the fact that $(\mathbf{R}\boldsymbol{\phi})^\wedge = \mathbf{R}\boldsymbol{\phi}^\wedge \mathbf{R}^\top$ [35], we can rearrange the above equation as

$$\dot{\mathbf{R}}_d^b (\mathbf{R}_d^b)^\top = \int_0^1 \left((\mathbf{R}_d^b)^{\alpha \tilde{\mathbf{r}}^\wedge} \right)^\wedge = \left(\mathbf{J}_l(\tilde{\mathbf{r}}) \dot{\tilde{\mathbf{r}}} \right)^\wedge. \quad (4.15)$$

Meanwhile, we have

$$\begin{aligned} \dot{\mathbf{R}}_d^b &= \mathbf{R}_d^b \mathcal{D} \boldsymbol{\omega}_{d/b}^\wedge = \mathbf{R}_d^b (\mathbf{R}_b^d \mathcal{B} \boldsymbol{\omega}_{d/b})^\wedge \\ &= \mathbf{R}_d^b \left(\mathbf{R}_b^d \mathcal{B} \boldsymbol{\omega}_{d/b}^\wedge (\mathbf{R}_b^d)^\top \right) = \mathcal{B} \boldsymbol{\omega}_{d/b}^\wedge (\mathbf{R}_b^d)^\top = \tilde{\boldsymbol{\omega}}^\wedge \mathbf{R}_d^b. \end{aligned} \quad (4.16)$$

Combining (4.15) and (4.16) yields attitude error dynamics (4.13).

The angular velocity error dynamics can be derived by taking the time derivative of (4.12) as

$$\begin{aligned} \dot{\tilde{\boldsymbol{\omega}}} &= \frac{d}{dt} (\mathbf{R}_d^b \mathcal{D} \boldsymbol{\omega}_{d/i}) - \mathcal{B} \dot{\boldsymbol{\omega}}_{b/i} \\ &= \dot{\mathbf{R}}_d^b \mathcal{D} \boldsymbol{\omega}_{d/i} + \mathbf{R}_d^b \mathcal{D} \dot{\boldsymbol{\omega}}_{d/i} - \mathcal{B} \dot{\boldsymbol{\omega}}_{b/i} \\ &\stackrel{(4.12), (4.16)}{=} (\mathbf{R}_d^b \mathcal{D} \boldsymbol{\omega}_{d/i} - \mathcal{B} \boldsymbol{\omega}_{b/i})^\wedge \mathbf{R}_d^b \mathcal{D} \boldsymbol{\omega}_{d/i} + \mathbf{R}_d^b \mathcal{D} \dot{\boldsymbol{\omega}}_{d/i} - \mathcal{B} \dot{\boldsymbol{\omega}}_{b/i} \\ &= -\mathcal{B} \boldsymbol{\omega}_{b/i}^\wedge \mathbf{R}_d^b \mathcal{D} \boldsymbol{\omega}_{d/i} + \mathbf{R}_d^b \mathcal{D} \dot{\boldsymbol{\omega}}_{d/i} - \mathcal{B} \dot{\boldsymbol{\omega}}_{b/i}. \end{aligned}$$

Multiplying both sides of the equation above with \mathbf{J} and substituting $\mathbf{J}^{\mathcal{B}} \dot{\boldsymbol{\omega}}_{b/i}$ with rotational dynamics (2.1d) yields the angular velocity error dynamics (4.14). \square

We are now poised to present the main results of adaptive tracking control for rotational dynamics, leveraging the following lemma to establish asymptotic stability.

Lemma 4.5.1 (A Special Case of Barbalat's Lemma [18]). *Let $f : [0, \infty) \rightarrow \mathbb{R}$. If $f, \dot{f} \in \mathcal{L}_\infty$ and $f \in \mathcal{L}_p$ for some $p \in [1, \infty)$, then $f(t) \rightarrow 0$ as $t \rightarrow \infty$.*

Theorem 4.5.1 (Adaptive Tracking Control for Rotational Dynamics). *Consider the attitude and angular velocity error dynamics given by (4.13) and (4.14), we define a control input (4.17) and an adaptive law for inertia matrix estimation as*

$$\boldsymbol{\tau} = \bar{\mathbf{J}} \dot{\boldsymbol{\omega}}_{d/i} - \bar{\mathbf{J}} \boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \bar{\mathbf{J}} \boldsymbol{\omega}_{b/i} + \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} + \mathbf{K}_w \tilde{\boldsymbol{\omega}}, \quad (4.17)$$

$$\dot{\bar{\mathbf{J}}} = \gamma_J (\tilde{\boldsymbol{\omega}}_{d/i}^\top - \tilde{\boldsymbol{\omega}}_{d/i}^\top \boldsymbol{\omega}_{b/i}^\wedge - \boldsymbol{\omega}_{b/i}^\wedge \tilde{\boldsymbol{\omega}}_{b/i}^\top), \quad (4.18)$$

where $\gamma_J \in \mathbb{R}$ is a positive constant, and $\mathbf{K}_r, \mathbf{K}_w \in \mathbb{R}^{3 \times 3}$ are positive definite gain matrices. The estimate of \mathbf{J} is given by $\bar{\mathbf{J}}$, with estimation error defined as $\tilde{\mathbf{J}} := \mathbf{J} - \bar{\mathbf{J}}$. Let $\tilde{\boldsymbol{\epsilon}} := \tilde{\boldsymbol{\omega}} + c\tilde{\mathbf{r}} \in \mathbb{R}^3$ be a composite error, then the zero equilibrium of rotational tracking errors $(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}) = (\mathbf{0}, \mathbf{0})$ is almost globally asymptotically stable, and furthermore the inertia matrix estimation error $\tilde{\mathbf{J}}$ is uniformly bounded if the control parameter $c \in \mathbb{R}$ is selected such that

$$c \in \left(0, \min \left\{ \sqrt{\frac{\lambda_r \lambda_m}{\lambda_M^2}}, \frac{4\lambda_r \lambda_w \varrho_m}{4\lambda_r \lambda_M \varrho_m \varrho_M + \lambda_w^2} \right\} \right), \quad (4.19)$$

where $\lambda_r := \lambda_{\min}(\mathbf{K}_r)$, $\lambda_w := \lambda_{\min}(\mathbf{K}_w)$, $\lambda_{\bar{w}} := \lambda_{\max}(\mathbf{K}_w)$, $\lambda_m := \lambda_{\min}(\mathbf{J})$, $\lambda_M := \lambda_{\max}(\mathbf{J})$, and ϱ_m, ϱ_M are the lower and upper bounds of $\|\mathbf{J}_l(\tilde{\mathbf{r}})^{-\top}\|_F$, respectively.

Proof. Similar to the proof of Theorem 4.4.1, we first derive the certainty-equivalent control law by assuming perfect knowledge on \mathbf{J} , followed by the relaxation of such assumption and the adaptive control synthesis.

Let $(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}) \in \mathcal{S} \times \mathbb{R}^3$, where $\mathcal{S} = \{\phi \mathbf{a} : |\phi| < \pi, \mathbf{a} \in \mathbb{S}^2\}$, and consider the Lyapunov candidate function

$$V_4(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}) = \frac{1}{2} \tilde{\mathbf{r}}^\top \mathbf{K}_r \tilde{\mathbf{r}} + \frac{1}{2} \tilde{\boldsymbol{\omega}}^\top \mathbf{J} \tilde{\boldsymbol{\omega}}.$$

By taking the time derivative and plugging it in the attitude and angular velocity error dynamics (4.13) and (4.14), we obtain

$$\begin{aligned}\dot{V}_4(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}) &= \tilde{\mathbf{r}}^\top \mathbf{K}_r \dot{\tilde{\mathbf{r}}} + \tilde{\boldsymbol{\omega}}^\top \mathbf{J} \dot{\tilde{\boldsymbol{\omega}}} \\ &= \tilde{\mathbf{r}}^\top \mathbf{K}_r \mathbf{J}_l(\tilde{\mathbf{r}})^{-1} \tilde{\boldsymbol{\omega}} + \tilde{\boldsymbol{\omega}}^\top (\mathbf{J} \dot{\boldsymbol{\omega}}_{d/i} - \mathbf{J} \boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \mathbf{J} \boldsymbol{\omega}_{b/i} - \boldsymbol{\tau}) \\ &= \tilde{\boldsymbol{\omega}}^\top (\mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} + \mathbf{J} \dot{\boldsymbol{\omega}}_{d/i} - \mathbf{J} \boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \mathbf{J} \boldsymbol{\omega}_{b/i} - \boldsymbol{\tau}).\end{aligned}$$

Hence, $\dot{V}_4(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}) = -\tilde{\boldsymbol{\omega}}^\top \mathbf{K}_w \tilde{\boldsymbol{\omega}} \leq 0$ if the desired (nominal) torque control input is designed as

$$\boldsymbol{\tau} = \mathbf{J} \dot{\boldsymbol{\omega}}_{d/i} - \mathbf{J} \boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \mathbf{J} \boldsymbol{\omega}_{b/i} + \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} + \mathbf{K}_w \tilde{\boldsymbol{\omega}}. \quad (4.20)$$

Furthermore, from $\dot{V}_4(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}) = 0$, we have $\tilde{\boldsymbol{\omega}} = \mathbf{0}$, which implies $\dot{\tilde{\boldsymbol{\omega}}} = \mathbf{0}$. Using (4.14) we can further deduce that $\mathbf{J} \dot{\boldsymbol{\omega}}_{d/i} - \mathbf{J} \boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \mathbf{J} \boldsymbol{\omega}_{b/i} - \boldsymbol{\tau} = \mathbf{0}$. Lastly, using the fact that $\mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r$ is full-rank and from (4.20), we conclude that $\tilde{\mathbf{r}} = \mathbf{0}$, hence showing that the largest invariant set in $\mathcal{S} \times \mathbb{R}^3$ is the origin. By LaSalle's invariance principle, we can draw conclusions on asymptotic stability.

Then, we consider \mathbf{J} to be uncertain and therefore we replace it by its estimate $\tilde{\mathbf{J}}$ in the certainty-equivalent controller (4.20), yielding (4.17). To derive the adaptive law for $\tilde{\mathbf{J}}$, we consider a composite Lyapunov candidate function as

$$\begin{aligned}V_5(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{J}}) &= V_4(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}) + (c\tilde{\mathbf{J}}\tilde{\boldsymbol{\omega}})^\top \tilde{\mathbf{r}} + \frac{1}{2\gamma_J} \|\tilde{\mathbf{J}}\|_F^2 \\ &\geq \frac{1}{2} \lambda_m \|\tilde{\boldsymbol{\omega}}\|^2 + \frac{1}{2} \lambda_r \|\tilde{\mathbf{r}}\|^2 - c\lambda_M \|\tilde{\boldsymbol{\omega}}\| \|\tilde{\mathbf{r}}\| + \frac{1}{2\gamma_J} \|\tilde{\mathbf{J}}\|_F^2 \\ &= \boldsymbol{\zeta}_1^\top \mathbf{W}_1 \boldsymbol{\zeta}_1,\end{aligned}$$

where

$$\begin{aligned}\boldsymbol{\zeta}_1 &:= [\|\tilde{\mathbf{r}}\|, \|\tilde{\boldsymbol{\omega}}\|, \|\tilde{\mathbf{J}}\|_F]^\top \in \mathbb{R}^3 \quad \text{and} \\ \mathbf{W}_1 &:= \begin{bmatrix} \frac{1}{2} \lambda_r & -\frac{1}{2} c \lambda_M & 0 \\ -\frac{1}{2} c \lambda_M & \frac{1}{2} \lambda_m & 0 \\ 0 & 0 & \frac{1}{2\gamma_J} \end{bmatrix} \in \mathbb{R}^{3 \times 3}.\end{aligned}$$

Therefore, ensuring $V_5(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{J}}) > 0$ is equivalent to having $\mathbf{W}_1 > 0$, or more precisely by Sylvester's criterion,

$$|c| < \sqrt{\frac{\lambda_r \lambda_m}{\lambda_M^2}}. \quad (4.21)$$

Taking the time derivative of $V_5(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{J}})$ and using trace properties as well as the fact that $\mathbf{x}^\top(\mathbf{y}^\wedge \mathbf{z}) = \mathbf{y}^\top(\mathbf{z}^\wedge \mathbf{x}) = \mathbf{z}^\top(\mathbf{x}^\wedge \mathbf{y})$ holds for arbitrary vectors $\mathbf{x}, \mathbf{y}, \mathbf{z}$ with same dimension, we have

$$\begin{aligned}
\dot{V}_5(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{J}}) &= \tilde{\boldsymbol{\omega}}^\top \mathbf{J} \dot{\tilde{\boldsymbol{\omega}}} + \tilde{\mathbf{r}}^\top \mathbf{K}_r \dot{\tilde{\mathbf{r}}} + (c\mathbf{J}\dot{\tilde{\boldsymbol{\omega}}})^\top \tilde{\mathbf{r}} + (c\mathbf{J}\tilde{\boldsymbol{\omega}})^\top \dot{\tilde{\mathbf{r}}} + \frac{1}{\gamma_J} \text{tr}(\tilde{\mathbf{J}}^\top \dot{\tilde{\mathbf{J}}}) \\
&\stackrel{(4.13), (4.14), (4.17)}{=} \tilde{\boldsymbol{\omega}}^\top (\tilde{\mathbf{J}}\dot{\boldsymbol{\omega}}_{d/i} - \tilde{\mathbf{J}}\boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \mathbf{J}\boldsymbol{\omega}_{b/i} - \boldsymbol{\omega}_{b/i}^\wedge \tilde{\mathbf{J}}\boldsymbol{\omega}_{b/i} \\
&\quad - \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} - \mathbf{K}_w \tilde{\boldsymbol{\omega}}) + c\tilde{\mathbf{r}}^\top (\tilde{\mathbf{J}}\dot{\boldsymbol{\omega}}_{d/i} - \tilde{\mathbf{J}}\boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + \boldsymbol{\omega}_{b/i}^\wedge \mathbf{J}\boldsymbol{\omega}_{b/i} - \boldsymbol{\omega}_{b/i}^\wedge \tilde{\mathbf{J}}\boldsymbol{\omega}_{b/i} \\
&\quad - \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} - \mathbf{K}_w \tilde{\boldsymbol{\omega}}) + \tilde{\mathbf{r}}^\top \mathbf{K}_r \mathbf{J}_l(\tilde{\mathbf{r}})^{-1} \tilde{\boldsymbol{\omega}} + (c\mathbf{J}\tilde{\boldsymbol{\omega}})^\top \mathbf{J}_l(\tilde{\mathbf{r}})^{-1} \tilde{\boldsymbol{\omega}} + \frac{1}{\gamma_J} \text{tr}(\tilde{\mathbf{J}}^\top \dot{\tilde{\mathbf{J}}}) \\
&= (\tilde{\boldsymbol{\omega}}^\top + c\tilde{\mathbf{r}}^\top) \tilde{\mathbf{J}} \dot{\boldsymbol{\omega}}_{d/i} - (\tilde{\boldsymbol{\omega}}^\top + c\tilde{\mathbf{r}}^\top) \tilde{\mathbf{J}} \boldsymbol{\omega}_{b/i}^\wedge \boldsymbol{\omega}_{d/i} + (\tilde{\mathbf{J}}\boldsymbol{\omega}_{b/i})^\top (\tilde{\boldsymbol{\omega}}^\wedge \boldsymbol{\omega}_{b/i} + c\tilde{\mathbf{r}}^\wedge \boldsymbol{\omega}_{b/i}) \\
&\quad - (\tilde{\boldsymbol{\omega}}^\top + c\tilde{\mathbf{r}}^\top) \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} - (\tilde{\boldsymbol{\omega}}^\top + c\tilde{\mathbf{r}}^\top) \mathbf{K}_w \tilde{\boldsymbol{\omega}} + \tilde{\mathbf{r}}^\top \mathbf{K}_r \mathbf{J}_l(\tilde{\mathbf{r}})^{-1} \tilde{\boldsymbol{\omega}} \\
&\quad + (c\mathbf{J}\tilde{\boldsymbol{\omega}})^\top \mathbf{J}_l(\tilde{\mathbf{r}})^{-1} \tilde{\boldsymbol{\omega}} + \frac{1}{\gamma_J} \text{tr}(\tilde{\mathbf{J}}^\top \dot{\tilde{\mathbf{J}}}) \\
&= \text{tr} \left\{ \tilde{\mathbf{J}}^\top \left[-\frac{1}{\gamma_J} \dot{\tilde{\mathbf{J}}} + \tilde{\boldsymbol{\omega}}_{d/i}^\top + \tilde{\boldsymbol{\omega}}_{d/i}^\top \boldsymbol{\omega}_{b/i}^\wedge - \boldsymbol{\omega}_{b/i}^\wedge \tilde{\boldsymbol{\omega}}_{b/i}^\top \right] \right\} + \tilde{\boldsymbol{\omega}}^\top (c\mathbf{J}\mathbf{J}_l(\tilde{\mathbf{r}})^{-1} - \mathbf{K}_w) \tilde{\boldsymbol{\omega}} \\
&\quad - c\tilde{\mathbf{r}}^\top \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} - c\tilde{\mathbf{r}}^\top \mathbf{K}_w \tilde{\boldsymbol{\omega}}.
\end{aligned}$$

By selecting the adaptive law for inertia matrix estimation as given in (4.18), we can show that

$$\begin{aligned}
\dot{V}_5(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{J}}) &= -c\tilde{\mathbf{r}}^\top \mathbf{J}_l(\tilde{\mathbf{r}})^{-\top} \mathbf{K}_r \tilde{\mathbf{r}} + \tilde{\boldsymbol{\omega}}^\top (c\mathbf{J}\mathbf{J}_l(\tilde{\mathbf{r}})^{-1} - \mathbf{K}_w) \tilde{\boldsymbol{\omega}} - c\tilde{\mathbf{r}}^\top \mathbf{K}_w \tilde{\boldsymbol{\omega}} \\
&\leq -c\lambda_{\underline{r}} \varrho_m \|\tilde{\mathbf{r}}\|^2 + (c\lambda_M \varrho_M - \lambda_{\underline{w}}) \|\tilde{\boldsymbol{\omega}}\|^2 + c\lambda_{\overline{w}} \|\tilde{\mathbf{r}}\| \|\tilde{\boldsymbol{\omega}}\| \\
&= -\zeta_2^\top \mathbf{W}_2 \zeta_2,
\end{aligned}$$

where

$$\begin{aligned}
\zeta_2 &:= [\|\tilde{\mathbf{r}}\|, \|\tilde{\boldsymbol{\omega}}\|]^\top \in \mathbb{R}^2 \quad \text{and} \\
\mathbf{W}_2 &:= \begin{bmatrix} c\lambda_{\underline{r}} \varrho_m & -\frac{1}{2}c\lambda_{\overline{w}} \\ -\frac{1}{2}c\lambda_{\overline{w}} & \lambda_{\underline{w}} - c\lambda_M \varrho_M \end{bmatrix} \in \mathbb{R}^{2 \times 2}.
\end{aligned}$$

That being said, $\dot{V}_5(\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{J}})$ is bounded from above, being semi-negative definite when $\mathbf{W}_2 > 0$, or equivalently

$$0 < c < \frac{4\lambda_{\underline{r}} \lambda_{\underline{w}} \varrho_m}{4\lambda_{\underline{r}} \lambda_M \varrho_m \varrho_M + \lambda_{\overline{w}}^2}, \quad (4.22)$$

which, together with the inequality given in (4.21), yields the sufficient condition (4.19). Hence, by far, we have shown that $\lim_{t \rightarrow \infty} V_5(t) = V_{5,\infty}$ and $\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}}, \tilde{\mathbf{J}} \in \mathcal{L}_\infty$.

Furthermore, from Eq. (4.13) and Eq. (4.14), we can deduce that $\dot{\tilde{\mathbf{r}}}, \dot{\tilde{\boldsymbol{\omega}}} \in \mathcal{L}_\infty$. Since $\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}} \in \mathcal{L}_2$ (by having $\int_0^\infty \zeta_2(\tau)^\top \mathbf{W}_2 \zeta_2(\tau) d\tau \leq V_5(0) - V_{5,\infty} < \infty$), it can be then concluded using Lemma 4.5.1 that $\tilde{\mathbf{r}}, \tilde{\boldsymbol{\omega}} \rightarrow 0$ for $t \rightarrow \infty$. \square

Remark. No assumptions were made about the symmetry of the estimated inertia matrix $\tilde{\mathbf{J}}$ (i.e., $\tilde{\mathbf{J}} = \tilde{\mathbf{J}}^\top$) throughout the proof of Theorem 4.5.1. This is due to the fact that the (direct) adaptive law (4.18) does not guarantee convergence to the ground truth \mathbf{J} ; it only ensures the boundedness of the estimation error. Acknowledging that the symmetric property of inertia matrix is grounded in physics, one could explore alternative forms of adaptive laws (e.g., via indirect methods) or learning rules (e.g., neural networks) to incorporate this structural information.

Moreover, the inequality condition (4.22) can be obtained due to the fact that $\lambda_r, \lambda_M, \varrho_m, \varrho_M$ are positive. Observing the first two conditions is straightforward: $\mathbf{K}_r > 0$ is ensured by design, as stated in Theorem 4.5.1; rigid body inertia matrices are known to be positive semidefinite, and for quadrotors, it is positive definite, i.e., $\mathbf{J} > 0$. For ϱ_m, ϱ_M , they are the lower and upper bound for $\|\mathbf{J}_l(\tilde{\mathbf{r}})^{-\top}\|_F$, hence being functions of $\tilde{\mathbf{r}} \in \mathcal{S} = \{\phi \mathbf{a} : |\phi| < \pi, \mathbf{a} \in \mathbb{S}^2\}$. We show their positiveness in Fig. 4.2, through numerical analysis in MATLAB. \square

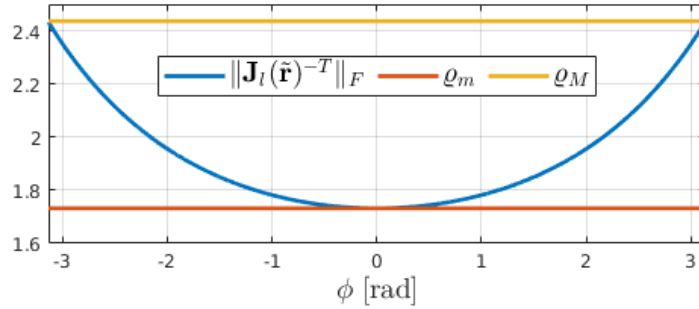


Fig. 4.2 Numerical analysis of the bounds of $\|\mathbf{J}_l(\tilde{\mathbf{r}})^{-\top}\|_F$ over $\mathcal{S} = \{\phi \mathbf{a} : |\phi| < \pi, \mathbf{a} \in \mathbb{S}^2\}$.

Similar to the robustification of adaptive law of mass estimation (4.2), we apply Γ -projection to (4.18) with slight difference of using its matrix extension as stated in the sequel.

Proposition 4.5.2 (Robustification of Adaptive Law using Projection Operator). *The adaptive law of inertia matrix estimation (4.18) is adjusted as*

$$\dot{\tilde{\mathbf{J}}} = \text{Proj}_{\gamma_J}(\tilde{\mathbf{J}}, \tilde{\mathbf{e}}\boldsymbol{\omega}_{d/i}^\top - \tilde{\mathbf{e}}\boldsymbol{\omega}_{d/i}^\top \boldsymbol{\omega}_{b/i}^\wedge - \boldsymbol{\omega}_{b/i}^\wedge \tilde{\mathbf{e}}\boldsymbol{\omega}_{b/i}^\top, H),$$

where H consists of three continuously differentiable convex functions, specifically $H = [h_1, h_2, h_3]^\top \in \mathbb{R}^3$, with

$$h_i(\bar{\mathbf{J}}(:, i), \mathbf{j}_0(i), \boldsymbol{\epsilon}_J(i)) = \frac{\|\bar{\mathbf{J}}(:, i)\|^2 - \mathbf{j}_0(i)^2}{2\boldsymbol{\epsilon}_J(i)\mathbf{j}_0(i) + \boldsymbol{\epsilon}_J(i)^2}, \quad i = \{1, 2, 3\},$$

and $\mathbf{j}_0, \boldsymbol{\epsilon}_J \in \mathbb{R}^3$ are two constant vector quantities.

Proof. The readers interested in the proof of projection operator are kindly referred to [18, 107] for details. \square

4.6 Results and Discussion

In this section, we present the simulation results of our proposed controller for executing aggressive maneuvers. We start with introducing the simulation environment along with the corresponding settings adopted in our study, followed by showcasing the tracking performance as well as the parameter estimation under uncertainties. In particular, we compare our proposed approach with a prior work [95], which also uses the logarithmic map of $SO(3)$ to formulate the attitude configuration error, and demonstrate the superiority of our controller in handling parameter uncertainties.

4.6.1 Simulator

The simulation environment for evaluating our proposed controller is the MathWorks quadcopter project², based on the Parrot[®] series of mini-drones. It consists of a nonlinear quadrotor model with sensor and actuator dynamics, as well as dynamic environmental modeling, providing a medium-fidelity simulator ideal for verifying control algorithms. Comparable with other simulator alternatives such as AirSim and Gazebo, as demonstrated in Section 3.4.1, Simulink features a user-friendly interface, MATLAB integration, extensive block library, real-time simulation, code generation capabilities, and widespread industry adoption, making it particularly well-suited for control system simulation.

²MathWorks quadcopter project based on the Parrot[®] series of mini-drones: <https://www.mathworks.com/help/aeroblks/quadcopter-project.html>.

The nominal parameters of the quadrotor used in the simulation are $m = 0.063\text{kg}$, $\mathbf{J} = 1e^{-4} \cdot \text{diag}(0.5829, 0.7169, 1)\text{kgm}^2$, $g = 9.8\text{ms}^{-2}$.

4.6.2 Maneuver #1: Doing a 360° Flip

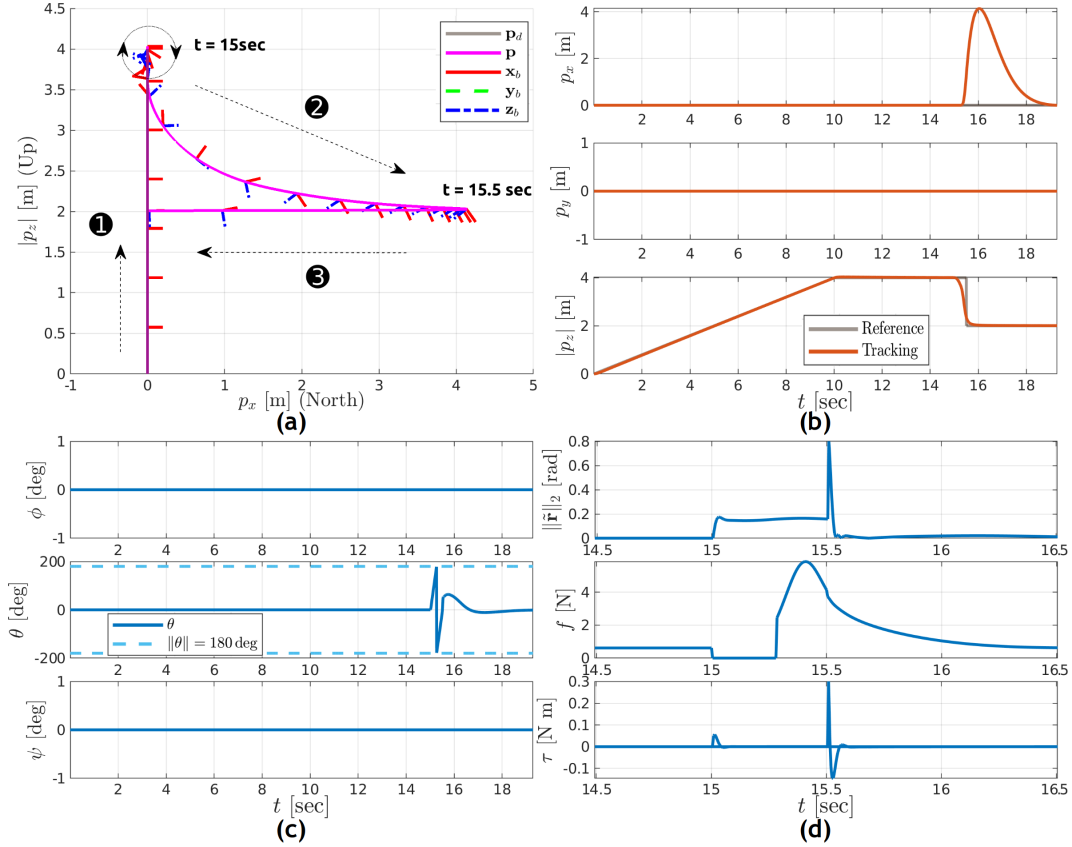


Fig. 4.3 **360° flip maneuver.** (a) Flip illustration in X-Z plane. (b) Quadrotor position \mathbf{p} . (c) Euler angles ϕ, θ, ψ . (d) Attitude error $\|\tilde{\mathbf{r}}\|_2$ and control inputs f, τ .

We first demonstrate the results of the proposed controller for performing a 360° flip maneuver. This is achieved by concatenating three control phases, which is shown in Fig. 4.3(a). The quadrotor is commanded to first take off and stop at a hovering condition: $\mathbf{p} = (0, 0, 4)^\top$, $\mathbf{v} = \boldsymbol{\omega} = (0, 0, 0)^\top$, $\mathbf{R} = \mathbf{I}_3$. Then, at 15sec, it is commanded to follow the desired trajectory:

$$\mathbf{R}_d^i(t) = \mathbf{I}_3 + \sin(4\pi t)\boldsymbol{\phi}_d^\wedge + (1 - \cos(4\pi t))(\boldsymbol{\phi}_d\boldsymbol{\phi}_d^\top - \mathbf{I}_3),$$

$$\boldsymbol{\phi}_d = [0, 1, 0]^\top,$$

which is a flipping maneuver where the quadrotor rotates about ϕ_d by 360° . During the execution of such a trajectory, the backstepping controller is disabled since the control objective is to rotate the quadrotor instead of driving it to a desired position. Correspondingly, the desired rotation matrix is directly fed to the geometric controller as a reference signal to generate proper torques for attitude tracking, yielding a so-called attitude controlled flight mode [93]. Lastly, the backstepping controller is again enabled to stabilize the quadrotor at a hovering condition: $\mathbf{p} = (0, 0, -2)^\top$, $\mathbf{R} = \mathbf{I}_3$. No uncertainties are considered for this maneuver as the quadrotor completes the flip in under 0.1 sec. Any uncertainties in mass or inertia matrix prior to the flip will be addressed beforehand.

Figure 4.3(b) and Figure 4.3(c) show the position and Euler angles, respectively, during the 360° flip. The latter illustrates the pitch angle θ transitioning from 0 to 180 deg and then from -180 deg back to 0, completing the 360° flip. Notably, during the flip around ϕ_d , the quadrotor deviates from its original position in x -axis, due to the temporary deactivation of backstepping controller. The corresponding attitude error and control inputs are shown in Fig 4.3(d).

4.6.3 Maneuver #2: Tracking an Elliptical Helix Trajectory

Next, we demonstrate the ability of the quadrotor to track an elliptical helix trajectory (see Fig. 4.4(a)) in the presence of imprecise knowledge on mass and inertia matrix.

The quadrotor is commanded to first take off and fly to the initial position: $[\mathbf{p}^\top(t) \ \psi(t)] = [0, 0, -5, 0]$, then track the following desired trajectory starting at 10sec:

$$[\mathbf{p}_{\text{ref}}^\top(t) \ \psi_{\text{ref}}(t)] = [t, \sin(\frac{\pi}{7}t), -4 - \cos(\frac{\pi}{7}t), \sin(0.3t)].$$

As indicated in Figure 4.4(b), we introduce an increase of inertia matrix of the quadrotor to $\text{diag}(0.0030, 0.0037, 0.0051)\text{kgm}^2$ at 10sec and an increase of mass to 0.093kg at 20sec.

To highlight the importance of our adaptive designs, we conduct a comparative analysis of the tracking performance between our proposed controller and the one in [95]. From Fig. 4.4(b), it can be observed that our controller successfully tracks the reference trajectory under conditions of uncertainty in both mass and inertia matrix (see orange curve), whereas the controller in [95] fails for either

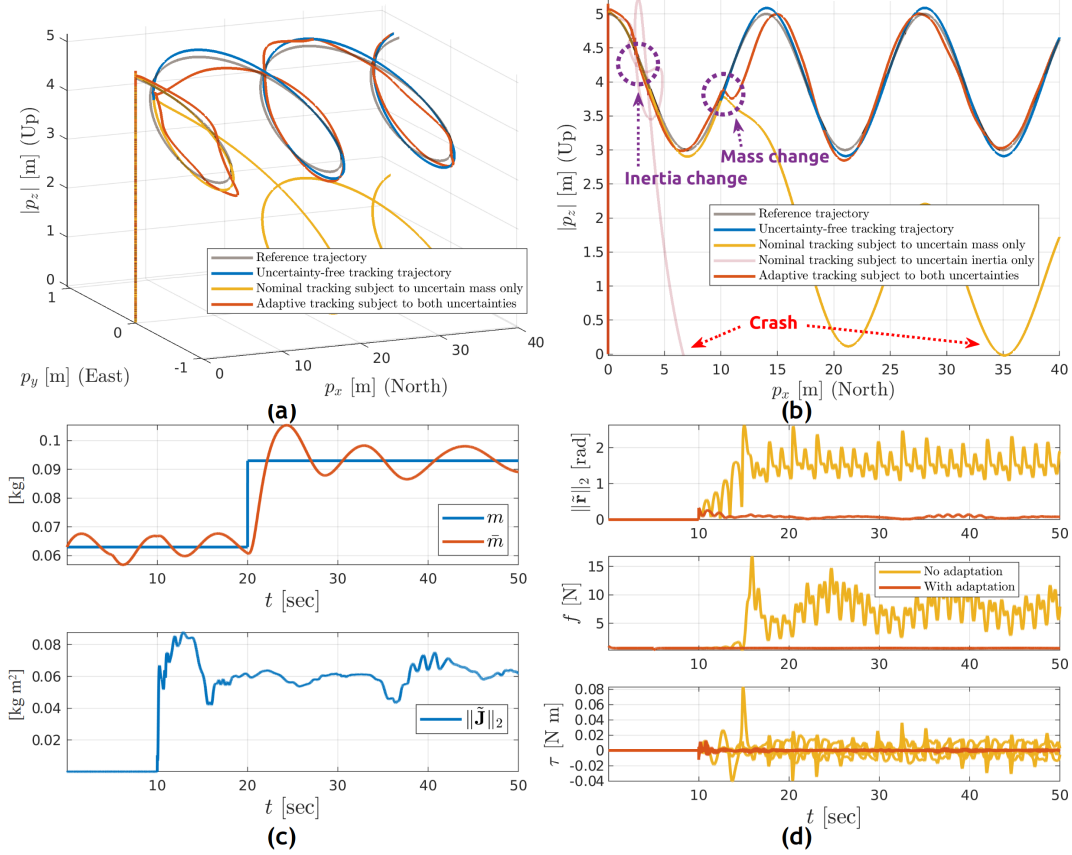


Fig. 4.4 **Elliptical helix trajectory tracking.** (a) 3-D visualization[†]. (b) Comparisons under nominal and uncertain scenarios. (c) Mass estimation \hat{m} and inertia matrix estimation error $\|\tilde{\mathbf{J}}\|_2$. (d) Attitude error $\|\tilde{\mathbf{r}}\|_2$ and control inputs f, τ .

[†] The nominal tracking subject to uncertain inertia matrix is not depicted in the 3-D visualization due to its unstable behavior, resulting in a cluttered and confusing plot. However, for completeness, we include it in the X-Z plane visualization in Fig. 4.4(b).

single uncertainty (see yellow curve when only uncertain mass exists and light red curve when only uncertain inertia matrix exists), even resulting in instability. Bounded mass estimate and the estimation error of inertia matrix are further noted for our adaptive designs in Fig. 4.4(c), validating Theorem 4.4.1 and 4.5.1. Oscillations have been observed in parameter estimation, which may arise from either the fast dynamics of the controlled system or a high adaptation gain being employed. Moreover, the attitude error and the control inputs are presented in Fig. 4.4(d), from which we observe that the controller in [95] (yellow curve) is incapable of reducing the attitude error in the presence of uncertainties, thereby

stressing the importance as well as the effectiveness of the adaptive design in our proposed controller.

4.6.4 Maneuver #3: Tracking a Figure-8 Trajectory

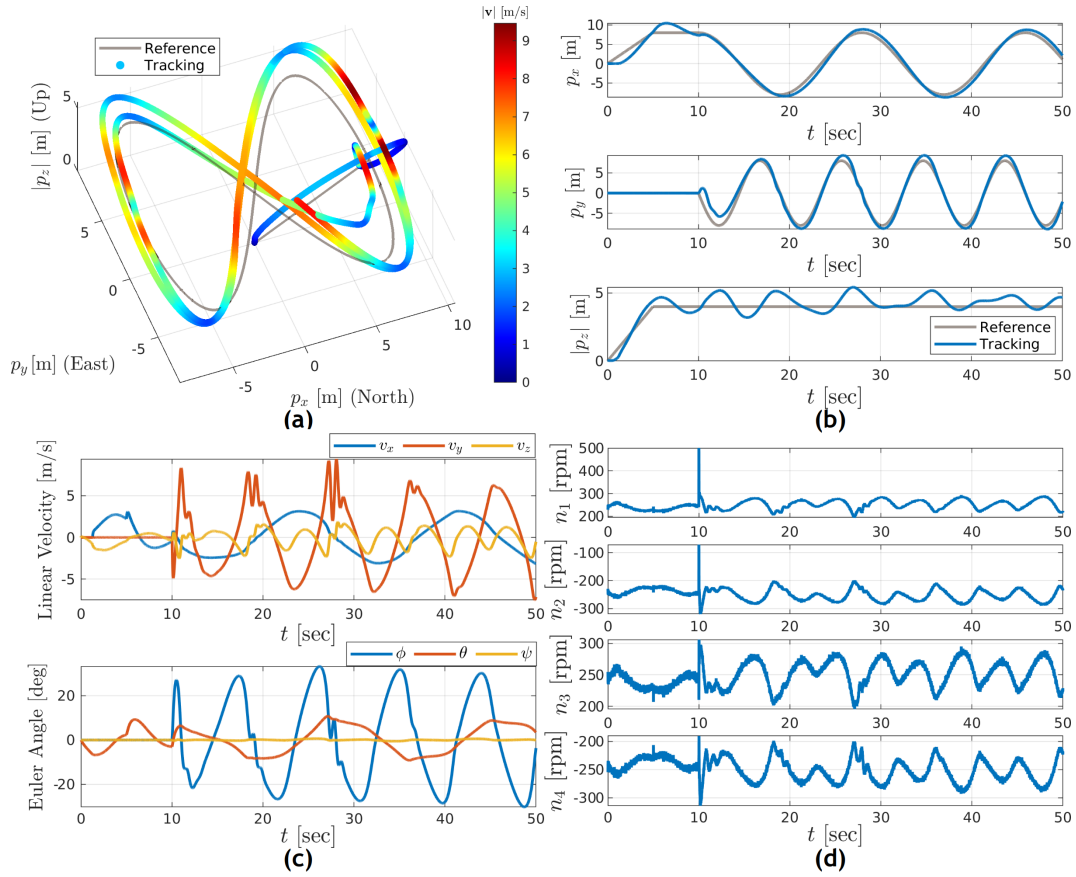


Fig. 4.5 **Figure-8 trajectory tracking.** (a) 3-D visualization with a colorbar indicating the absolute value of the magnitude of linear velocity \mathbf{v} . (b) Quadrotor position \mathbf{p} . (c) Linear velocity \mathbf{v} and Euler angles ϕ, θ, ψ . (d) Motor speeds.

Lastly, we demonstrate the effectiveness of the proposed controller in successfully executing a variety of aggressive maneuvers by tracking a high-speed Figure-8 trajectory (see Fig. 4.5(a)). Similar to Section 4.6.3, the quadrotor is first commanded to reach the starting position and then begins to track the following desired trajectory at 10sec:

$$[\mathbf{p}_{\text{ref}}^{\top}(t) \ \psi_{\text{ref}}(t)] = [8 \cos(0.35t), 8 \sin(0.7t), -4, 0].$$

In addition to the parametric uncertainties in m and \mathbf{J} , we incorporate sensor noises³ and actuator dynamics⁴ in the simulation.

From Fig. 4.5(b) and (d), we observe satisfactory tracking performance, particularly in horizontal coordinates, alongside the corresponding motor speeds reported in RPM, respectively. The aggressive nature of the commanded maneuvers can be revealed in Fig. 4.5(a) and (c), showcasing both high linear velocity (with a maximum absolute value exceeding 9m/sec) and large roll angle (with a maximum absolute value more than 30deg). It is noteworthy that the performance of altitude tracking exhibits room for improvement, which is attributed to the rapid changes in linear velocities and Euler angles, leading to altitude loss. Our simulations (results not presented here) suggest that mitigating the aggressiveness of maneuvers can lead to enhanced altitude tracking performance.

4.7 Conclusions

In this chapter, we introduced a novel adaptive geometric tracking controller tailored for quadrotor aggressive maneuvers. The control synthesis is inherently nonlinear, leveraging BSC for position tracking and the logarithmic map of $SO(3)$ for attitude tracking, underpinned by rigorous proofs using Lyapunov analysis. The advantages of our proposed approach lie in its ability to effectively handle uncertainties, making it robust against variations in mass and inertia matrices as well as non-parametric uncertainties such as sensor noises. This is achieved through two specifically designed adaptive laws, complemented by projection operators that enhance robustness to sensor noise.

Through extensive simulations, we demonstrated the effectiveness of our proposed controller in executing diverse aggressive maneuvers, including a 360° flip, an elliptical helix trajectory, and a figure-8 trajectory. Notably, our controller exhibits superior performance compared to a recent study analogous to our approach, thereby showcasing applicability beyond uncertainty-free scenarios.

³The 3-axis Inertial Measurement Unit (IMU) employed in the simulation sets accelerometer measurement bias to $[0.0900, -0.0600, 0.3370]^T \text{msec}^2$, gyro measurement bias to $[-0.0095, -0.0075, 0.0015]^T \text{rad/sec}$, and noise power (or the height of the power spectral density of the white noise) for each axis of the accelerometer and gyroscope to $1.0 \times 10^{-3} \cdot [0.2183, 0.1864, 0.3725, 0.0000, 0.0000, 0.0000]^T \text{m/sec}^2/\text{Hz}$.

⁴This includes rotor dynamics, which is a nonlinear function of air density, body velocity of rotor, angular velocity, and rotor speed; see, e.g., [108] for more details.

However, our study identified room for improvement in altitude tracking for figure-8 maneuvers, potentially influenced by rapid changes in linear velocities and Euler angles during aggressive maneuvers. Future potential work also involves validating the proposed controller through real-world experimental tests, providing a practical assessment of its efficacy. Furthermore, our ongoing research focuses on extending the controller to explicitly address unmodeled dynamics and disturbances, aiming to broaden its applicability across various operational scenarios.

Chapter 5

Learning-based Controller Design for Non-Parametric Uncertainties

5.1 Introduction

While quadrotors have achieved impressive feats in tasks such as hovering and trajectory tracking [15, 94, 105], their complex dynamics, nonlinearity, and underactuation make them vulnerable to uncertainties and disturbances, such as wind. Ignoring these factors can lead to detrimental effects on control, possibly resulting in crashes. To improve robustness and prevent instability in dynamic control systems, *model-based* approaches, particularly robust control and adaptive control, have undergone extensive studies over the last two decades; see, e.g., [13] and the references therein. It is noteworthy, however, that robust control may exhibit over-conservative performance due to assumptions made on the uncertainty bounds; whereas adaptive control, mostly suitable in handling parametric uncertainties, requires persistent excitation conditions. In complex and highly dynamic scenarios, data are accessible in the form of online measurements and prior knowledge about uncertainties is often scant. Hence, *data-driven* approaches are enticing for both system identification and controller design; see, e.g., leveraging the direct utilization of data to improve performance and alleviating the requirements for extensive prior knowledge [21, 22, 26–28, 109, 110].

Data-driven approaches can be generally classified into methods based on learning and those that do not rely on learning. The latter includes methods

like ADRC [21] and Hankel-matrix-based identification and control [22]. Despite ADRC's effectiveness and widespread recognition as a robust alternative to PID control, it requires tuning, and a rigorous stability proof seems lacking. While Hankel-matrix-based method exhibits promise for linear systems, its extension to nonlinear systems remains an open research area. Learning-based methods, on the other hand, harness the powerful approximation capabilities of ANNs, leveraging recent advancements in big data and computer hardware. A number of studies utilizing DNNs to learn various uncertainties have substantiated the effectiveness of learning-based methods; see, e.g., [26–28]. Nonetheless, a well-known challenge with deep learning lies in its *interpretability*, primarily stemming from the black-box nature of DNNs. This could hinder the deployment of DNN-based solutions in real-world safety-critical systems such as UAVs, since practitioners may lack trust in these models [29]. A further challenging issue is related to the *generalizability* of the trained model to unseen data, particularly for offline-trained models, as their performance may degrade when confronted with OOD data samples.

An intuitive solution for achieving generalizable and interpretable learning-based models is to combine model-based and data-driven approaches, trading off between the best features of both worlds [10]. A viable strategy may be, for example, to use domain knowledge to construct a nominal system model and employ learning to understand the residual dynamics, i.e., the part of system behavior that the model is not able to reproduce. Yet, the integration of knowledge and data remains challenging, particularly as the learning module for the residuals lacks transparency and requires interpretation. A recent promising paradigm, PIML [30], emphasizes the alignment of learning models with physical principles through the incorporation of knowledge via inductive and learning bias. Research in UAV modeling and control using PIML is recently gaining momentum, see e.g. [31, 111, 112]. Another approach to enhance the interpretability of learning models is through post-hoc analysis [29]. For example, in [113], decision trees were adopted as human-readable reasoning modules derived from neuron activation of small-sized Liquid Time Constant (LTC) networks for quadrotor goal reaching and obstacle avoidance. However, there is a lack of exploration into efficient algorithms for extracting these decision trees for more complex tasks and network models. Additionally, clustering methods such as t-SNE serve as valuable tools for interpreting network parameters or predictions [28]. It is crucial to highlight that

among these methods, offline-trained ANNs may exhibit limited generalizability to unseen scenarios.

In this chapter, we propose an online learning module for learning the unknown residual dynamics of quadrotors on the fly. Specifically, we employ an ESN[48] due to its powerful learning capabilities for temporal patterns from time-series data and its efficient online training through the RC paradigm [114]. This ensures broader applicability of our approach to unseen scenarios by not relying on a set of fixed offline-trained model parameters. Moreover, we present post-hoc interpretation techniques tailored for ESNs, aimed at demystifying the “black box” and enhancing trustworthiness. Our analysis delves into reservoir layers from a dynamical systems perspective, followed by an elucidation of network predictions through visualization. The main outcomes from this chapter are summarized as follows:

- We designed an online residual learning module based on ESN to actively compensate for unknown quadrotor dynamics or disturbances in real time.
- We enhanced the trustworthiness of the model by leveraging reservoir dynamics and the interpretations for ESN predictions.

5.2 Problem Formulation

We aim to address the tracking control problem of given reference trajectories $\xi_{\text{ref}}(t) = [\mathbf{p}_{\text{ref}}^\top(t) \psi_{\text{ref}}(t)]^\top \in \mathbb{R}^4$ (i.e., position and yaw angle) for a quadrotor governed by (2.1), subject to unknown uncertainties, represented by \mathbf{f}_a and $\boldsymbol{\tau}_a$. Our control objective is to achieve asymptotic tracking performance even in the presence of uncertainties. For the controller design discussed in Section 5.4, we make the following assumption that is widely adopted and practically valid.

Assumption 5.2.1. *The desired trajectories satisfy $\xi_{\text{ref}}(t) \in \mathcal{C}^3$ (i.e., the derivatives up to $\xi_{\text{ref}}^{(3)}(t)$ exist and are continuous) and these derivatives are bounded.*

Assumption 5.2.2. *The uncertainties are continuous and bounded.*

5.3 Reservoir Computing Paradigm

Before delving into the controller design, it is essential to lay the foundation for the RC paradigm. This paradigm facilitates in-flight online learning of unknown residual dynamics in quadrotors with the use of an ESN, thereby enabling adaptability to unforeseen scenarios. In this section, we present the mathematical model of ESNs, discuss their training methods, and underscore the significance of the Echo State Property (ESP) for post-training analysis of their dynamics.

5.3.1 Mathematical Model of Echo State Networks

The functioning of the simplest of ESN (i.e., a *shallow* ESN with feedback connections and leaky integration) can be formulated in terms of a dynamical system, whose state transition and output equations can be described as

$$\mathbf{x}[k] = (1 - \alpha)\mathbf{x}[k-1] + \alpha \tanh(\mathbf{W}_{\text{in}}\mathbf{u}[k] + \mathbf{W}_{\text{state}}\mathbf{x}[k-1] + \mathbf{W}_{\text{fb}}\mathbf{y}[k-1]), \quad (5.1a)$$

$$\mathbf{y}[k] = \mathbf{W}_{\text{out}}(\mathbf{x}[k]; \mathbf{u}[k]), \quad (5.1b)$$

where $\mathbf{u} \in \mathbb{R}^{N_u}$ denotes the input sequence, $\mathbf{x} \in \mathbb{R}^{N_r}$ denotes the reservoir state, $\mathbf{y} \in \mathbb{R}^{N_y}$ denotes the output sequence, $\alpha \in [0, 1]$ denotes the leaking rate, $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_r \times N_u}$, $\mathbf{W}_{\text{state}} \in \mathbb{R}^{N_r \times N_r}$, $\mathbf{W}_{\text{fb}} \in \mathbb{R}^{N_r \times N_y}$, $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_y \times (N_r + N_u)}$ denote input-to-reservoir, reservoir, readout-to-reservoir, and readout weight matrix, respectively. For better learning capability, one can further enhance the reservoir layer by stacking multiple layers, akin to the architecture used for deep learning. To this aim, in this work we consider the following *deep* ESN¹ with N_l reservoir layers (see Fig. 5.2):

$$\mathbf{x}^{(l)}[k] = \begin{cases} \tanh(\mathbf{W}_{\text{in}}\mathbf{u}[k] + \mathbf{W}_{\text{state}}^{(l)}\mathbf{x}^{(l)}[k-1]) & \text{if } l = 1, \\ \tanh(\mathbf{W}_{\text{il}}^{(l)}\mathbf{x}^{(l-1)}[k] + \mathbf{W}_{\text{state}}^{(l)}\mathbf{x}^{(l)}[k-1]) & \text{if } l = 2, \dots, N_l, \end{cases} \quad (5.2a)$$

$$\mathbf{y}^{(l)}[k] = \mathbf{W}_{\text{out}}^{(l)}\mathbf{x}^{(l)}[k], \quad (5.2b)$$

where the superscript (l) is used to represent the network parameters at the l -th layer, and $\mathbf{W}_{\text{il}} \in \mathbb{R}^{N_r \times N_r}$ denotes the inter-layer reservoir weight matrix. Note

¹The mathematical expression of each layer in the deep ESN can be obtained from (5.1a) and (5.1b) by setting $\alpha = 1$, omitting readout-to-reservoir connections, and removing the connections from input to readout.

that reservoir states (depicted as blue circles in Fig. 5.2), in dynamical systems, are synonymous with the system's state, while a reservoir (shown by blue-shaded ellipses in Fig. 5.2) comprises a collection of such states, with possibly feedforward or recurrent connections.

5.3.2 Training Methods

Unlike other ANNs, for which back-propagation is commonly the standard training technique, training process of an ESN follows the RC paradigm, which only involves the update of the readout weight matrix, whereas the rest of the weight matrices are initialized at random [114]. Training the readouts of an ESN without feedback connections can be generally addressed by means of linear regression, i.e., solving the following optimization problem:

$$\mathbf{W}_{\text{out}}^{(l)} = \operatorname{argmin} \|\mathbf{W}_{\text{out}}^{(l)} \mathbf{X}^{(l)} - \mathbf{Y}_{\text{label}}\|_2^2, \quad (5.3)$$

where $\mathbf{X}^{(l)} = [\mathbf{x}^{(l)}(1), \dots, \mathbf{x}^{(l)}(N)]$, $\mathbf{Y}_{\text{label}} = [\mathbf{y}_{\text{label}}(1), \dots, \mathbf{y}_{\text{label}}(N)]$ are the collection of reservoir states and target values over the time span $k = 1, \dots, N$. Hence, Moore-Penrose pseudo-inversion $\mathbf{W}_{\text{out}} = \mathbf{Y}_{\text{label}} \mathbf{X}^+$ or ridge regression $\mathbf{W}_{\text{out}} = \mathbf{Y}_{\text{label}} \mathbf{X}^{\top} (\mathbf{X} \mathbf{X}^{\top} + \lambda_r \mathbf{I})^{-1}$ ($\lambda_r \in \mathbb{R}_{>0}$ is the regularization coefficient) can be used for offline or semi-online training.

5.3.3 Echo State Property

Since all the reservoir weight matrices are initialized at random, ESP [48] was shown to be a critical property for ESNs to guarantee valid dynamics. From a dynamical system perspective, it guarantees that the reservoir states should asymptotically depend only on the driving input signal, regardless of their initial conditions [115]. Assuming the Euclidean distance metric in the reservoir space and tanh as the activation function, meeting the following condition practically ensures ESP:

$$\rho(\mathbf{W}_{\text{state}}) < 1, \quad (5.4)$$

where $\rho(\cdot)$ denotes the spectral radius (i.e., the largest absolute eigenvalue) of the given square matrix. In the context of dynamical systems, (5.4) also implies that $\mathbf{W}_{\text{state}}$ is contractive.

5.4 Learning-based Tracking Controller

The control scheme of the proposed learning-based tracking controller is illustrated in Fig. 5.1. It consists of a nominal controller² and an online residual learning module based on ESN, conceived to address non-parametric uncertainties. In the sequel, we elaborate on the working principles of the online residual learning module (Section 5.4.1) and the formulation of the learning-based control laws (Section 5.4.2).

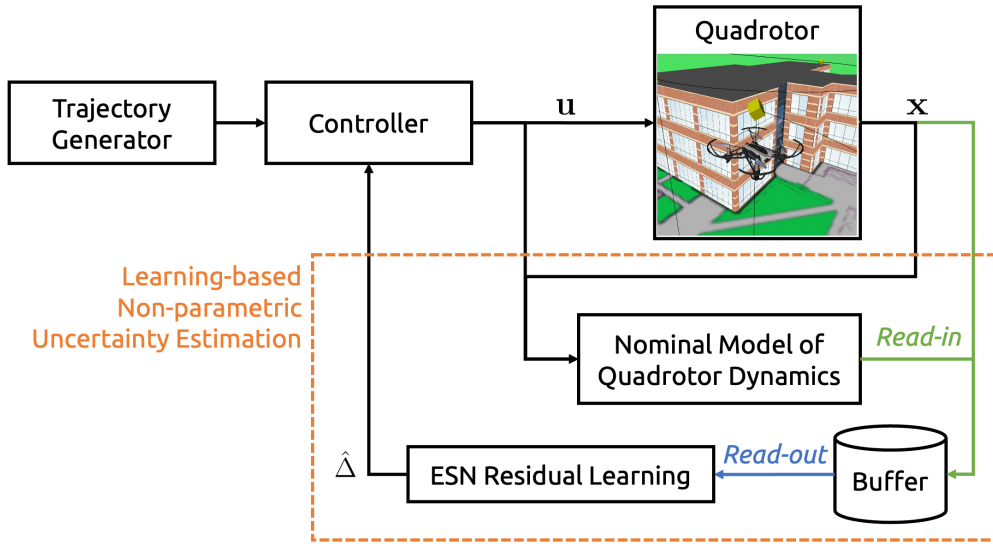


Fig. 5.1 **Control scheme of the proposed learning-based tracking controller.** The online learning module for non-parametric uncertainty estimation (framed by an orange dashed box) takes as inputs the online measurements of quadrotor states $\mathbf{x} = [\mathbf{p}^\top, \mathbf{v}^\top, \boldsymbol{\eta}^\top, \boldsymbol{\omega}^\top]^\top$ and control inputs $\mathbf{u} = [\mathbf{f}_u^\top, \boldsymbol{\tau}_u^\top]^\top$, and generates uncertainty estimates $\hat{\Delta} = [\hat{\mathbf{f}}_a^\top, \hat{\boldsymbol{\tau}}_a^\top]^\top$.

5.4.1 Online Learning Module for Residual Dynamics

To estimate the unknown and possibly dynamic uncertainties, an online residual learning module is designed with an ESN at its core. Endowed with online learning capability thanks to the efficient training of RC paradigm, such module is expected to “generalize” (or more precisely, actively adapt) better than the networks trained offline when facing unseen scenarios. Meanwhile, the online

²The nominal controller can be either model-based or learning-based; in our paper, we utilize a model-based approach.

residual learning module, as a standalone uncertainty estimator offers greater flexibility, compared to end-to-end data-driven control solutions such as [15], enabling seamless integration with existing controllers such as nominal ones.

The proposed online residual learning module is illustrated in Fig. 5.2. As depicted, the module comprises a *deep ESN*, described by (5.2), for learning and inferring uncertainties, and a *circular data buffer*, which stores the feature and label data from flight history in a sliding window fashion, facilitating efficient online training. The algorithm for online residual learning is outlined as follows:

- (i) **Initializing ESN:** We consider an ESN that takes system state³ as network input $\mathbf{u} = [p_z, \mathbf{v}^\top, \boldsymbol{\eta}^\top, \boldsymbol{\omega}^\top]^\top \in \mathbb{R}^{10}$ and predicted forces and torques as network output $\mathbf{y} = [\hat{\mathbf{f}}_a^\top, \hat{\boldsymbol{\tau}}_a^\top]^\top \in \mathbb{R}^6$. Besides configuring the ESN with designated numbers of input units (N_u), readout units (N_y), reservoir units (N_r), and reservoir layers (N_l), this step includes random initialization of all weight matrices (i.e., \mathbf{W}_{in} , \mathbf{W}_{il} , $\mathbf{W}_{\text{state}}$), adhering to the ESP.
- (ii) **Collecting flight data and computing label data:** For implementation purposes, the continuous-time dynamical model of the quadrotor (2.1) needs to be converted into its discrete-time counterpart with sampling time T_s as given in (5.5). Through the latter, we can calculate the label data at the previous time step from the current and previous state.

$$\mathbf{v}[k+1] = \mathbf{v}[k] + \frac{T_s}{m} (mg\mathbf{e}_3 + \mathbf{f}_u[k] + \mathbf{f}_a[k]), \quad (5.5a)$$

$$\boldsymbol{\omega}[k+1] = \boldsymbol{\omega}[k] + T_s \mathbf{J}^{-1} (-\boldsymbol{\omega}[k]^\wedge \mathbf{J} \boldsymbol{\omega}[k] + \boldsymbol{\tau}_u[k] + \boldsymbol{\tau}_a[k]). \quad (5.5b)$$

- (iii) **Updating data buffer:** The buffer retains the training data pair within the specified time span (denoted by N_b). Upon the arrival of a new data pair that exceeds the buffer size, the earliest stored data in the buffer is replaced. In other words, the buffer maintains data in a sliding window fashion, where the window size corresponds to the buffer size.
- (iv) **Training ESN and inferring from the latest flight data:** We use ridge regression for training the ESN. The training data pairs are drawn from the data buffer, and training initiates once the buffer is fully populated (hence,

³The altitude p_z is chosen as a network feature owing to its contribution to certain aerodynamic phenomena such as ground effect.

requiring an initial data collection period). The RC paradigm enables online training, specifically within our sampling time, as demonstrated empirically in Section 5.6.2. Upon completion of training, the ESN takes the latest flight data for inference.

Note that except for Step (i), which will be executed only once, Steps (ii) to (iv) will be iteratively performed during the flight. The pseudo-code of our implementation of online residual learning is provided as Algorithm 2. We leave the discussion of technical details on network architecture, ESP, and online running time to Section 5.6.

5.4.2 Tracking Control Laws

The decoupled nature of our architecture (as depicted in Fig. 5.1) allows to select any nominal controller, independently from the rest of the control structure. Here, we adopt the nominal controller in [105] as a starting point, whereby the tracking control laws for thrust and torque determination of (5.5) under nominal conditions are given by

$$\bar{\mathbf{f}}_u[k] = -\mathbf{K}_p \tilde{\mathbf{p}}[k] - \mathbf{K}_v \tilde{\mathbf{v}}[k] - mg\mathbf{e}_3 + m\dot{\mathbf{v}}_{\text{ref}}[k], \quad (5.6)$$

$$\bar{\boldsymbol{\tau}}_u[k] = -\mathbf{K}_r \tilde{\mathbf{r}}[k] - \mathbf{K}_\omega \tilde{\boldsymbol{\omega}}[k], \quad (5.7)$$

where the tracking errors of position, velocity, attitude, and angular rate are defined as

$$\tilde{\mathbf{p}}[k] = \mathbf{p}[k] - \mathbf{p}_{\text{ref}}[k], \quad (5.8a)$$

$$\tilde{\mathbf{v}}[k] = \mathbf{v}[k] - \mathbf{v}_{\text{ref}}[k], \quad (5.8b)$$

$$\tilde{\mathbf{r}}[k] = \frac{1}{2}(\mathbf{R}_{\text{ref}}^\top[k]\mathbf{R}[k] - \mathbf{R}^\top[k]\mathbf{R}_{\text{ref}}[k])^\vee, \quad (5.8c)$$

$$\tilde{\boldsymbol{\omega}}[k] = \boldsymbol{\omega}[k] - \boldsymbol{\omega}_{\text{ref}}[k], \quad (5.8d)$$

with $(\cdot)^\vee : \mathfrak{so}(3) \rightarrow \mathbb{R}^3$ denoting the vee operator and subscript “ref” denoting the desired reference. Moreover, the desired body z -axis can be computed from $\mathbf{k}_{b,d} = -\tilde{\mathbf{f}}_u / \|\tilde{\mathbf{f}}_u\| \in \mathbb{S}^2$, and hence the desired rotation matrix \mathbf{R}_{ref} can be derived

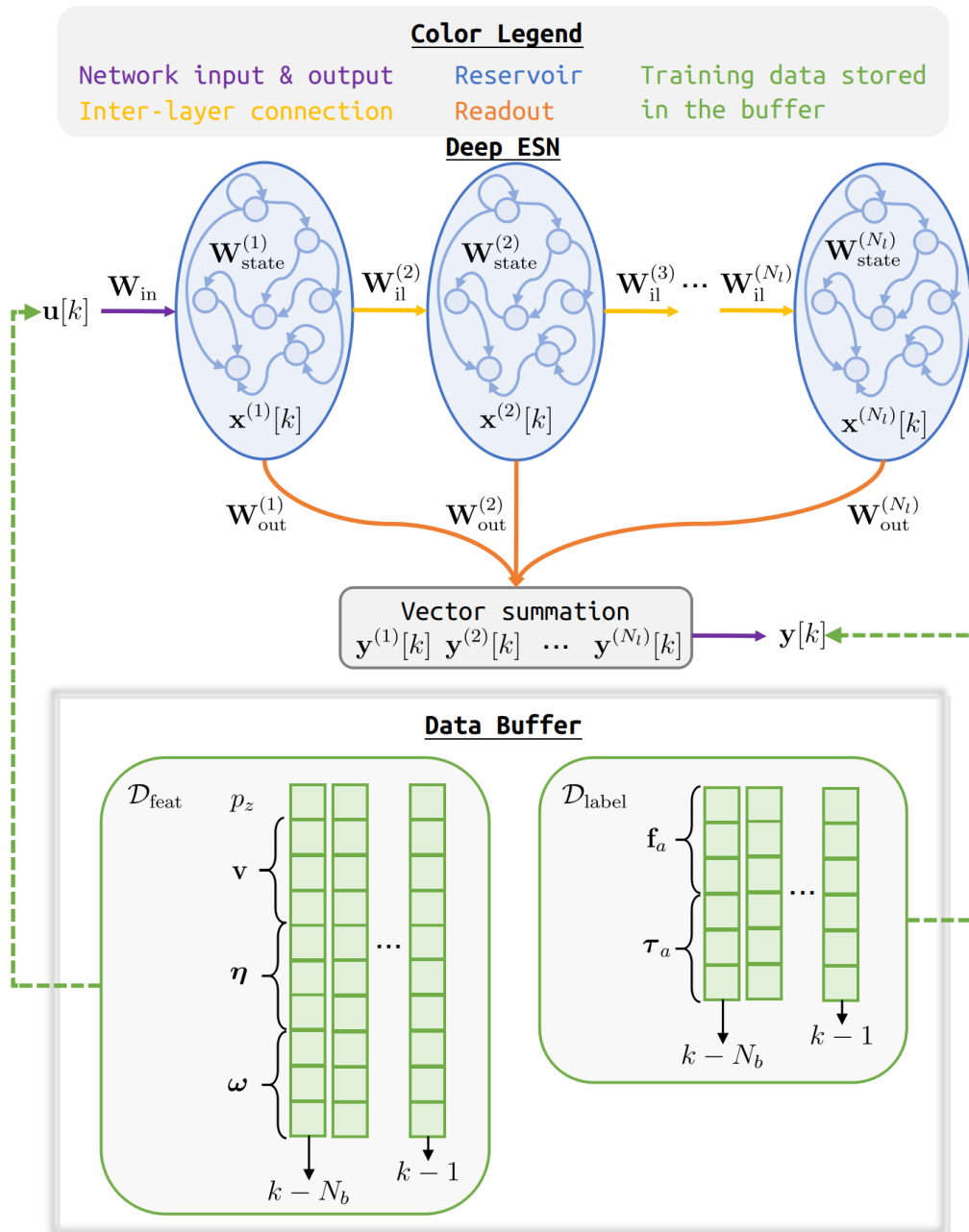


Fig. 5.2 **Structure of online residual learning module: Deep ESN and data buffer.** The data buffer of length N_b stores the feature and label data from flight history in a column-wise fashion (shown by green blocks), which are subsequently used for online training of readout weights of ESN (depicted in orange arrows). The ESN model adopts a hierarchical architecture for reservoir layers (shaded in blue) between which inter-layer connections exist (depicted in yellow arrows).

Algorithm 2: Online residual learning using ESN

Parameter: $N_b, \mathcal{E} = (N_r, N_l, N_u, N_y)$
Input: $\mathcal{X}[k] = (\mathbf{v}[k]^\top, \boldsymbol{\omega}[k]^\top, \mathbf{f}_u[k]^\top, \boldsymbol{\tau}_u[k]^\top)^\top, \mathcal{P} = (m, \mathbf{J}, T_s)$
Output: $\mathcal{Y}[k] = (\hat{\mathbf{f}}_a[k]^\top, \hat{\boldsymbol{\tau}}_a[k]^\top)^\top$
Data: $\mathcal{D}_{\text{feat}}, \mathcal{D}_{\text{label}}$ (from flight data)

- 1 **Function** CalculateLabel($\mathcal{X}[k], \mathcal{X}[k-1], \mathcal{P}$):
 - 2 Calculate $\mathbf{f}_a[k-1]$ from (5.5a)
 - 3 Calculate $\boldsymbol{\tau}_a[k-1]$ from (5.5b)
 - 4 **return** $\mathbf{f}_a[k-1], \boldsymbol{\tau}_a[k-1]$
- 5 **Function** UpdateBuffer($\mathcal{X}[k], N_b$):
 - 6 **if** $\mathcal{D}_{\text{feat}}, \mathcal{D}_{\text{label}}$ are undefined **then**
 - 7 $\mathcal{D}_{\text{feat}} = \mathcal{D}_{\text{label}} = []$ // initialization
 - 8 **if** $\mathcal{D}_{\text{feat}}$ has more than one column **then**
 - 9 $\mathbf{f}_a[k-1], \boldsymbol{\tau}_a[k-1] = \text{CalculateLabel}(\mathcal{X}[k], \mathcal{X}[k-1], \mathcal{P})$
 - 10 $\mathcal{D}_{\text{label}}.\text{append}(\mathbf{f}_a[k-1], \boldsymbol{\tau}_a[k-1])$
 - 11 **if** $\mathcal{D}_{\text{feat}}, \mathcal{D}_{\text{label}}$ have more than N_b columns **then**
 - 12 Delete the first column (i.e., the earliest data)
 - 13 $\mathcal{D}_{\text{feat}}.\text{append}(\mathcal{X}[k])$ // append column
 - 14 **return** $\mathcal{D}_{\text{feat}}, \mathcal{D}_{\text{label}}$
- 15 **Function** InitESN(\mathcal{E}):
 - 16 Instantiate and initialize an ESN using \mathcal{E}
 - 17 **return** net
- 18 **Function** TrainESN($net, \mathcal{D}_{\text{feat}}, \mathcal{D}_{\text{label}}$):
 - 19 Train net on $\mathcal{D}_{\text{feat}}, \mathcal{D}_{\text{label}}$ by solving (5.3)
 - 20 **return** 0
- 21 **Function** Main:
 - 22 Specify N_b and \mathcal{E}
 - 23 $net = \text{InitESN}(\mathcal{E})$
 - 24 $k \leftarrow 1$
 - 25 **while** non-stop **do**
 - 26 Get data $\mathcal{X}[k]$
 - 27 $\mathcal{D}_{\text{feat}}, \mathcal{D}_{\text{label}} = \text{UpdateBuffer}(\mathcal{X}[k], N_b)$
 - 28 **if** $\mathcal{D}_{\text{label}}$ has N_b columns **then**
 - 29 $\text{TrainESN}(net, \mathcal{D}_{\text{feat}}(:, 1:N_b), \mathcal{D}_{\text{label}})$
 - 30 $k \leftarrow k + 1$
 - 31 **return** 0

using the differential flatness property of quadrotors as

$$\mathbf{j}_c = [-\sin(\psi_{\text{ref}}), \cos(\psi_{\text{ref}}), 0]^\top, \quad (5.9a)$$

$$\mathbf{i}_{b,\text{ref}} = \mathbf{j}_c^\wedge \mathbf{k}_{b,\text{ref}} / \|\mathbf{j}_c^\wedge \mathbf{k}_{b,\text{ref}}\|, \quad (5.9b)$$

$$\mathbf{j}_{b,\text{ref}} = \mathbf{k}_{b,\text{ref}}^\wedge \mathbf{i}_{b,\text{ref}}, \quad (5.9c)$$

$$\mathbf{R}_{\text{ref}} = [\mathbf{i}_{b,\text{ref}}, \mathbf{j}_{b,\text{ref}}, \mathbf{k}_{b,\text{ref}}]. \quad (5.9d)$$

This nominal controller ensures the faithful tracking of given reference trajectories with proper selections of controller gains \mathbf{K}_Ξ ($\Xi \in \{p, v, r, \omega\}$) (see [105] for proofs).

Our proposed controller augments the nominal controller with the online residual learning module. Hence, the learning-based control laws for thrust and torque determination are given as

$$\mathbf{f}_u[k] = \bar{\mathbf{f}}_u[k] - \hat{\mathbf{f}}_a[k], \quad (5.10)$$

$$\boldsymbol{\tau}_u[k] = \bar{\boldsymbol{\tau}}_u[k] - \hat{\boldsymbol{\tau}}_a[k], \quad (5.11)$$

where $\hat{\mathbf{f}}_a$ and $\hat{\boldsymbol{\tau}}_a$ are the estimates of the uncertainties predicted by the ESN. The latter are derived as

$$\mathbf{y}[k] = \text{ESN}(\mathbf{u}[k] | \mathbf{W}_{\text{out}}[k]), \text{ where} \quad (5.12a)$$

$$\mathbf{y}[k] = [\hat{\mathbf{f}}_a^\top[k], \hat{\boldsymbol{\tau}}_a^\top[k]]^\top, \mathbf{u}[k] = [p_z[k], \mathbf{v}^\top[k], \boldsymbol{\eta}^\top[k], \boldsymbol{\omega}^\top[k]]^\top, \quad (5.12b)$$

upon completion of network training (through Algorithm 2) within each sampling interval. The desired body z -axis for constructing the desired rotation matrix is modified to $\mathbf{k}_{b,\text{ref}} = -\mathbf{f}_u / \|\mathbf{f}_u\|$, taking into account the compensation of unknown forces predicted by the ESN.

5.5 Post-hoc Analysis of Model Dynamics and Interpretability

Interpreting the outcome of ANNs, particularly deep networks, is crucial for establishing trustworthiness in their deployment in safety-critical systems. In pursuit of this goal, we aim to demystify the ESN from two perspectives. We first

analyze reservoir layers through the lens of dynamical systems. Then, we seek to elucidate the interpretations behind network predictions.

Goodness of reservoir dynamics: “Random” is nearly by definition an antonym to “optimal”, as perceptively pointed out in [114]. Hence, it is necessary to understand if the network with randomly initialized weights is *good* or not. In the ESN literature, various metrics assess the goodness of reservoir dynamics (i.e., (5.2a)), including ESP Index (ESPI), Linearly Uncoupled Dynamics (LUD) indicator, and condition number [116]. Here, we focus solely on ESPI, as the condition number pertains to network training with stochastic gradient descent, and LUD lacks a benchmark reference for determining the richness of reservoir dynamics. Motivated by ESP, the ESPI was introduced to gauge the robustness of reservoir layers to perturbations in initial conditions. Given N_{ic} randomly selected initial conditions, we can compute the ESPI of the l -th reservoir layer as

$$\text{ESPI}^{(l)} = \frac{1}{N_{ic}} \sum_{j=1}^{N_{ic}} \left(\frac{1}{N - N_w} \sum_{k=N_w+1}^N \|\mathbf{x}_0^{(l)}(t) - \mathbf{x}_j^{(l)}(t)\|_2 \right), \quad (5.13)$$

where $\mathbf{x}_0^{(l)}$ denotes the reservoir states obtained by initiating the reservoir from $\mathbf{0}$, $\mathbf{x}_j^{(l)}$ denotes the achieved reservoir states starting from randomly chosen initial conditions but driven by the same input, and N denotes the total time duration. ESPI quantifies the average deviation among reservoir state trajectories starting from different initial conditions under the same input signal. As illustrated in Fig. 5.3, regardless of initial conditions, reservoir layers that satisfy ESP shall exhibit asymptotic stable behavior after a washout period.

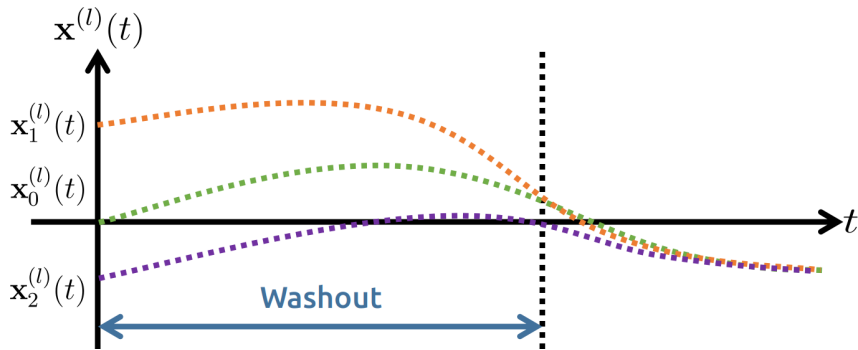


Fig. 5.3 Asymptotic stable behavior of reservoir layers ensured by ESP reflects the goodness of reservoir dynamics.

Interpretations for network predictions: Interpreting the predictions of a neural network often involves extracting information from its weights and internal states if available. Visualization serves as a direct and intuitive technique for exploratory data analysis. For example, when handling time-series data, the analysis of activations can be visualized using heat maps [113], while network weights can be analyzed through clustering methods such as k-means clustering and Gaussian Mixture Model (GMM). In our experiments, various visualization methods were explored, but not all proved effective. For example, plotting the heat maps of neuron activities is limited to small-size networks, which is not suitable for our deep ESN. Consequently, we adopt the t-SNE method with the Euclidean distance metric, as it allows for automatic determination of the number of clusters without a priori information. In this approach, all reservoir layers are considered as feature extraction for basis functions shared by uncertainties. The only trainable parameters are the readout weights, serving as a set of linear coefficients in (5.2b) updated for each specific condition. By applying t-SNE to visualize the readout weights over the time span $k = 1, \dots, N$, i.e.,

$$\mathbf{W}_{\text{out}} = \begin{pmatrix} \mathbf{W}_{\text{out}}^{(1)}[1] & \mathbf{W}_{\text{out}}^{(1)}[2] & \cdots & \mathbf{W}_{\text{out}}^{(1)}[N] \\ \mathbf{W}_{\text{out}}^{(2)}[1] & \mathbf{W}_{\text{out}}^{(2)}[2] & \cdots & \mathbf{W}_{\text{out}}^{(2)}[N] \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{W}_{\text{out}}^{(N_l)}[1] & \mathbf{W}_{\text{out}}^{(N_l)}[2] & \cdots & \mathbf{W}_{\text{out}}^{(N_l)}[N] \end{pmatrix} \in \mathbb{R}^{N_y \times (N_r N_l)}, \quad (5.14)$$

distinct clusters can be observed as shown in Section 5.6.4.

5.6 Results and Discussion

In this section, we present the simulation results of our proposed controller for quadrotor tracking tasks. We begin with elaborating on the design considerations behind the ESN structure and training, demonstrating its powerful approximation and real-time learning capability through experimentation with two synthetic datasets. Then, we conduct a comparative analysis between our learning-based controller and its nominal counterpart, emphasizing the effectiveness of the online residual learning module. Lastly, we compare our proposed controller with an offline learning-based approach, commonly employed in prior studies,

highlighting the superior performance of our online learning-based approach in addressing OOD uncertainty data.

5.6.1 Simulator

We constructed the simulation environment for evaluating our proposed controller using MATLAB scripts. This environment incorporates various components, including quadrotor forward dynamics, synthetic aerodynamics such as quadrotor ground effect [117], reference trajectory generator, training and inference modules of neural networks, control algorithms, auxiliary functions, and more.

5.6.2 Network Selection and Training

We considered an ESN with the parameters given in Table 5.1 to study its online learning capability. Specifically, all the weights were initialized randomly from a uniform distribution over $[-1, 1]$, and reservoir state weight matrix satisfied ESP by having the spectral radius $\rho = 0.9^4$.

| Model parameter | Selected value |
|---|--|
| Number of input/readout/reservoir units | $(N_u, N_y, N_r) = (10, 6, 20)$ |
| Number of reservoir layers | $N_l = 5$ |
| Input-to-reservoir weight | $\mathbf{W}_{\text{in}} \sim \mathcal{U}_{[-1,1]}$ |
| Inter-layer reservoir weight | $\mathbf{W}_{\text{il}} \sim \mathcal{U}_{[-1,1]}$ |
| Reservoir weight | $\mathbf{W}_{\text{state}} \sim \mathcal{U}$ with $\rho = 0.9$ |
| Leaking rate | $\alpha = 0$ |
| Washout | $N_w = 5$ |
| Regularization coefficient | $\lambda_r = 0.1$ |

Table 5.1 ESN parameters.

⁴For practitioners, here is a brief summary of “rule of thumbs” [116] for selecting proper network parameters: (i) Higher values of the leaking rate α result in reservoirs that respond more rapidly to the input, and higher values of the spectral radius $\rho (< 1)$ lead to richer dynamics; (ii) The number of reservoir layers N_l is one of the most critical parameters, as choosing a large value may induce instability in higher reservoir layers, compromising the adherence to ESP; (iii) In deeper ESN structure, the inter-layer scaling on \mathbf{W}_{il} (i.e., the connectivity strength) has a more pronounced impact on reservoir quality than input scaling.

To evaluate real-time training performance, we trained the ESN using Algorithm 2 on two simulated data sets collected at the sampling time $T_s = 0.02$ sec – one under periodic wind, and the other under quadrotor ground effect [117] – in MATLAB using the DeepESN class [118]. For each data set, we ran the training over 50 random seeds on a laptop with AMD Ryzen 7 5800h (no GPU involved). Consistent and accurate predictions on quadrotor ground effect and periodic wind were observed in Fig. 5.4(a) and Fig. 5.4(b), respectively. Note that there was an initial period in both cases where the ESN did not make any predictions, waiting for the data buffer to be filled with N_b samples. We further report the training time of ESN in Fig. 5.4(c). Notably, the average network training time for both cases is considerably less than our chosen sampling time $T_s = 0.02$ sec, demonstrating the feasibility of real-time training.

5.6.3 Flight Control with Online Learning

Next, we evaluated our proposed controller with online residual learning (i.e., (5.10) and (5.11)) by commanding the quadrotor to track a figure-8 trajectory: $\mathbf{p}_d(t) = [\cos(0.5t), \sin(t), -5]^\top$, with disturbances $\mathbf{f}_a(t) = [5, 3, 0]^\top$, $\boldsymbol{\tau}_a(t) = [2 \sin(2\pi t), -2 \sin(2\pi t), 0]^\top$. We considered $m = 2.95$ kg, $\mathbf{J} = \text{diag}([0.5, 0.5, 0.5])$ kgm² and $\mathbf{p}(0) = [0, 0, -5]^\top$, $\mathbf{v}(0) = \boldsymbol{\eta}(0) = \boldsymbol{\omega}(0) = \mathbf{0}_{3 \times 1}$ as initial conditions. Same parameters in Table 5.1 were used for the ESN with all reservoir states initialized from zeros. Simulation and sampling time were chosen as 20 sec and 0.02 sec, respectively. Buffer size was selected as $N_b = 20$.

Our proposed controller demonstrated satisfactory tracking performance, as observed in Fig. 5.5(c). A comparison with the nominal controller [105] (i.e., (5.6) and (5.7), without online residual learning) is illustrated in Fig. 5.5(a). It can be seen that the nominal controller failed in asymptotic tracking of the reference trajectory under disturbances, exhibiting a large deviation due to the wind gust as depicted in blue arrows. Our proposed controller, however, deviated slightly at the beginning of the simulation, yet rapidly converged to the desired trajectory thanks to its online learning capability. The comparison of commanded control inputs shown in Fig. 5.5(b) further demonstrates the vital role that ESN played in our proposed controller on the fly, compensating for uncertainties and guaranteeing asymptotic tracking performance. The small oscillations observed in τ_x and τ_y are attributed to the presence of uncertainties in torque.

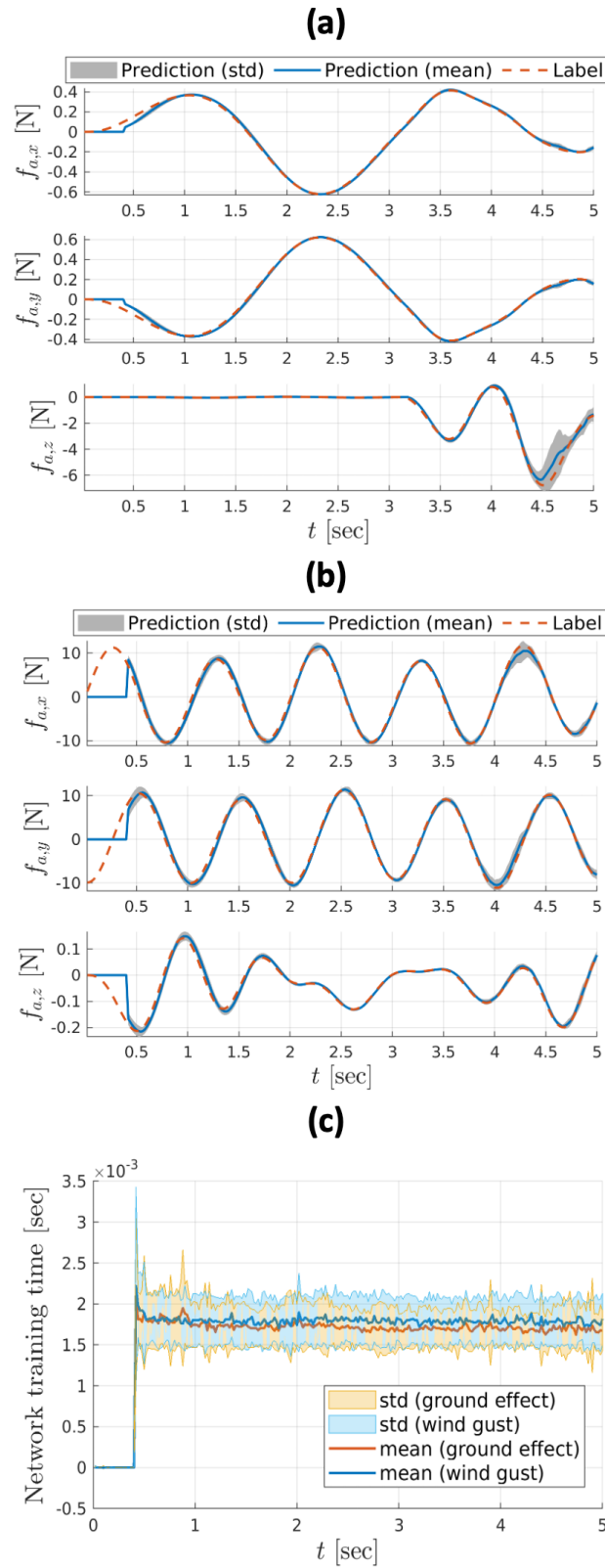


Fig. 5.4 Online learning performance over 50 simulation campaigns. (a) Predictions of quadrotor ground effect. (b) Predictions of periodic wind. (c) Training time of ESN for learning ground effect and periodic wind.

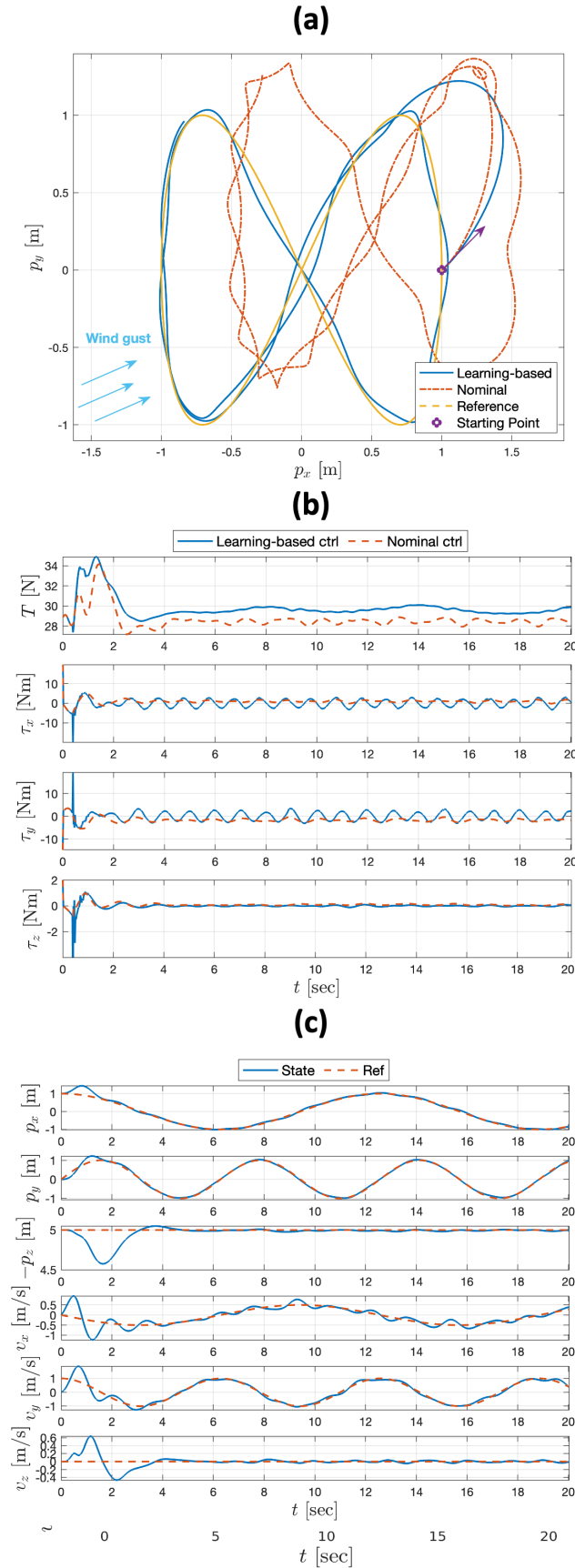


Fig. 5.5 Comparison of control behavior between our proposed learning-based controller and a nominal controller. (a) Tracking performance for executing a figure-8 maneuver (viewed in 2D). (b) Control input. (c) Position and velocity.

Furthermore, we assessed ESPI to verify the proper initialization of all reservoirs, ensuring adherence to ESP. This can also be evaluated by the performance of our proposed controller under different initial conditions of the ESN through a simulation campaign with 20 rounds initialized with random seeds. The results are reported in Table 5.2 where bold font highlights the best performance. It is straightforward to see that our proposed controller outperforms the nominal one, especially in x - and y -axis, since the wind gust considered in this case is merely horizontal. Moreover, our method demonstrates a comparable standard deviation, affirming consistent performance regardless of network initialization⁵.

5.6.4 Comparison with an Offline Learning-based Controller

Lastly, we demonstrate the superiority of online learning by comparing our proposed controller with an offline-trained FNN controller. To this end, we considered a more intricate wind profile, comprising the summation of two Gaussian functions, given as

$$f_{\text{wind}}(a, \mu, \sigma) = a \exp\left(-\frac{(t - \mu)^2}{2\sigma^2}\right), \quad (5.15)$$

$$\mathbf{f}_a(t) = \underbrace{[f_{\text{wind}}(35, 3, 10) + f_{\text{wind}}(20, 5, 5), *, 0]}_{(*)}. \quad (5.16)$$

The quadrotor was commanded to execute the same figure-8 maneuver as in Section 5.6.3.

We implemented a cascaded feedforward network with 5 layers, each containing 20 hidden neurons, to be consistent with our ESN's parameters for making fair comparisons. The training of this FNN was conducted completely offline using the Levenberg–Marquardt (LM) algorithm, and early stopping was used to prevent overfitting. We utilized the data from 0-28 sec as training data, keeping the rest until 50 sec for testing. Notably, from Fig. 5.6(a)-(b), it can be seen that test data consist of both In-Distribution (ID) data, spanning 28-31 sec and 39-50 sec (shaded in blue), and OOD data, spanning 31-39 sec (shaded in red). The distribution shift lies in both network inputs (see Euler angles in Fig. 5.6(a))

⁵Standard deviation is excluded for the nominal case due to the absence of randomness.

and predictions (see the aerodynamic forces over the horizontal dotted line in Fig. 5.6(b)).

As a result, FNN demonstrated poor generalization capability for OOD data, leading to high prediction errors. Incorrect predictions further affected the effectiveness of the controller, yielding unacceptable tracking performance in OOD regime; see Fig. 5.6(c) for 3-D illustration and Fig. 5.6(d) for position and velocity. In contrast, our proposed learning-based controller, leveraging the online learning capability of ESN, showed promising tracking results. Consistent control behavior was also observed across different reservoir state initialization, as reported in Table 5.2.

| Controller | $\tilde{\mathbf{p}}_x$ [m] | | $\tilde{\mathbf{p}}_y$ [m] | | $\tilde{\mathbf{p}}_z$ [m] | |
|----------------------|----------------------------|--------|----------------------------|--------|----------------------------|--------|
| | MAE | Std. | MAE | Std. | MAE | Std. |
| Nominal ([105]) | 0.4951 | n/a | 0.2969 | n/a | 0.0342 | n/a |
| Learning-based (ESN) | 0.0387 | 0.0056 | 0.0277 | 0.0032 | 0.0096 | 0.0018 |
| Learning-based (FNN) | 0.0905 | n/a | 0.0842 | n/a | 0.0060 | n/a |
| Learning-based (ESN) | 0.0036 | 0.0011 | 0.0167 | 0.0015 | 0.0028 | 0.0008 |

Table 5.2 **Tracking errors of three different controllers over 20 simulation campaigns.** Controllers include: (i) A nominal controller, (ii) a learning-based controller with offline-trained FNN, and (iii) our proposed learning-based controller with online residual learning using ESN.

Furthermore, we analyzed the information extracted from the readout weights (\mathbf{W}_{out} as defined in (5.14)) and reservoir states ($\mathbf{X}^{(l)}$ as defined in (5.3)) of the ESN using the t-SNE method. Notably, distinct clusters for different wind magnitudes (indicated in the legend with the unit m/sec) are evident for readout weights (see Fig. 5.7(a)). Additionally, the coarser the division in wind magnitude, the more distinct the clusters. In contrast, no clear clusters were observed for reservoir states, regardless of the layer (see Fig. 5.7(b)-(f) for the first to the fifth layer). This aligns with our hypothesis that reservoir layers learn shared features of uncertainties, while the last readout layer represents each specific condition. Hence, analyzing the readout weight matrix allows us to infer information about wind magnitude.

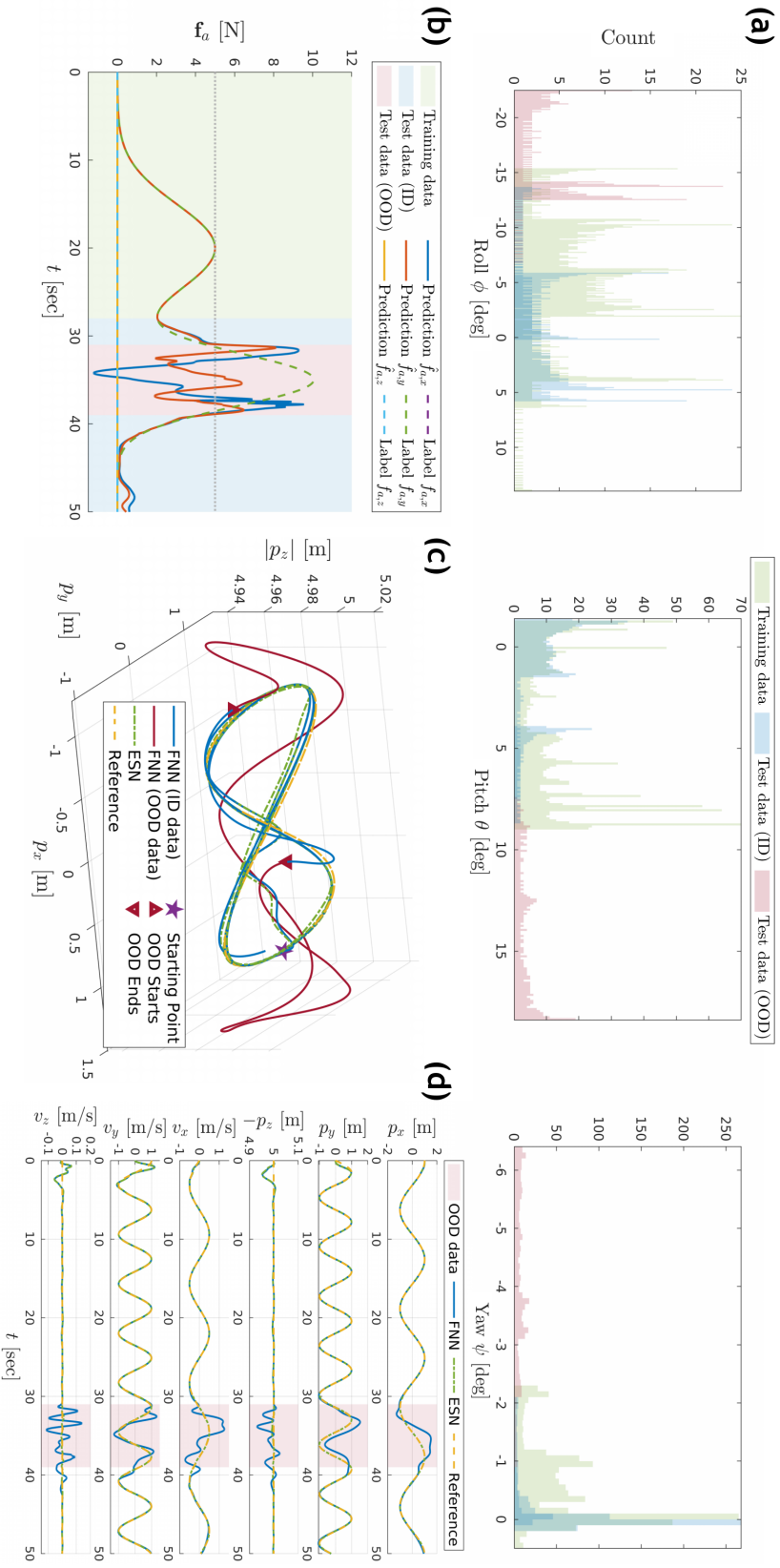


Fig. 5.6 Comparison of control behavior between our proposed learning-based controller (ESN) and an offline learning-based controller (FNN). (a) Distribution shift in Euler angles (i.e., network inputs). (b) Distribution shift in network predictions and the learning performance of FNN. (c) Tracking performance for executing a figure-8 maneuver. (d) Position and velocity.

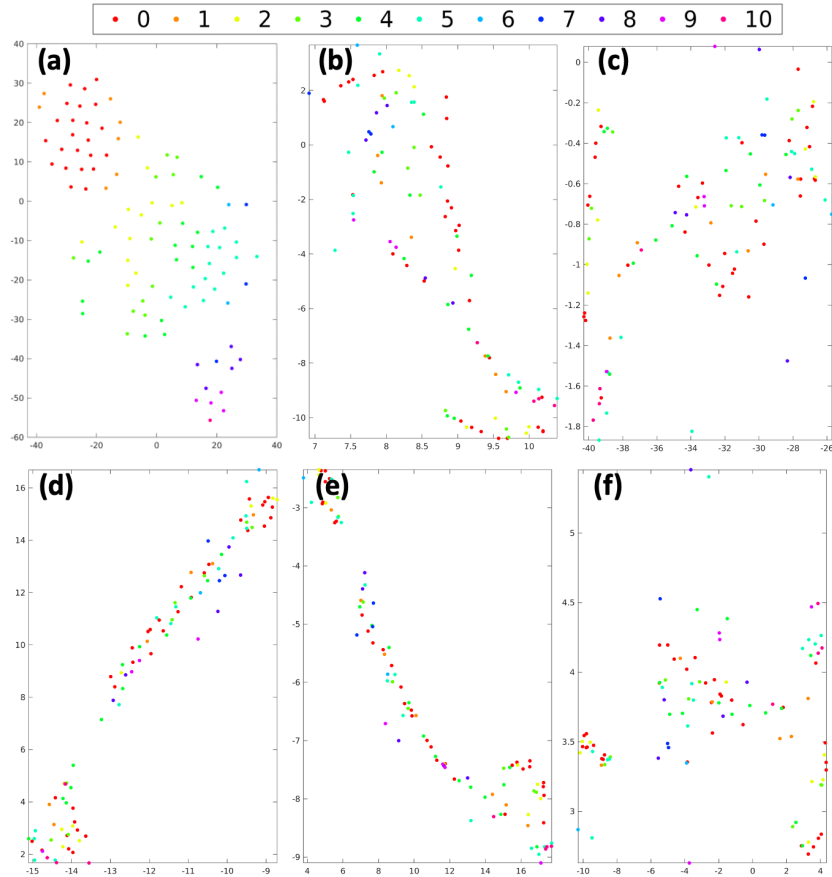


Fig. 5.7 **Post-hoc visualization using t-SNE.** (a) Clustering of readout weight matrix. (b)-(f) Clustering of reservoir states.

5.7 Conclusions

In this chapter, we proposed a novel control architecture, comprising a nominal controller with an online residual learning module based on the RC paradigm for quadrotor tracking control under non-parametric uncertainties. Our approach excels in dynamically learning the unknowns on the fly, thereby extending its applicability to a broader operating range. Moreover, we addressed the challenge of enhancing trustworthiness inherent in most of the learning-based approaches through tailored post-hoc interpretation techniques designed for ESN. This involves analyzing reservoir dynamics and seeking interpretations for network predictions. Our simulations demonstrate the feasibility of online learning, showcasing superior performance over both a nominal and an offline-trained learning-based controller.

As for future work, we aim to extend the current results to handle noisy measurement data, which could pose difficulties for network interpretation. To address this, we will explore alternative clustering methods that, ideally, do not necessitate a specified number of clusters, support arbitrarily shaped clusters, prove effective for outlier detection, and provide deterministic results. Furthermore, we aim to devise novel control algorithms that utilize interpretable information to expedite adaptation and enhance context-awareness of the robot operating in unknown scenarios.

Chapter 6

Concluding Remarks and Future Work

The growing recognition of trustworthiness as a crucial factor for deploying deep learning and other learning-based approaches in safety-critical systems has led to a demand for revolutionary advancements in modern black-box solutions. While a systematic framework is currently lacking, transparent models and post-hoc analysis have emerged as two promising avenues to address this challenge.

Transparent models offer the potential to unveil the inner workings of black-box systems by integrating data with prior knowledge. These models exhibit enhanced learning capabilities, requiring less data, while achieving improved generalizability and interpretability compared to their black-box counterparts. Conversely, post-hoc analysis provides alternative means for gaining insights into models that are inherently opaque. Techniques included within this category can be found such as model simplification, feature relevance assessment, and visualization.

In the specific domain of UAV modeling and control, numerous open problems persist, which are outlined as follows:

- (i) **The need of online learning or adaptation:** While the majority of existing efforts in the field of UAVs focus on offline solutions, there is a pressing need to develop algorithms that enable UAVs to continuously learn and adapt in real-time. This capability is essential for effectively navigating dynamic environments and tackling evolving tasks. However, several challenges must be addressed to achieve this goal, including the paucity of representative

data and the limited computational resources available onboard for training purposes.

- (ii) **Addressing the "chicken or egg" problem:** In the context of PINN modeling or control, a challenging issue arises, which we call the "chicken and egg problem." This problem arises when we aim to utilize PINNs to model unknown and complex dynamics by leveraging their data-driven characteristics, i.e., their strong approximation capabilities. However, in order to incorporate physics into the data-driven model, additional prior knowledge needs to be embedded through appropriate means, e.g., either by introducing an inductive bias or a learning bias. This challenge stems from the difficulty of finding suitable knowledge for the unknowns we intend to model using PINNs. The key concern is that we require known information that cannot be explicitly expressed using mathematical expressions, as is typically done with MBC techniques. This creates a dilemma in finding the necessary prior knowledge that can enhance the physics-informed aspect of the PINN solutions.
- (iii) **Development of a systematic framework for post-hoc analysis:** While several post-hoc analysis techniques are available, there is still a lack of a systematic framework to perform effective and insightful analysis. For example, a specific challenge in this regard is determining which parameters of a DNN should be subjected to analysis. Given the complex structure with numerous layers and parameters in a DNN, selecting representative features becomes challenging due to the opacity of these models. To address this issue, there is a crucial need for both theoretical and empirical studies to explore this topic and provide guidance for conducting meaningful post-hoc analysis. In terms of UAV applications, it is highly desirable to extract relevant information that offers insights into the behavior and performance of the system, e.g., the system's state or high-level situation awareness. By achieving this, transparency and interpretability of learning-based UAV systems can be significantly enhanced.
- (iv) **Leveraging post-hoc information effectively:** In addition to enhancing transparency and interpretability, one of the most intriguing benefits of post-hoc analysis is the potential for informing the design process by leveraging the obtained insights. This raises the important question of how we can

effectively exploit these insights to improve controllers and overall system performance in UAV applications. For example, by analyzing the results of post-hoc analysis, we can identify areas that require improvement and make informed decisions to enhance various aspects of UAV systems. This may involve refining controller designs, optimizing control strategies, and fine-tuning parameters based on the patterns and behaviors identified through post-hoc analysis. By leveraging these insights, we can fully harness the potential of post-hoc analysis and achieve tangible benefits such as improved system performance, more effective decision-making processes, and a deeper understanding of the underlying dynamics and behaviors of UAVs.

- (v) **Real-world algorithm validation for complex tasks:** Real-world validation plays a crucial role in the development and deployment of UAV systems. While simulators are valuable tools for initial testing and validation, they have inherent limitations in accurately representing the complexities of real-world environments, especially for complex tasks. These limitations give rise to the sim-to-real gap, where algorithms and designs that perform well in simulation may not perform as expected in real-world scenarios. Along with the development of trustworthy data-driven modeling and control algorithms, allocating resources and efforts to real-world validation is essential for building trustworthy UAV systems.

By addressing these open problems, significant advancements can be achieved, leading to the development of safer, more reliable, and more interpretable UAV control systems. In particular, enhancing the interpretability of data-driven modeling and control algorithms is crucial for building trust and facilitating human understanding. Interpretable systems allow operators and stakeholders to comprehend the decision-making processes and reasoning behind system actions. This transparency is extremely important in safety-critical systems.

References

- [1] Mostafa Hassanalian and Abdessattar Abdelkefi. Classifications, applications, and design challenges of drones: A review. *Progress in Aerospace Sciences*, 91:99–131, 2017.
- [2] Chenyao Wang, Su Wang, Guido de Croon, and Salua Hamaza. Embodied airflow sensing for improved in-gust flight of flapping wing MAVs. *Frontiers in Robotics and AI*, 9, 2022.
- [3] Zhiwei Zhang, Yuhang Zhong, Junlong Guo, Qianhao Wang, Chao Xu, and Fei Gao. Auto Filmer: Autonomous aerial videography under human interaction. *IEEE Robotics and Automation Letters*, 8(2):784–791, 2023.
- [4] Jun Zeng, Prasanth Kotaru, and Koushil Sreenath. Geometric control and differential flatness of a quadrotor UAV with load suspended from a pulley. *2019 American Control Conference (ACC)*, pages 2420–2427, 2019.
- [5] Weibin Gu, Dewen Hu, Liang Cheng, Yabing Cao, Alessandro Rizzo, and Kimon P. Valavanis. Autonomous wind turbine inspection using a quadrotor. *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 709–715, 2020.
- [6] Jacob R. Goodman, Juan S. Cely, Thomas Beckers, and Leonardo Jesus Colombo. Geometric control for load transportation with quadrotor UAVs by elastic cables. *ArXiv*, abs/2111.00777, 2021.
- [7] Kenta Takaya, Hiroshi Ohta, Keishi Shibayama, and Valeri Kroumov. Tracking control of unmanned aerial vehicle for power line inspection. *Motion Planning [Working Title]*, 2021.
- [8] Chao Huang, Zhenxing Ming, and Hailong Huang. Drone stations-aided beyond-battery-lifetime flight planning for parcel delivery. *IEEE Transactions on Automation Science and Engineering*, 20(4):2294–2304, 2023.
- [9] Nihal Dalwadi, Dipankar Deb, and S. M. Muyeen. Adaptive backstepping controller design of quadrotor biplane for payload delivery. *IET Intelligent Transport Systems*, 2022.

-
- [10] Weibin Gu, Kimon P. Valavanis, Matthew J. Rutherford, and Alessandro Rizzo. UAV model-based flight control with artificial neural networks: A survey. *Journal of Intelligent & Robotic Systems*, 100:1469 – 1491, 2020.
- [11] Dario Brescianini, Markus Hehn, and Raffaello D’Andrea. Quadcopter pole acrobatics. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3472–3479, 2013.
- [12] Elia Kaufmann, Antonio Loquercio, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. *ArXiv*, abs/2006.05768, 2020.
- [13] Simone Martini, Margareta Stefanovic, Alessandro Rizzo, Matthew J. Rutherford, Patrizia Livreri, and Kimon P. Valavanis. A benchmark framework for testing, evaluation, and comparison of quadrotor linear and nonlinear controllers. *2023 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 471–478, 2023.
- [14] Jessica Alvarenga, Nikolaos I Vitzilaios, Kimon P Valavanis, and Matthew J Rutherford. Survey of unmanned helicopter model-based navigation and control techniques. *Journal of Intelligent & Robotic Systems*, 80(1):87–138, 2015.
- [15] Elia Kaufmann, Leonard Bauersfeld, Antonio Loquercio, Matthias Mueller, Vladlen Koltun, and Davide Scaramuzza. Champion-level drone racing using deep reinforcement learning. *Nature*, 620:982–987, 08 2023.
- [16] Leonard Bauersfeld, Elia Kaufmann, Philipp Foehn, Sihao Sun, and Davide Scaramuzza. NeuroBEM: Hybrid aerodynamic quadrotor model. *ArXiv*, abs/2106.08015, 2021.
- [17] G.M. Hoffmann, Haomiao Huang, Steven L. Waslander, and Claire J. Tomlin. Precision flight control for a multi-vehicle quadrotor helicopter testbed. *Control Engineering Practice*, 19:1023–1036, 2011.
- [18] Petros A. Ioannou and Jing Sun. Robust adaptive control. 2012.
- [19] Tsung-Wei Ou and Yen-Chen Liu. Adaptive backstepping tracking control for quadrotor aerial robots subject to uncertain dynamics. *2019 American Control Conference (ACC)*, pages 1–6, 2019.
- [20] Tinashe Chingozha and Otis T. Nyandoro. Adaptive sliding backstepping control of quadrotor UAV attitude. *IFAC Proceedings Volumes*, 47:11043–11048, 2014.
- [21] Jingqing Han. From PID to active disturbance rejection control. *IEEE Trans. Ind. Electron.*, 56:900–906, 2009.
- [22] Julian Berberich, Johannes Köhler, Matthias A. Müller, and Frank Allgöwer. Linear tracking MPC for nonlinear systems—part II: The data-driven case. *IEEE Transactions on Automatic Control*, 67(9):4406–4421, 2022.

- [23] KM Hornik, M Stinchcomb, and H White. Multilayer feedforward networks are universal approximator. *IEEE Transactions on Neural Networks*, 2, 01 1989.
- [24] Franco Scarselli and Ah Chung Tsoi. Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural Networks*, 11(1):15–37, 1998.
- [25] Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *Artificial Neural Networks–ICANN 2006: 16th International Conference, Athens, Greece, September 10–14, 2006. Proceedings, Part I 16*, pages 632–640. Springer, 2006.
- [26] Guanya Shi, Xichen Shi, Michael O’Connell, Rose Yu, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural Lander: Stable drone landing control using learned dynamics. *2019 International Conference on Robotics and Automation (ICRA)*, pages 9784–9790, 2019.
- [27] Guanya Shi, Wolfgang Honig, Xichen Shi, Yisong Yue, and Soon-Jo Chung. Neural-Swarm2: Planning and control of heterogeneous multirotor swarms using learned interactions. *ArXiv*, abs/2012.05457, 2020.
- [28] Michael O’Connell, Guanya Shi, Xichen Shi, Kamyar Azizzadenesheli, Anima Anandkumar, Yisong Yue, and Soon-Jo Chung. Neural-Fly enables rapid learning for agile flight in strong winds. *Science Robotics*, 7, 2022.
- [29] Alejandro Barredo Arrieta, Natalia Díaz Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, A. Barbado, Salvador García, Sergio Gil-Lopez, Daniel Molina, Richard Benjamins, Raja Chatila, and Francisco Herrera. Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *ArXiv*, abs/1910.10045, 2019.
- [30] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. 2021.
- [31] Weibin Gu, Stefano Primatesta, and Alessandro Rizzo. Physics-informed neural network for quadrotor dynamical modeling. *Robotics and Autonomous Systems*, 2023.
- [32] Weibin Gu, Stefano Primatesta, and Alessandro Rizzo. Robust adaptive tracking control on SE(3) for quadrotor aggressive maneuvers. *IEEE Transactions on Automation Science and Engineering*, 2024 (Under Review).
- [33] Weibin Gu, , and Alessandro Rizzo. Online residual learning using interpretable reservoir computing for quadrotor control. *2024 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2024 (In Press).
- [34] Johan Ernest Mebius. Derivation of the Euler-Rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations. *arXiv: General Mathematics*, 2007.

-
- [35] T.D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.
- [36] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of mathematical biology*, 52 1-2:99–115; discussion 73–97, 1990.
- [37] Frank F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.
- [38] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [39] Geoffrey E. Hinton and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313 5786:504–7, 2006.
- [40] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks: the official journal of the International Neural Network Society*, 61:85–117, 2015.
- [41] Mark Hudson Beale, Martin T Hagan, and Howard B Demuth. Neural network toolbox™ user’s guide. *The Mathworks Inc*, 1992.
- [42] David S. Broomhead and David Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2, 1988.
- [43] Qinghua Zhang and Albert Benveniste. Wavelet networks. *IEEE transactions on neural networks*, 3 6:889–98, 1992.
- [44] Richard H. R. Hahnloser, Rahul Sarpeshkar, Misha A. Mahowald, Rodney J. Douglas, and H. Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405:947–951, 2000.
- [45] James S. Albus. New approach to manipulator control: The cerebellar model articulation controller (CMAC). 1975.
- [46] John Joseph Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79 8:2554–8, 1982.
- [47] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.
- [48] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks. 2001.
- [49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

-
- [50] Sepp Hochreiter and Yoshua Bengio. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. 2001.
- [51] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *ArXiv*, abs/1409.1259, 2014.
- [52] Alan Lloyd Hodgkin and Andrew Fielding Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117 4:500–44, 1952.
- [53] Wulfram Gerstner and Werner M. Kistler. Spiking neuron models: Single neurons, populations, plasticity. 2002.
- [54] Paul J. Werbos. Backpropagation through time: What it does and how to do it. 1990.
- [55] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.
- [56] Hazim Shakhatreh, Ahmad H. Sawalmeh, Ala Al-Fuqaha, Zuochoao Dou, Eyad K. Almaita, Issa M. Khalil, Noor Shamsiah Othman, Abdallah Khreishah, and Mohsen Guizani. Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges. *IEEE Access*, 7:48572–48634, 2019.
- [57] Ali Punjani and Pieter Abbeel. Deep learning helicopter dynamics models. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3223–3230, 2015.
- [58] Somil Bansal, Anayo K. Akametalu, Frank J. Jiang, Forrest Laine, and Claire J. Tomlin. Learning quadrotor dynamics using neural network for flight control. *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 4653–4660, 2016.
- [59] Jesse Callanan, Rayhaan Iqbal, Revant Adlakha, Amir Behjat, Souma Chowdhury, and Mostafa Nouh. Large-aperture experimental characterization of the acoustic field generated by a hovering unmanned aerial vehicle. *The Journal of the Acoustical Society of America*, 150 3:2046, 2021.
- [60] Rayhaan Iqbal, Amir Behjat, Revant Adlakha, Jesse Callanan, Mostafa Nouh, and Souma Chowdhury. Efficient training of transfer mapping in physics-infused machine learning models of UAV acoustic field. *ArXiv*, abs/2201.06090, 2022.
- [61] Rayhaan Iqbal, Amir Behjat, Revant Adlakha, Jesse Callanan, Mostafa Nouh, and Souma Chowdhury. Auto-differentiable transfer mapping architecture for physics-infused learning of acoustic field. *IEEE Transactions on Artificial Intelligence*, 2023.

- [62] Michael Lutter, Christian Ritter, and Jan Peters. Deep lagrangian networks: Using physics as model prior for deep learning. *ArXiv*, abs/1907.04490, 2019.
- [63] Jayesh K. Gupta, Kunal Menda, Zachary Manchester, and Mykel J. Kochenderfer. A general framework for structured learning of mechanical systems. *ArXiv*, abs/1902.08705, 2019.
- [64] Michael Lutter and Jan Peters. Combining physics and deep learning to learn continuous-time dynamics models. *ArXiv*, abs/2110.01894, 2021.
- [65] Anuj Karpatne, William Watkins, Jordan S. Read, and Vipin Kumar. Physics-guided neural networks (PGNN): An application in lake temperature modeling. *ArXiv*, abs/1710.11431, 2017.
- [66] Sagi Shaier, Maziar Raissi, and Padmanabhan Seshaiyer. Data-driven approaches for predicting spread of infectious diseases through DINNs: Disease informed neural networks. 2021.
- [67] S. Shah, Debadeepta Dey, C. Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *FSR*, 2017.
- [68] Carsten Schelp. An alternative way to plot the covariance ellipse. https://carstenschelp.github.io/2018/09/14/Plot_Confidence_Ellipse_001.html, 2018.
- [69] J. Enrique Sierra and Matilde Santos Peñas. Modelling engineering systems using analytical and neural techniques: Hybridization. *Neurocomputing*, 271:70–83, 2018.
- [70] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.
- [71] Sam Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. *ArXiv*, abs/1906.01563, 2019.
- [72] Yaofeng Desmond Zhong, Biswadip Dey, and Amit Chakraborty. Symplectic ODE-Net: Learning Hamiltonian dynamics with control. *ArXiv*, abs/1909.12077, 2020.
- [73] Anuj Karpatne, Gowtham Atluri, James H. Faghmous, Michael S. Steinbach, Arindam Banerjee, Auroop Ratan Ganguly, Shashi Shekhar, Nagiza F. Samatova, and Vipin Kumar. Theory-guided data science: A new paradigm for scientific discovery from data. *IEEE Transactions on Knowledge and Data Engineering*, 29:2318–2331, 2017.

- [74] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigating KL vanishing. In *NAACL*, 2019.
- [75] Lorenz Meier, Dominik Honegger, and M. Pollefeys. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240, 2015.
- [76] Nathan P. Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2149–2154 vol.3, 2004.
- [77] I. C. Cheeseman, Ph. D, W. E. Bennett, Ph. D, and W. E. Bennett. The effect of ground on a helicopter rotor in forward flight, 1955.
- [78] Stephen A. Conyers, Matthew J. Rutherford, and Kimon P. Valavanis. An empirical evaluation of ground effect for small-scale rotorcraft. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1244–1250, 2018.
- [79] Girish Joshi, Jasvir Viridi, and Girish V. Chowdhary. Asynchronous deep model reference adaptive control. *ArXiv*, abs/2011.02920, 2020.
- [80] Xinyue Kan, Justin R. Thomas, Hanzhe Teng, Herbert G. Tanner, Vijay R. Kumar, and Konstantinos Karydis. Analysis of ground effect for small-scale UAVs in forward flight. *IEEE Robotics and Automation Letters*, 4:3860–3867, 2019.
- [81] Yuhong Hou, Dengkai Chen, and Shuming Yang. Adaptive robust trajectory tracking controller for a quadrotor UAV with uncertain environment parameters based on backstepping sliding mode method. *IEEE Transactions on Automation Science and Engineering*, 2023.
- [82] Xiangyu Shao, Guanghui Sun, Weiran Yao, Jianxing Liu, and Ligang Wu. Adaptive sliding mode control for quadrotor UAVs with input saturation. *IEEE/ASME Transactions on Mechatronics*, 27(3):1498–1509, 2022.
- [83] A. R. P. Andriën, Demy Kremers, Dave Kooijman, and Duarte J. Antunes. Model predictive tracking controller for quadcopters with setpoint convergence guarantees. *2020 American Control Conference (ACC)*, pages 3205–3210, 2020.
- [84] Xuetao Zhang, Yan Zhuang, Xuebo Zhang, and Yongchun Fang. A novel asymptotic robust tracking control strategy for rotorcraft UAVs. *IEEE Transactions on Automation Science and Engineering*, 20(4):2338–2349, 2023.
- [85] John Stuelpnagel. On the parametrization of the three-dimensional rotation group. *Siam Review*, 6:422–430, 1964.

- [86] Sanjay P. Bhat and Dennis S. Bernstein. A topological obstruction to continuous global stabilization of rotational motion and the unwinding phenomenon. *Systems & Control Letters*, 39:63–70, 2000.
- [87] Evan G. Hemingway and Oliver M. O’Reilly. Perspectives on Euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments. *Multibody System Dynamics*, 44:31–56, 2018.
- [88] Christopher G. Mayhew, Ricardo G. Sanfelice, and Andrew R. Teel. On quaternion-based attitude control and the unwinding phenomenon. *Proceedings of the 2011 American Control Conference*, pages 299–304, 2011.
- [89] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). *49th IEEE Conference on Decision and Control (CDC)*, pages 5420–5425, 2010.
- [90] Taeyoung Lee. Geometric tracking control of the attitude dynamics of a rigid body on SO(3). *Proceedings of the 2011 American Control Conference*, pages 1200–1205, 2010.
- [91] Farhad A. Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric nonlinear PID control of a quadrotor UAV on SE(3). *2013 European Control Conference (ECC)*, pages 3845–3850, 2013.
- [92] Taeyoung Lee. Robust adaptive attitude tracking on SO(3) with an application to a quadrotor UAV. *IEEE Transactions on Control Systems Technology*, 21:1924–1930, 2013.
- [93] Farhad A. Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric adaptive tracking control of a quadrotor unmanned aerial vehicle on SE(3) for agile maneuvers. *Journal of Dynamic Systems Measurement and Control-Transactions of The Asme*, 137:091007, 2014.
- [94] Yun Yu, Shuo Yang, Mingxi Wang, Cheng Li, and Zexiang Li. High performance full attitude control of a quadrotor on SO(3). *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1698–1703, 2015.
- [95] Jacob Johnson and Randal W. Beard. Globally-attractive logarithmic geometric control of a quadrotor for aggressive trajectory tracking. *IEEE Control Systems Letters*, 6:2216–2221, 2021.
- [96] Taeyoung Lee. Geometric control of quadrotor UAVs transporting a cable-suspended rigid body. *IEEE Transactions on Control Systems Technology*, 26(1):255–264, 2018.
- [97] Sangli Teng, William Clark, Anthony M. Bloch, Ram Vasudevan, and Maani Ghaffari. Lie algebraic cost function design for control on Lie groups. *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 1867–1874, 2022.
- [98] Francesco Bullo and Richard M. Murray. Proportional derivative (PD) control on the Euclidean group. 1995.

- [99] Eugene Lavretsky and Kevin A. Wise. Robust and adaptive control. 2013.
- [100] Manmohan Sharma and Indra Narayan Kar. Adaptive geometric control of quadrotors with dynamic offset between center of gravity and geometric center. *Asian Journal of Control*, 23:1923 – 1935, 2020.
- [101] Davide Invernizzi and Marco Lovera. Geometric tracking control of a quadcopter tiltrotor UAV. *IFAC-PapersOnLine*, 50:11565–11570, 2017.
- [102] Xiao-Ning Shi, Yong-An Zhang, and Di Zhou. Almost-global finite-time trajectory tracking control for quadrotors in the exponential coordinates. *IEEE Transactions on Aerospace and Electronic Systems*, 53:91–100, 2017.
- [103] Yushu Yu and Xilun Ding. A global tracking controller for underactuated aerial vehicles: Design, analysis, and experimental tests on quadrotor. *IEEE/ASME Transactions on Mechatronics*, 21:2499–2511, 2016.
- [104] Jennifer S Yeom, Guanrui Li, and Giuseppe Loianno. Geometric fault-tolerant control of quadrotors in case of rotor failures: An attitude based comparative study. *ArXiv*, abs/2306.13522, 2023.
- [105] Daniel Mellinger and Vijay R. Kumar. Minimum snap trajectory generation and control for quadrotors. *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.
- [106] Petar V. Kokotovic. The joy of feedback: nonlinear and adaptive. *IEEE Control Systems*, 12:7–17, 1992.
- [107] Eugene Lavretsky and Travis E. Gibson. Projection operator in adaptive systems. *ArXiv*, abs/1112.4232, 2011.
- [108] Fabian Riether. Agile quadrotor maneuvering using tensor-decomposition-based globally optimal control and onboard visual-inertial estimation. 2016.
- [109] Girish Joshi and Girish V. Chowdhary. Adaptive control using Gaussian-process with model reference generative network. *2018 IEEE Conference on Decision and Control (CDC)*, pages 237–243, 2018.
- [110] Geesara Kulathunga, Hany Hamed, and Alexandr Klimchik. Residual dynamics learning for trajectory tracking for multi-rotor aerial vehicles. *ArXiv*, abs/2305.15791, 2023.
- [111] Alessandro Saviolo, Guanrui Li, and Giuseppe Loianno. Physics-inspired temporal learning of quadrotor dynamics for accurate model predictive trajectory tracking. *IEEE Robotics and Automation Letters*, 7:10256–10263, 2022.
- [112] Bennett Breese, Manish Kumar, Michael A. Bolender, and David W. Casbeer. Physics-based neural networks for modeling & control of aerial vehicles. *2022 American Control Conference (ACC)*, pages 3218–3223, 2022.

-
- [113] Paul Tylkin, Tsun-Hsuan Wang, Kyle Palko, R. Allen, Ho Chit Siu, Daniel Wrafter, Tim Seyde, Alexander Amini, and Daniela Rus. Interpretable autonomous flight via compact visualizable neural circuit policies. *IEEE Robotics and Automation Letters*, 7:3265–3272, 2022.
- [114] Mantas Lukosevicius and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Comput. Sci. Rev.*, 3:127–149, 2009.
- [115] Claudio Gallicchio and Alessio Micheli. Echo state property of deep reservoir computing networks. *Cognitive Computation*, 9:337–350, 2017.
- [116] Claudio Gallicchio and Alessio Micheli. Architectural richness in deep reservoir computing. *Neural Computing and Applications*, 35:24525 – 24542, 2022.
- [117] Stephen A Conyers, Matthew J Rutherford, and Kimon P Valavanis. An empirical evaluation of ground effect for small-scale rotorcraft. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1244–1250. IEEE, 2018.
- [118] Claudio Gallicchio, Alessio Micheli, and Luca Pedrelli. Deep reservoir computing: A critical experimental analysis. *Neurocomputing*, 268:87–99, 2017.
- [119] Stephen A. Conyers. Empirical evaluation of ground, ceiling, and wall effect for small-scale rotorcraft. 2019.
- [120] P. J. Sánchez-Cuevas, G. Heredia, and A. Ollero. Characterization of the aerodynamic ground effect and its influence in multirotor control. *International Journal of Aerospace Engineering*, 2017:1–17, 2017.
- [121] Elizabeth Bondi, Debadepta Dey, Ashish Kapoor, Jim Piavis, S. Shah, Fei Fang, B. Dilkina, R. Hannaford, Arvind Iyer, L. Joppa, and Milind Tambe. AirSim-W: A simulation environment for wildlife conservation with UAVs. *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, 2018.
- [122] Ratnesh Madaan, N. Gyde, Sai Vemprala, Matthew Brown, K. Nagami, Tim Taubner, E. Cristofalo, D. Scaramuzza, M. Schwager, and Ashish Kapoor. AirSim drone racing lab. In *NeurIPS*, 2019.

Appendix A

Multicopter Ground Effect Plugin

Introduction

Ground effect is a well-known phenomenon for UAVs in both research and practice, which normally refers to an increase lift when UAVs fly close to the ground. For helicopters, Cheeseman-Bennett equation [77] is one of the most acknowledged models to describe such thrust increment due to ground effect. However, this model was proposed based on assumptions only valid for large rotors with uniform blades and variable pitch spinning at a constant speed, which no longer hold for multicopters.

Drawn by the imperative need for a mathematical foundation in controller design, a number of studies have emerged in recent years, focusing on the empirical modeling and evaluation of multicopter ground effect. For example, [119] showed that Cheeseman-Bennett model cannot be applied to today's multicopters and hence constructed a parametric model based on the collected empirical data using a quadrotor test frame; [120] analyzed the partial ground effect of multicopter which appears when some of the rotors experience the ground effect but not all. Besides model-based approaches, learning-based techniques have also been investigated to learn the ground effect model directly from data, such as [26] where a DNN was employed to predict unknown ground effect force and then was used in the controller synthesis to improve the landing precision for quadrotor.

Despite the success of previous studies on multicopter ground effect, all the efforts were exclusively directed towards real UAV platforms, introducing inherent

risks during near-ground flight operations. Additionally, it could be laborious and require extensive efforts for both data collection and performance evaluation. These challenges underscore the pressing need for alternative methodologies that can expedite research and development processes while minimizing associated risks.

As a matter of fact, medium-to-high-fidelity simulators have played a pivotal role in developing advanced algorithms for UAVs in the last decades. Among them, AirSim is one of the most powerful simulators thanks to the visual and physics engine at its core. Compared to other alternatives such as Gazebo [76], AirSim is built on top of physics engine with more advanced render techniques made by UE4, thereby enabling complex and rich environments that are close to the real world. Moreover, it is extensible to accommodate new types of hardware platforms and software protocols, and allows users to create custom APIs in different programming languages including C++ and Python. As a result, a range of research has been carried out on AirSim expansions for UAV applications. For example, AirSim-W [121] presented a simulation environment that was designed specifically for wildlife conservation, while [122] proposed a simulation framework to facilitate fast prototyping of drone racing algorithms. In a similar vein, our work exploits AirSim as the core to build the simulation environment that can emulate quadrotor ground effect.

We introduce AirSim-GE (“GE” stands for Ground Effect), a recently released quadrotor ground effect plugin designed to seamlessly integrate with Microsoft AirSim [67] ecosystem. Two ground effect models, comprising the widely recognized Cheeseman-Bennett model and a parametric model derived from empirical data, were implemented and assessed with high-level APIs provided, opening avenues for future research within the UAV community. AirSim-GE is now available on GitLab at <https://gitlab.com/PoliToComplexSystemLab/AirSim-GE>.

In the following sections, we begin by introducing two ground effect models, namely the Cheeseman-Bennett and parametric models, both of which have been integrated into the AirSim libraries. Then, we provide a detailed explanation of the simulator architecture along with implementation specifics. Lastly, we present an illustrative example of utilizing AirSim-GE to simulate quadrotor ground effect within the simulator.

Mathematical Model of Ground Effect

Cheeseman-Bennett model [77] is one of the most widely accepted mathematical models for describing helicopter ground effect, given as

$$\frac{T_{IGE}}{T_{OGE}} = \frac{1}{1 - (\frac{R}{4Z})^2} \quad \text{iff } \frac{Z}{R} > 0.25, \quad (\text{A.1})$$

where Z denotes the altitude above the ground, R denotes propeller radius, and T_{IGE}, T_{OGE} denote the lift experienced while hovering In-Ground Effect (IGE) and Out-of-Ground Effect (OGE), respectively. Nonetheless, this model is based on assumptions that are only valid for helicopters which use large rotors with uniform blades and variable pitch spinning at a constant speed. Hence, it fails to correctly predict the ground effect for multirotors with fixed-pitch propellers spinning at variable speeds.

To substantiate this claim, experimental findings in [119] showed the discrepancy of Cheeseman-Bennett model with real-world flight data of a quadrotor. Furthermore, a parametric model was derived through MATLAB Curve Fitting Toolbox using empirical data collected from quadrotor test frame, given as

$$\frac{T_{IGE}}{T_{OGE}} = \frac{p_1 x^4 + p_2 x^3 + p_3 x^2 + p_4 x + p_5}{x^3 + q_1 x^2 + q_2 x + q_3}, \quad (\text{A.2})$$

where $x := Z/R$, $p_1, p_2, p_3, p_4, p_5, q_1, q_2, q_3$ are polynomial functions of propeller spacing (i.e., two times of the moment arm) and propeller radius, as given therein. Following this, the aerodynamic force induced by the ground effect along body z -axis can thereby be calculated as $T_{IGE} - T_{OGE}$. Note that (A.2) is computationally efficient since all of the coefficients can be calculated a priori, given the geometry information of the quadrotor. Once the coefficients are known, the model only requires altitude measurement to calculate the thrust ratio through basic arithmetic operations.

AirSim-GE

Simulator Architecture

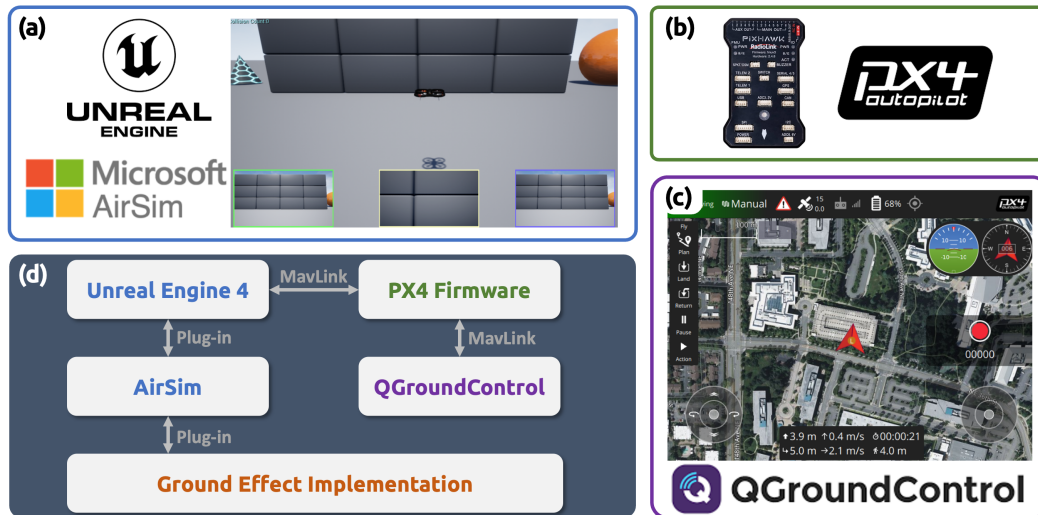


Fig. A.1 **Architecture and components of our custom simulation environment.** (a) Microsoft AirSim and UE4, (b) PX4 firmware, (c) Ground control station, QGroundControl, and (d) Overall architecture of the simulation environment enhanced with our ground effect implementation.

The overall architecture of our custom simulation environment is depicted in Fig. A.1, which consists of three major components (see Fig. A.1(a)-(c)):

(i) Visual and physical simulator

AirSim, an open-source simulator, is designed for drones, cars, and other vehicles. Supporting cross-platform compatibility, it facilitates Software-In-The-Loop (SITL) simulation with baseline flight controllers such as PX4 and ArduPilot, and Hardware-In-The-Loop (HITL) with PX4 for realistic simulation. Aimed at fostering AI research, AirSim serves as a platform, leveraging the UE4 for advanced rendering, for experimenting with deep learning, computer vision, and reinforcement learning algorithms for autonomous vehicles. Furthermore, AirSim provides APIs for data retrieval and vehicle control in a platform-independent manner, enhancing its utility for research purposes.

(ii) Flight controller firmware

The PX4 flight stack provides a range of flight modes and safety features that enable various capabilities, including autonomous navigation, waypoint following, altitude hold, and manual control. Additionally, it offers fail-safe mechanisms such as return-to-home, geofencing, and battery monitoring to enhance safety and reliability during flight operations.

(iii) Ground control station

QGroundControl serves as a comprehensive ground control station software for managing and monitoring UAVs. It facilitates mission planning, waypoint navigation, real-time telemetry monitoring, and firmware updates. Additionally, it provides features for configuring vehicle parameters, managing flight modes, and conducting pre-flight checks to ensure safe and efficient UAV operations.

Implementation Details

Setting up distance sensors

For computing either Cheeseman-Bennett (A.1) or parametric model (A.2), altitude above the ground (Z) is necessary information. To get this data, we used the distance sensor (i.e., a 1-D LiDAR mounted at the bottom of the quadrotor pointing downwards) through AirSim APIs. The distance sensor was specified in the AirSim configuration file `settings.json` located in the directory of `Documents/AirSim`, as shown in Code Listing A.1. Note that the position of the sensor was set to be 0.85m below the center of mass of the multirotor ($Z: 0.85$). This is for sensor calibration from simulation trials, taking the average of subtraction between distance sensor measurement and ground truth when $Z = 0$.

```
1  "DistanceCustom": {
2    "SensorType": 5,
3    "Enabled": true,
4    "MinDistance": 0.1,
5    "MaxDistance": 40,
6    "X": 0, "Y": 0, "Z": 0.85,
7    "Yaw": 0, "Pitch": -90, "Roll": 0,
8    "DrawDebugPoints": true
```

```
9     },
```

Listing A.1 Sensor specification

Calculating thrust difference

In our implementation, it is assumed that the ground effect only affects thrust in the vertical axis of the multicopter, with negligible horizontal variation, which is consistent with the setting used in the previous research [26, 119]. Therefore, the change in thrust resulting from the ground effect is determined by the difference $T_{IGE} - T_{OGE}$, where T_{IGE} and T_{OGE} are computed from (A.1) or (A.2).

Interacting with physics engine

The main part of our ground effect implementation can be found in a source file, namely `MultiRotorGroundEffect.cpp`. To interface with the physics engine in AirSim, we established and initialized the ground effect pointer within `MultiRotorPhysicsBody.hpp` as

```
1  #include "MultiRotorGroundEffect.hpp"
2  MultiRotorGroundEffect* multicopter_ge_;
3  multicopter_ge_ = new MultiRotorGroundEffect;
```

and defined the ground effect function that overrides the virtual one defined in the base class `PhysicsBody.hpp` as

```
1  virtual real_T getGroundEffect() const override
```

Lastly, we integrated our custom function of ground effect `getGroundEffect()` into the `getNextKinematicsNoCollision()` within `FastPhysicsEngine.hpp`. This integration ensures that the total wrench, comprising the body wrench, drag wrench, and ground effect, is computed for each time step. Subsequently, the linear acceleration for the subsequent time step is derived from this aggregated wrench. It is important to note that the outlined procedures provide a basic understanding of the ground effect implementation within the physics engine, acknowledging the existence of additional code modifications beyond the scope of this appendix.

C++/Python APIs

To streamline the utilization of AirSim-GE, we have created high-level APIs, namely `simSetGroundEffect()`, in both C++ and Python. The prefix `sim` denotes that the functionality is exclusively applicable within simulation mode. These APIs enable users to activate the ground effect feature and select a specific model (Cheeseman-Bennett or parametric) tailored to their needs.

The creation of these APIs started with the definition of the ground effect API as a vehicle-based API within `VehicleApiBase.hpp`:

```
1 virtual void simSetGroundEffect(int model) = 0;
```

followed by its implementation in `MavLinkMultirotorApi.hpp`. Then, an RPC handler was added in the server to invoke our implemented method within `RpcLibServerBase.cpp` as

```
1 pimpl_>server.bind("simSetGroundEffect", [&](int model, const std::string& vehicle_name)
  -> void {
2   getVehicleApi(vehicle_name)->simSetGroundEffect(model);
3 });
```

Afterward, C++ and Python client API methods were added to `RpcLibClientBase.cpp` and `client.py` with the following code snippets, respectively:

```
1 void RpcLibClientBase::setGroundEffect(int model, const std::string& vehicle_name)
2 {
3   pimpl_>client.call("simSetGroundEffect", model,
4     vehicle_name);
5 }
```

```
1 def simSetGroundEffect(self, model, vehicle_name = ''):
2     """
3     Set ground effect characterised by model for multirotor corresponding to
4     vehicle_name
5
6     Args:
7     model (int): 0 to no ground effect, 1 to Cheeseman-Bennett model,
8     2 to Parametric model by Stephen Conyers
9     vehicle_name (str, optional): Name of the vehicle to send this command to
10    """
11    self.client.call('simSetGroundEffect', model, vehicle_name)
```

Miscellaneous

The specifications of the quadrotor considered for ground effect modeling within the simulator are detailed in Table 3.1.

Minor adjustments to the model output have also been implemented to address numerical stability issues, specifically setting the output ratio to 1 when $\frac{Z}{R} > 10$ and $\frac{Z}{R} < 0.25$. A comparison between theoretical models and final implementations is presented in Fig. A.2, with dotted lines indicating the collected simulation data.

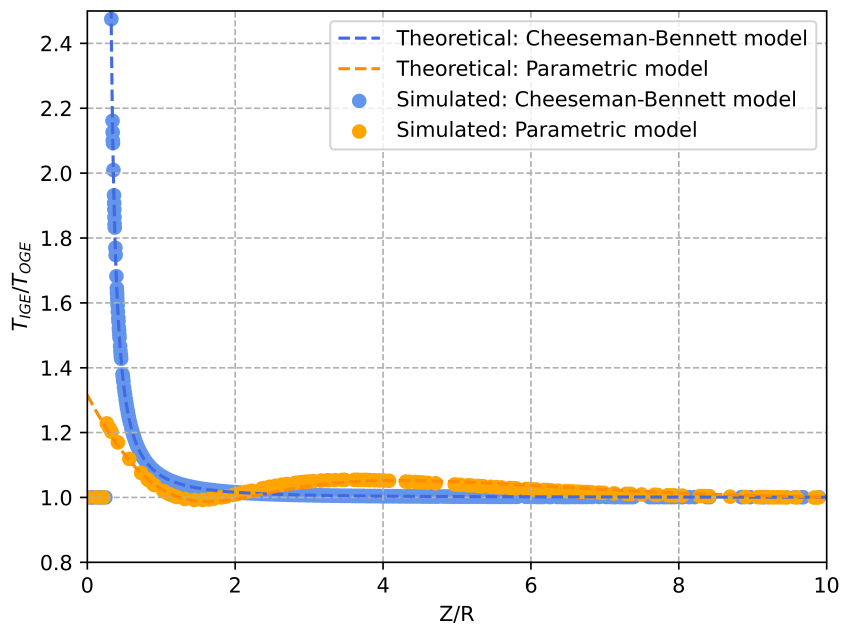


Fig. A.2 Comparison of two ground effect models: Cheeseman-Bennett model vs. parametric model.

Sample Code

The provided code snippet in Code Listing A.2 serves to demonstrate the usage of the AirSim Python API for triggering the ground effect in the simulation environment.

Initially, the code establishes a connection with the AirSim client and enables API control for the multirotor. Subsequently, within a continuous loop, the script waits for user input before switching to different ground effect models: no ground effect (0), the Cheeseman-Bennett model (1), and the parametric model (2). The selected ground effect model will be applied, along with a printed message confirming the type of the model. This example showcases the flexibility of our AirSim API for dynamically adjusting simulation conditions.

```
1 import pprint
2 import setup_path
3 import airsims
4
5 client = airsims.MultirotorClient()
6 client.confirmConnection()
7 client.enableApiControl(True)
8
9 """
10 Model type
11 0: no ground effect; 1: Cheeseman-Bennett model; 2: Parametric model
12 """
13 GND_EFFECT_MODEL = 0
14 while(True):
15     airsims.wait_key('Press any key to change ground effect model')
16     client.simSetGroundEffect(GND_EFFECT_MODEL)
17     print(f"Python API: Current ground effect model is {GND_EFFECT_MODEL}")
18     GND_EFFECT_MODEL += 1
19     if GND_EFFECT_MODEL > 2:
20         GND_EFFECT_MODEL = 0
```

Listing A.2 Sample code in Python

Concluding Remarks

We hope that AirSim-GE will serve as a valuable resource for both the UAV and AI research communities, offering a versatile tool and a practical test bed for research endeavors that necessitate the consideration of ground effect phenomena. It is important to note, however, that Microsoft is poised to launch a new simulation as a successor to the original 2017 AirSim – namely, Microsoft Project AirSim. Designed to meet the escalating demands of the aerospace industry, Project Air-

Sim will offer an end-to-end platform for safe development and testing of aerial autonomy through simulation. This platform will incorporate safety measures, code review capabilities, testing procedures, advanced simulation techniques, and AI functionalities, all uniquely bundled within a commercial product. In light of these developments, we endeavor to integrate our ground effect plugin into Project AirSim upon its release and warmly invite constructive feedback to refine our contributions in the interim.