Data-Driven Aeroelastic Modeling of Flexible Blended-Wing-Body Aircraft

by

Aghil Zeinalzadeh
M.Sc., Sharif University of Technology, 2014

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Mechanical Engineering Department

We acknowledge and respect the Lək ʷəŋən (Songhees and Xʷsepsəm/Esquimalt) Peoples on whose territory the university stands, and the Lək ʷəŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

Data-Driven Aeroelastic Modeling of Flexible Blended-Wing-Body Aircraft

by

Aghil Zeinalzadeh
M.Sc., Sharif University of Technology, 2014

Supervisory Committee

Dr. Afzal Suleman. Co-Supervisor,
(Department of Mechanical Engineering)

Dr. Mario Bras. Co-Supervisor,
(Department of Mechanical Engineering)

Dr. Issa Traoré. Outside Member,
(Department of Electrical and Computer Engineering)

ABSTRACT

As modern aircraft configurations like the BWB feature high-aspect-ratio wings prone to aeroelastic instabilities, there is an increasing need for real-time, constraint-aware flutter suppression methods. Traditional empirical and analytical techniques often struggle to capture time-varying dynamics near flutter speed. This thesis presents a data-driven framework for predicting the nonlinear aeroelastic response of flexible Blended Wing Body (BWB) aircraft near flutter condition. This technique is positioned within the context of active control frameworks such as Model Predictive Control (MPC), which require accurate, low-dimensional models for real-time application.

The study begins with a comprehensive review of aeroelasticity, flutter suppression strategies, and emerging data-driven methods, including Dynamic Mode Decomposition with control (DMDc) and Long Short-Term Memory (LSTM) networks. To support model development, low-fidelity aeroelastic simulations are performed using SHARPy, a nonlinear simulation platform for data generation and flutter analysis. A cantilevered BWB wing is excited with gust profiles near flutter, and transient responses are recorded. This simulation data is then used to build both DMDc and LSTM models. The LSTM model is designed for sequence forecasting, enabling it to capture long-term dependencies in the time series, while DMDc provides a linear approximation of system dynamics influenced by control inputs.

The SHARPy solver setup is validated using a Pazy wing benchmark, confirming its ability to reproduce unsteady behaviors such as Limit Cycle Oscillations (LCO) and transient gust responses. These results establish SHARPy as a reliable platform for data generation and flutter analysis. Building on this, the final chapter develops and evaluates the predictive models of the flexible wing based on the ORCA configuration. Flutter speed is estimated through modal analysis, followed by dynamic simulations using PRBS and sine-sweep velocity inputs.

Results show that LSTM significantly outperforms DMDc near flutter, achieving less than 2% prediction error within the training velocity range. Although accuracy decreases during extrapolation, LSTM maintains a computational advantage making it highly suitable for real-time use. The chapter concludes by conceptually integrating LSTM into an MPC framework, highlighting its potential for flutter suppression and flight envelope enhancement. Overall, this thesis establishes a robust framework for real-time aeroelastic control using machine learning–based models, enabling safer and more efficient operation of flexible aircraft structures.

Contents

Supervisory Committee **ii**

Abstract **iii**

Table of Contents **iv**

List of Tables **vi**

List of Figures **vii**

Acknowledgements **xii**

Dedication **xiii**

1 Introduction **1**

1.1 Motivation 2
1.2 Background 2
1.3 Objectives 5
1.4 Contributions 7

2 Data-Driven Modeling Methodology **8**

2.1 Aerostructural Modeling Using SHARPy 9
2.2 DMDc Design 11
2.3 LSTM Neural Network Design 13
2.3.1 LSTM Internal Operations 14
2.3.2 LSTM Design Considerations 19

3 Aerostructural Modeling of A Flexible Rectangular Wing Using SHARPy **24**

3.1 Introduction 25
3.1.1 SHARPy Package Features 26
3.2 Steps in SHARPy Modal Analysis 28
3.3 Steps in SHARPy Transient Analysis 29
3.3.1 Problem Definition 29

3.3.2 Aerodynamic Model Setup . . . 29
3.3.3 Structural Model Setup . . . 30
3.3.4 Aerodynamic Solver . . . 31
3.3.5 Structural Solver . . . 31
3.3.6 Aerostructural Coupling . . . 31
3.3.7 Time Integration Solver . . . 31
3.3.8 Simulation . . . 32
3.3.9 Post-Processing and Analysis . . . 32
3.4 Pazy Wing Benchmark Model . . . 33
3.5 Technical Challenges and Special Features . . . 33
3.6 Results . . . 36
3.7 Summary . . . 43

4 Data-Driven Aeroelastic Modeling of the BWB Aircraft Wing 45
4.1 Modal Analysis of the Wing . . . 46
4.2 Data-Driven Aerostructural Modeling . . . 48
4.3 Conclusion . . . 65

5 Conclusions 67
5.1 Key Findings . . . 68
5.2 Future Work . . . 69

Bibliography 70

A Additional Information 76
A.1 Step-by-step DMDc Implementation with Python . . . 76
A.2 Pseudo code for updating aerodynamic properties . . . 77
A.3 Pseudo code for solver setup and post-processing . . . 79
A.4 Python code for transient simulation solver setup . . . 80

List of Tables

Table 2.2 SHARPy Solver Configurations. 10
Table 2.3 LSTM Internal Operations Aligned with Diagram Components. 16
Table 2.4 LSTM Model Setup. 18
Table 2.5 Overlapping Window Approach for Sequence-to-Sequence LSTM Training ($x\_seq\_length = 5$, $y\_seq\_length = 3$). 21
Table 2.6 Non-Overlapping Window Approach for Sequence-to-Sequence LSTM Training ($x\_seq\_length = 5$, $y\_seq\_length = 3$). 21

Table 3.1 Pros and Cons of SHARPy 28
Table 3.2 Properties of the Pazy Wing 33

Table 4.1 Elastic modal analysis results for the first four modes 46

List of Figures

Figure 2.1 BWB wing FEM model in SHARPy. 9
Figure 2.2 DMDc Architecture. 13
Figure 2.3 LSTM Architecture. 14
Figure 2.4 Block diagram of the data-driven aero elastic modeling. 17
Figure 2.5 Evolution of LSTM model loss versus epoch over train dataset #1. 18
Figure 2.6 Structure of data processing in the trained LSTM model. 19
Figure 2.7 Representation of original and normalized state space for an arbitrary hysteresis loop. 20

Figure 3.1 SHARPy applications 27
Figure 3.2 Pazy wing aerostructual model in SHARPy 30
Figure 3.3 Demonstration of the frame of reference in SHARPy for an arbitrary airplane 31
Figure 3.4 Dynamic response issue: No excitation due to stationary gust input, resolved using *relative_motion = True*. 34
Figure 3.5 Onset of flutter only beyond flutter speed due to zero initial AoA, requiring high-speed gust input. 35
Figure 3.6 Need for steady-state convergence before applying gust excitation in transient simulations. 35
Figure 3.7 Experimental results of the Pazy wing measured by FBG and MRS 37
Figure 3.8 Wingtip displacement of the Pazy wing at different speeds 37
Figure 3.9 Comparison of wing tip displacement calculated by SHARPy and measured in the experimental article 38
Figure 3.10 Aeroelastic modal analysis of the cantilevered wing 38
Figure 3.11 Dynamic response of the wing for Test A gust excitation 40
Figure 3.12 Dynamic response of the wing for Test B gust excitation 40
Figure 3.13 Dynamic response of the wing for Test C gust excitation 41
Figure 3.14 Dynamic response of the wing for Test D gust excitation 41
Figure 3.15 Non-linear phenomenon captured in the transient simulation of the cantilevered wing 42
Figure 3.16 Demonstration of the deformed geometry and wake 43

Figure 4.1 First four elastic mode shapes of the BWB wing 46
Figure 4.2 Aeroelastic modal analysis for the BWB wing with lumped mass at the wing tip 48
Figure 4.3 Effects of change of the horizon length on the prediction performance of LSTM 49
Figure 4.4 Evolution of wing transverse displacement over time at the tip, midspan, and root sections, as simulated by SHARPy and predicted by DMDC and LSTM (h=1) 50
Figure 4.5 Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the train dataset . 51
Figure 4.6 Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the test dataset #1 52
Figure 4.7 Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the test dataset #2 53
Figure 4.8 Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the test dataset #3 54
Figure 4.9 Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the train dataset 54
Figure 4.10 Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #1 55
Figure 4.11 Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #2 55
Figure 4.12 Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #3 56
Figure 4.13 Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the train dataset 57
Figure 4.14 Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #1 57

Figure 4.15 Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #2 58
Figure 4.16 Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #3 58
Figure 4.17 Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the train dataset 59
Figure 4.18 Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the test dataset #1 60
Figure 4.19 Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the test dataset #2 61
Figure 4.20 Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the test dataset #3 62
Figure 4.21 Comparison of the gamma distribution of aerodynamic panels for test dataset #2 LSTM Prediction vs. SHARPy 63
Figure 4.22 LSTM prediction of wingtip vertical displacement at different times within the horizon compared with SHARPy for the test dataset #1 64
Figure 4.23 LSTM prediction of wingtip vertical displacement at different times within the horizon compared with SHARPy for the test dataset #2 64
Figure 4.24 LSTM prediction of wingtip vertical displacement at different times within the horizon compared with SHARPy for the test dataset #3 65

List of Abbreviations

Abbreviation	Full Term
ROM	Reduced-Order Model
LSTM	Long Short-Term Memory (neural network)
DMD	Dynamic Mode Decomposition
DMDc	Dynamic Mode Decomposition with Control
MPC	Model Predictive Control
CFD	Computational Fluid Dynamics
FEA	Finite Element Analysis
FBG	Fiber Bragg Grating
MRS	Motion Reference Sensor
LCO	Limit Cycle Oscillation
BWB	Blended Wing Body
SINDy	Sparse Identification of Nonlinear Dynamics
POD	Proper Orthogonal Decomposition
AE	Aeroelastic
PRBS	Pseudo-Random Binary Sequence
DoF	Degrees of Freedom
SHARPy	Simulation of High-Aspect Ratio Planforms in Python
ORCA	Open Research Configuration Aircraft
PID	Proportional–Integral–Derivative (controller)

List of symbols

Symbol	Description	Units
t	Time	s
$\mathbf{x}$	State vector	–
$\mathbf{X}$	State matrix	–
$\mathbf{X}'$	Future state matrix	–
$\mathbf{u}$	Input velocity at free stream direction	m/s
$\mathbf{w}$	Input velocity perpendicual to the free-stream direction	m/s
$\mathbf{y}$	Output vector	–
$\mathbf{A}$	State matrix (linear system dynamics)	–
$\mathbf{B}$	Input matrix (control influence)	–
$\mathbf{C}$	Output matrix	–
$\mathbf{D}$	Feedthrough matrix	–
$\hat{\mathbf{x}}$	Predicted state vector	–
$\boldsymbol{\theta}$	Neural network weights or parameters	–
N	Number of time steps or samples	–
h	Time delay or prediction horizon	s
ω	Angular frequency	rad/s
U_∞	Freestream velocity	m/s
α	Angle of attack	deg or rad
z	Vertical displacement (e.g., wingtip)	m
y	Spanwise location	m
M	Mass matrix	kg
K	Stiffness matrix	N/m
C	Damping matrix	N·s/m
ϕ	Mode shape	–
λ	Eigenvalue or growth rate	s^{-1} or rad/s
$\mathcal{L}$	Loss function (used in LSTM training)	–
$\mathbb{R}$	Set of real numbers	–
$\mathbf{m}$	Total number of snapshots in a dataset	–

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to:

Supervisors: for your mentorship, encouragement, patience, and continuous support.

NSERC and Bombardier: for generously funding my studies through a scholarship.

Simin, the rock of my family: for supporting me during difficult moments.

> *"I have to believe in a world outside my own mind. I have to believe that my actions still have meaning, even if I can't remember them. I have to believe that when my eyes are closed, the world's still there."*
>
> — Leonard Shelby, *Memento*

DEDICATION

Just hoping this is useful!

Chapter 1

Introduction

This chapter presents a comprehensive review of the advancements in modeling and control of aeroelastic systems, with a focus on data-driven and reduced-order modeling techniques for flutter prediction and suppression. It begins with the historical development of aeroelasticity and active flutter suppression, emphasizing the need for integrating control systems into structural design. Blended Wing Body aircraft are introduced as a modern, efficient configuration, though their high-aspect-ratio wings increase susceptibility to aeroelastic instabilities. Traditional analytical and empirical approaches, while foundational, face limitations in capturing complex, nonlinear, time-varying dynamics near flutter conditions. Emerging techniques such as Dynamic Mode Decomposition with control, Sparse Identification of Nonlinear Dynamics, and deep learning models like Long Short-Term Memory networks have shown promise in overcoming these challenges. The integration of LSTM-based reduced-order models with Model Predictive Control frameworks is discussed as a novel approach to enable real-time, constraint-aware control of flexible aircraft. Finally, the SHARPy simulation platform is introduced as the computational tool used in this study to generate high-fidelity aerostructural data for model training and evaluation.

1.1 Motivation

The increasing demand for efficient, lightweight, and aerodynamically advanced aircraft designs—such as Blended Wing Body (BWB) configurations—has introduced new challenges in understanding and managing aeroelastic instabilities, particularly flutter. Traditional physics-based models often fall short in accurately predicting the nonlinear and time-varying behavior of flexible aircraft near critical dynamic instabilities. This limitation is especially prominent in modern flight regimes involving large structural deformations, strong fluid-structure coupling, and the need for real-time control. With the growing availability of high-fidelity simulation tools like SHARPy and advances in machine learning, data-driven modeling techniques—such as Long Short-Term Memory (LSTM) neural networks—offer a powerful alternative. These methods enable accurate modeling of complex aeroelastic phenomena with reduced computational cost and are well-suited for integration into real-time control frameworks such as Model Predictive Control (MPC). This research is motivated by the need to develop such robust, predictive, and computationally efficient models that can enhance the stability, safety, and performance of next-generation flexible aircraft systems operating near flutter boundaries.

1.2 Background

Aeroelasticity is the branch of physics and engineering that examines the interactions between inertial, elastic, and aerodynamic forces when an elastic body is subjected to fluid flow. The study of aeroelasticity can be broadly classified into two fields: static aeroelasticity, which focuses on the static or steady-state response of an elastic body to fluid flow, and dynamic aeroelasticity, which investigates the dynamic (usually vibrational) response of the body [1]. The establishment of aeroelasticity was in the mid-20th century, particularly during and after World War II, when the interaction between aerodynamics and the structural dynamics of aircraft became a focus of engineers and researchers. Some early seminar papers were published by von Kármán and Ludwig Prandtl, laying the groundwork for understanding aeroelasticity in aerospace engineering.

The advanced technology known as active flutter suppression (AFS) offers an effective means to eliminate flutter instabilities, resulting in significant weight savings and a more adaptable airframe. The 1970s marked the initial validation of AFS for aircraft [2]. Another earliest work was conducted by Shinji Suzuki's 1993 [3]. He presented a comprehensive approach to the optimization of structure considering a cantilever wing with an aileron surface, actively controlled to alleviate gust loading. The methodology leveraged a job control program that coordinates various analysis programs (structural, aerodynamic, control system analysis) and manages data interchange among

them. Suzuki developed a mathematical model for the integrated system and outlined a computer program system based on sequential goal programming (GP) capable of handling of multiple objectives and a large number of design variables efficiently. This approach addresses the complexity of design requirements, which include maintaining sufficient stability margins across different flight conditions and ensuring structural integrity both with and without active control systems. This method not only enhances the aerodynamic performance and structural efficiency of aircraft but also ensures greater adaptability and safety in operational environments.

In parallel with these developments, Blended Wing Body (BWB) aircraft have gained significant attention owing to their advantages such as high aerodynamic efficiency, extended endurance capabilities, and exceptional stealth performance [4–6]. To cater for the demands of long-endurance missions, high-aspect-ratio aerodynamic configurations making use of lightweight composite materials are preferred due to their enhanced lift-to-drag performance but lead to increased wing flexibility. However, flexible aircraft structures may exhibit instabilities arising from the coupling of aerodynamic forces, structural elasticity, and inertial forces leading to flutter, which poses a severe risk of structural failure [7].

To effectively implement control strategies, the dynamics of the flexible aircraft must be accurately captured. Despite progress made by traditional mathematical methods [2, 3, 8–10], developing computational models to facilitate efficient model and data transfer between structural design and control systems has been a crucial step to enhance multidisciplinary aerospace design [11].

To bridge the gap between structural design and control systems, Dynamic Mode Decomposition (DMD) has demonstrated its ability to extract dominant system dynamics and facilitate control-focused modelling [12]. Unlike Proper Orthogonal Decomposition (POD), which focuses solely on spatial correlation and energy content, DMD offers a modal decomposition where each mode exhibits the same linear behavior over time. Thus, DMD not only reduces dimensionality but also provides a temporal model for mode evolution. However, DMD identification may not be suitable when interventions are ongoing, as they perturb the dynamical system. DMD with control (DMDc) addresses this challenge by incorporating the effects of interventions into the modeling process [13].

DMDc can be used to derive surrogate models from computational fluid dynamics simulations [14]. Instead of constructing models for the entire flow field or concentrating solely on overall performance metrics, the emphasis is placed on directly forecasting the pressure distribution along the body surface. For instance, N. Fonzi et al. utilized DMDc to create reduced-order aeroelastic models for handling the nonlinearities of morphing wings suitable for various operating condi-

tions and model predictive control [15] [16]. The developed reduced order model (ROM) was a super compact DMDc which introduced two modifications tailored for flexible aeroelastic systems in the design and control of morphing wings. Firstly, a method to address algebraic equations was proposed, and secondly, an interpolation technique to link multiple linear DMDc models was developed under different operating conditions. This methodology shows advantages over existing methods by not only reducing computational demands but also improving model adaptability and accuracy in real-time control applications. This technique is useful for identifying reduced-order models from data where control or forcing is present.

Ripepi et al. explored the implementation of parametric aerodynamic ROM using proper orthogonal decomposition (POD) and Isomap techniques within the scope of the German Aerospace Center's (DLR) Digital-X project [17]. This research is pivotal in advancing the use of high-fidelity computational fluid dynamics (CFD) models in aerodynamic simulations, by significantly reducing computational time and resource allocation without compromising on the accuracy essential for practical applications. The ROM was adept at predicting surface pressure distributions with considerably lower evaluation times. A remarkable innovation in their approach is the interpolation techniques that approximate solutions within these reduced dimensional spaces, thereby maintaining a balance between computational efficiency and the fidelity of physical phenomena represented.

Handling large deformation angles is typically challenging for traditional analytical models. Sparse identification of nonlinear dynamics (SINDy) combined with optimization routines like LASSO and STLSQ to identify the nonlinear dynamical systems from computational fluid dynamics (CFD) simulation data maintains high accuracy in the modeling of complex aeroelastic phenomena due to large deformation [18]. They significantly enhanced the process of capturing complex fluid-structure interactions to predict the aerodynamic loads on structures undergoing large amplitude and frequency oscillations, such as heaving and pitching motions of a flat plate. The study has particular focus on stability and numerical bifurcation analysis. The models were shown to be effective in handling large deformation angles, which are typically challenging for traditional analytical models. IN summary, the SINDy-based framework enables a hybrid approach by incorporating known physical laws into modern data-driven modeling, allowing it to learn simple, physics-informed models from data and uncover the underlying governing equations [19]. While this approach improves the selection of relevant terms and can effectively handle noise, outliers, and parametric dependencies, it relies on a predefined library of candidate functions (e.g., polynomials, trigonometric functions). If the appropriate basis functions are not included in this library, the resulting model may fail to accurately capture the underlying physics.

Some data-driven methods like feed-forward (FNN) and recurrent neural networks (RNN) have shown good potential to resolve the issues associated with SINDy-based models. Natarajan, Kapania, and Inman [20] explored the application of a time-delay recurrent neural network to analyze the aeroelastic behavior of aircraft with time-variant structural properties. While neural networks have been increasingly utilized to study dynamical systems, especially for predicting and understanding complex systems with unknown governing equations [21–24], their ability to accurately predict the dynamics of flexible aircraft structures in the presence of unstable aeroelastic behavior has not been quantified in the open literature. For example, in the vicinity of flutter, the system can become highly sensitive to external disturbances such as angle of attack. By accurately modeling the dynamics in this region, more effective control systems can be designed to maintain stability and performance under these challenging conditions.

Recent methods have successfully employed RNN architectures, particularly Long-Short Term Memory (LSTM) networks, for sequential data tasks. LSTM networks can effectively learn the temporal patterns of normal sequential data by being trained exclusively on non-anomalous examples. During training, the network captures the typical behavior and structure of the data. When new data is introduced, the LSTM attempts to predict or reconstruct it based on what it has learned. If the new data follows the normal pattern, the prediction error remains low. However, if the data deviates from the learned patterns, the error increases significantly, allowing such deviations to be identified as outliers [25, 26]. Moreover, LSTM-based ROMs have been used for predicting transonic buffet on airfoils and capturing time-delayed effects associated with unsteady aerodynamics [27, 28]. Here, the authors trained the model on a dataset generated using CFD simulations on a NACA 64A010 airfoil, where aerodynamic responses were obtained under different flight conditions. However, the limitations of the method included residual errors (up to 12% for moment coefficients in some cases), the need for extensive training data, and the computational cost associated with training deep learning models.

1.3 Objectives

Given the effectiveness of LSTMs, the main goal of the current research is to develop a data-driven framework using LSTM neural networks to provide an efficient representation of the aerostructural dynamics of a flexible BWB aircraft wing near the onset of flutter. To do so, an aerostructural model is developed using SHARPy, or Simulation of High Aspect Ratio Planforms. SHARPy is an aeroelastic analysis package developed by Palacios et al. in the Department of Aeronautics, Imperial College London [29]. It is used for modal, static, and transient simulation of structural,

aerodynamic, aeroelastic, and flight dynamics analysis of flexible aircraft, flying wings, and wind turbines [30, 31].

Extensive experimental data on the aeroelastic behavior of the Pazy wing are available, obtained from studies conducted in renowned wind tunnels [32]. These experimental results have been recently compared with aeroelastic analysis of the Pazy wing modeled by different non-linear aeroelastic solvers including SHARPy [33]. The results showed good performance of SHARPy in terms of nonlinear aeroelastic simulation due to large deformation. This accurate and fast simulation will be able to capture some non-linearity in terms of large deformation because of different flight conditions.

Dealing with transient analysis in SHARPy can lead to large deformation angles when aircraft is far beyond the flutter phenomenon. Therefore, predicting flutter speed ensures that aircraft can operate within safe limits during transient simulation in term of numerical divergence in SHARPy. Various methods exist for predicting flutter speed. Some literature employs empirical methods to simplify the system's nonlinear dynamics [34]. Other research utilizes ROMs with low or high-fidelity solvers for aerodynamics and structural simulations [35] [36]. SHARPy provides modal analysis through linearized aeroelastic stability analysis, and use a frequency domain approach in order to perform eigen value analysis.

The model considered in this work is a flexible high-aspect ratio wing based on the BWB design of the Bombardier EcoJet Research Project. After constructing the time-evolving state of the aircraft—including the structural deformations of the wing and the distribution of aerodynamic forces and moments across the wingspan—the LSTM neural network is trained. As a result, a comprehensive database is developed including the aircraft aerostructural characteristics using SHARPy considering control inputs.

The performance of the proposed LSTM-based data-driven framework is compared with DMDc. The comparison shows the effectiveness of the framework in capturing the complex aerostructural dynamics near the flutter speed. We show that LSTM networks are well-suited for modeling aeroelastic behavior near flutter due to their ability to learn from sequential data without relying on linearity assumptions. This makes them more effective in capturing the complex, nonlinear dynamics leading up to flutter. In contrast, DMDc is a linear, data-driven method that approximates system dynamics using linear operators, which limits its ability to accurately represent such nonlinear behaviors near the flutter boundary.

While the work discussed in [37, 38] focuses on the application of MPC in aerospace systems and nonlinear modeling such as aeroelasticity, LSTMs offer a powerful data-driven approach that can generalize aeroelastic behavior across different flight conditions, making them a promising candidate for integration with MPC frameworks. The controller predicts the future behavior of the system over a defined horizon, h, and optimizes the sequence of control actions to achieve desired performance, all while satisfying system constraints such as actuator limits. Since the LSTM model uses a sequence of past observations to predict multiple future steps (horizon), the controller will use these predictions in a rolling optimization framework. In this approach, MPC makes decisions based on both real-time optimization and long-term memory-based prediction, making it suitable for complex dynamical systems with nonlinearities.

1.4 Contributions

This thesis offers several novel contributions to the field of aeroelasticity and flight control:

- Application of a validated, nonlinear low-fidelity simulation framework using SHARPy for flexible aircraft wings under gust excitation.
- A comparative benchmark of DMDc and LSTM for flutter prediction across both linear and nonlinear regimes.
- A demonstration of LSTM's effectiveness in reproducing complex aeroelastic responses, establishing it as a robust candidate for real-time control applications.
- A structured workflow for using simulation data to train and evaluate time-series prediction models in the context of aeroelasticity.

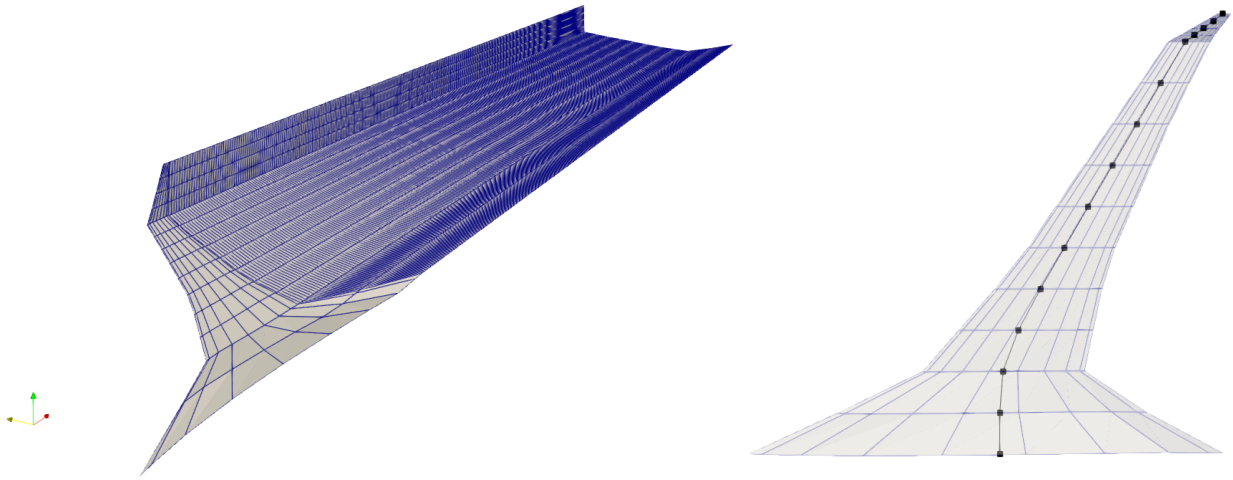
Chapter 2

Data-Driven Modeling Methodology

This chapter outlines the methodology used to develop and evaluate data-driven models for predicting the aeroelastic response of a flexible Blended Wing Body aircraft. The study integrates high-fidelity transient simulations from SHARPy, a low-fidelity aeroelastic solver, with two predictive modeling techniques: Dynamic Mode Decomposition with control and Long Short-Term Memory neural networks. First, a cantilevered wing configuration is simulated in SHARPy using its nonlinear structural and aerodynamic solvers, capturing the unsteady aerostructural behavior under gust-induced excitations near flutter conditions. Key solver configurations, time integration schemes, and modeling limitations are addressed in detail. The extracted simulation data is then used to train the DMDc model by identifying system dynamics and control influence from state and input snapshots. In parallel, an LSTM architecture is trained using sequence-to-sequence forecasting to learn long-term dependencies in the time-evolving aeroelastic data. The design of the LSTM model is explained thoroughly. Finally, the chapter discusses how both models are evaluated and prepared for future integration with Model Predictive Control frameworks to enable real-time control of nonlinear aeroelastic systems.

2.1 Aerostructural Modeling Using SHARPy

Figure 2.1 shows the aeroelastic model of the BWB wing in SHARPy including 15 structural nodes corresponding to 7 three-nodded beam elements. The number of chord panels in chordwise direction is 9 considering 1-cos distribution for the panels in the same direction. An important challenge during the setup of transient simulations was the need to run the analysis for a long time and let it reach a steady-state condition before applying any external excitation, which significantly increased computational time.



(a) Isometric View. **(b)** Top View.

Figure 2.1: BWB wing FEM model in SHARPy.

Configuring solvers to handle coupled aeroelastic behaviors with large non-linear deformations demands extensive parameter testing to ensure accuracy and stability to detect non-linear phenomenon. We choose StepUVLM in SHARPy which uses Vortex Lattice Method [39] as a suitable flow solver for low-fidelity results. Note that UVLM model does not account for stall, flow separation, or nonlinear aerodynamic effects. However, SHARPy can account for nonlinearity in terms of structural deformation. In this regard, NonLinearDynamic structural solver provides an interface to the structural library *xbeam* in SHARPy and updates the structural parameters for every time step of the simulation. This solver is called as part of a standalone structural simulation. The DynamicCoupled solver couples the chosen aerodynamic and structural solvers to advance the aeroelastic system's solution over time. To use the DynamicCoupled solver, an instance of the StaticCoupled solver must be included in the SHARPy solution flow when defining the problem case. Finally, NewmarkBeta is the time integration method used to solve the equations of motion for dynamic simulations, particularly in aeroelastic problems. In summary, Table 2.2 shows the important information in SHARPy to provide a transient analysis.

In term of post processing SHARPy can deliver aerostructural information in text files.

Table 2.2: SHARPy Solver Configurations.

Solver Type	**Solver**	**Key Settings**
Static	BeamLoader	unsteady = on
	AerogridLoader	wake_shape_generator = StraightWake
	NonLinearStatic	gravity_on = True
	StaticUvlm	newmark_damp = 0.2
	StaticCoupled	velocity_generator = SteadyVelocityField
		structural_solver = NonLinearStatic
		aero_solver = StaticUvlm
Dynamic	StepUvlm	velocity_generator = GustVelocityField
	NonLinearPrescribedStep	gravity_on = True
	DynamicCoupled	newmark_damp = 0.2
	BeamLoader	structural_solver = NonLinearStatic
	StaticCoupled	aero_solver = StepUvlm
		unsteady = on
		relaxation_factor = 0.5

The state space representation of a dynamical system requires capturing the system response to specific input signals such as air speed. However, SHARPy is limited to providing transient simulations as speed changes versus time. To address this, we will activate gust profiles in different directions resulting in changing effective angle of attack. This can be as inputs to the dynamical system to study the effects of varying velocity on wing stability, although it adds more complexity in terms of unsteady aerodynamics due to change in effective angle of attack. Another challenge is that in transient simulation we need to spend perhaps long time to reach steady state condition before exciting the system with gust profile. This could remarkably affects on the computational time. Setting an appropriate relaxation factor, bending, and torsional stiffness is the key role to control the stability and convergence of iterative processes. Other limitations of SHARPy, such as restrictions on different types of boundary conditions, do not affect our methodology since we use a fixed wing with a simple boundary condition.

2.2 DMDc Design

Modeling complex, high-dimensional systems that involve dynamic behavior and require control is becoming increasingly relevant across not only traditional engineering and physical sciences but also modern fields like disease control, distribution networks, and power grids. DMDc leverages both system measurements and external control inputs to uncover system behavior and input-output relationships without relying on explicit equations. Building on the strengths of standard DMD, DMDc works directly with snapshot data, manages high-dimensional datasets efficiently, and links data-driven analysis to nonlinear system dynamics through Koopman operator theory. Additionally, DMDc can identify input-output patterns, enabling the development of ROMs for forecasting and control design in complex systems.

DMDc was initially created to analyze data from complex systems influenced by external forces during observation. A key example that motivated its development is the effort to eliminate infectious diseases like polio. While cases of paralysis are recorded, mass vaccination campaigns are simultaneously carried out to control the outbreak. DMDc integrates both the system's state data and external inputs to distinguish the system's natural dynamics from the effects of interventions. Understanding these dynamics and the input-output relationships is essential for building predictive models and designing effective control strategies. This section introduces the fundamental formulation of the DMDc method.
The goal of DMDc is to characterize the relationship between three quantities: the current state $\mathbf{x}_k$, the future state $\mathbf{x}_{k+1}$, and the control input $\mathbf{u}_k$. This relationship is described by the canonical discrete linear dynamical system:

$$\mathbf{x}_{k+1} \approx A\mathbf{x}_k + B\mathbf{u}_k, \tag{2.2.1}$$

where:

- $\mathbf{x}_k \in \mathbb{R}^n$ is the current state,
- $\mathbf{u}_k \in \mathbb{R}^q$ is the control input,
- $A \in \mathbb{R}^{n\times n}$ governs the unforced system dynamics,
- $B \in \mathbb{R}^{n\times q}$ captures how the input affects the system.

As in standard DMD, the above relationship is approximate rather than exact. The goal is to estimate the matrices A and B from available measurement data. Let the state snapshot matrices be defined

as:

$$X = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_{m-1} \\ | & | & & | \end{bmatrix}, \quad X' = \begin{bmatrix} | & | & & | \\ \mathbf{x}_2 & \mathbf{x}_3 & \cdots & \mathbf{x}_m \\ | & | & & | \end{bmatrix}.$$

Additionally, we define the input snapshot matrix as:

$$\mathbf{U} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_{m-1} \\ | & | & & | \end{bmatrix}. \tag{2.2.2}$$

The system relationship in matrix form becomes:

$$\mathbf{X}' \approx A\mathbf{X} + B\mathbf{U}. \tag{2.2.3}$$

Operators A and B capture the intrinsic system dynamics and the effect of external control inputs. In control theory, it is common to augment the system dynamics with a measurement equation to model how the internal states are observed. This is typically represented by a linear observation model:

$$\mathbf{y} = C\mathbf{x}, \tag{2.2.4}$$

where:

- $\mathbf{x}$ is the (possibly hidden) state vector,
- $\mathbf{y}$ is the measurement or observed output,
- C is the observation or measurement matrix.

In many DMD applications, particularly those involving numerical simulations, the matrix C is often the identity matrix or is derived directly from simulated data, implying full-state measurement. Figure 2.2 illustrates the collection of state and control input data from numerical simulation in SHARPy. DMDc relies on three snapshot matrices $\mathbf{X}$, $\mathbf{X}'$, and $\mathbf{U}$. $\mathbf{X}$ contains snapshots of aerostructural information such as life coefficient of airfoils across the wing span (denoted as $\mathbf{x}_1$ in the Figure), structural displacements, $\mathbf{x}_2$, in different directions for each structural node of the wing. $\mathbf{X}'$ represents the matrix of future state snapshots—that is, the system state one time step ahead of the corresponding columns in $\mathbf{X}$. After collecting the required data from SHARPy, we start finding operators A and B using the *full-rank* DMDc algorithm as shown in Appendix A.1.

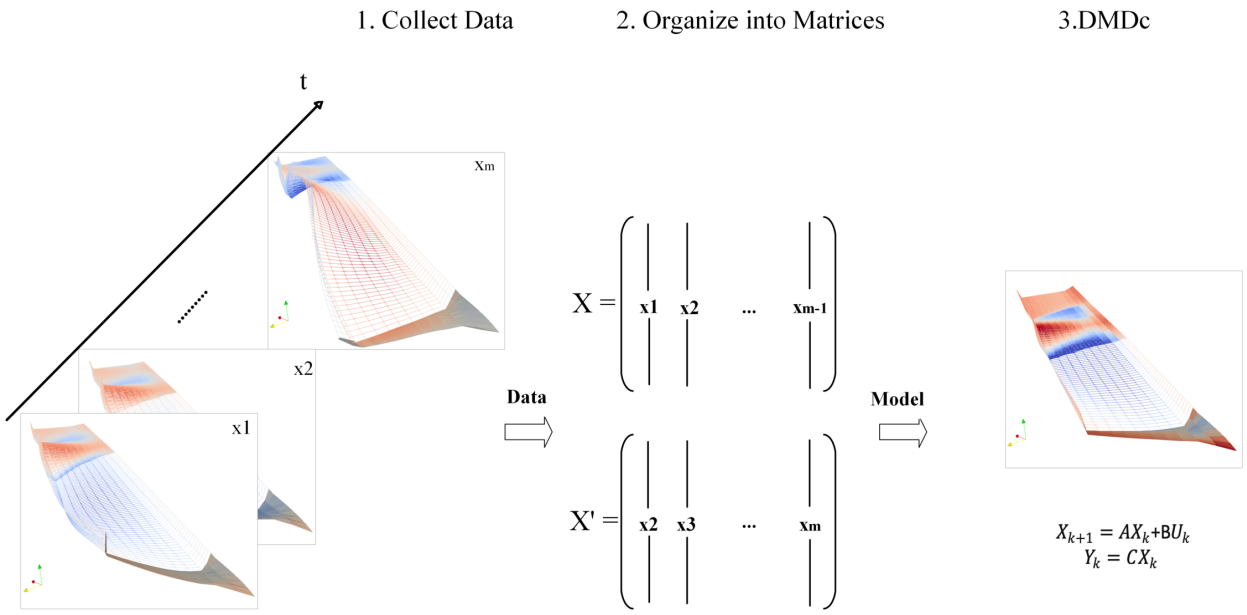


Figure 2.2: DMDc Architecture.

2.3 LSTM Neural Network Design

LSTM network is an advanced recurrent neural network that provides a well-structured architecture by incorporating gates within its basic unit, known as a cell, shown in Figure 2.3. These gates enable the model to capture both long-term and short-term dependencies across time steps while mitigating the issues of gradient explosion and/or vanishing, which commonly occur in standard RNNs in prediction of long-term dependent non-linear aeroelastic phenomena [33]. Indeed, LSTM networks are more robust to the vanishing gradient problem. The gates in LSTMs help regulate the flow of gradients, preventing them from becoming too small during back propagation. This allows LSTMs to learn long-term dependencies more effectively than standard RNNs. Lastly, LSTM networks are versatile and can be used in various applications. They have been successfully applied in fields such as natural language processing, time series analysis, and anomaly detection, demonstrating their broad applicability and effectiveness. The gates are referred to as the forget, input and output gates. This architecture is useful for MPC where control decisions depend on past state and inputs of the dynamical system over extended time horizons.

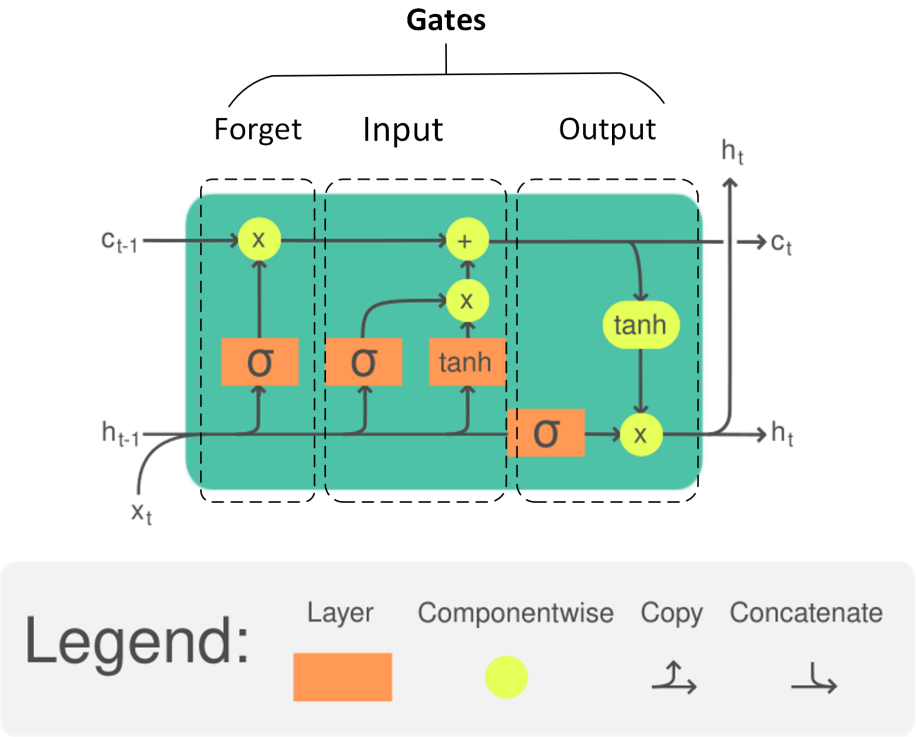


Figure 2.3: LSTM Architecture.

Table 2.3 provides a detailed breakdown of the internal operations within a LSTM cell, aligned with its standard architectural diagram. Each row outlines a specific computational step—from gate activations to memory updates—along with the corresponding mathematical formulation and its functional role in regulating the flow of information. The table also indicates where each operation typically appears in the standard LSTM schematic, aiding visual association. This structured overview highlights how the LSTM architecture selectively filters and integrates past and current information to produce reliable predictions in sequential learning tasks, such as modeling nonlinear aeroelastic behavior.

2.3.1 LSTM Internal Operations

The Long Short-Term Memory (LSTM) architecture consists of several interacting gates and operations that regulate the flow of information across time steps in a sequence. Each component plays a unique role in retaining relevant temporal features, suppressing irrelevant information, and enabling long-term memory. Table 2.3 summarizes the key operations and their mathematical formulations, aligned with their visual location in the schematic diagram shown in Figure 2.3.

At each time step t, the LSTM receives two inputs: the current external input vector $x_t \in \mathbb{R}^{n_x}$ and the hidden state from the previous time step $h_{t-1} \in \mathbb{R}^{n_h}$. The first gate computed is the *forget gate*, which determines which parts of the previous cell state c_{t-1} should be retained. It is defined as:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{2.3.1}$$

where W_f and b_f are the weight matrix and bias for the forget gate, and $\sigma(\cdot)$ is the sigmoid activation function. The operator $[\cdot, \cdot]$ denotes concatenation. The elementwise product between f_t and c_{t-1}, given by:

$$f_t \odot c_{t-1}, \tag{2.3.2}$$

represents the retained memory from the past and forms one part of the updated cell state. Next, the LSTM computes the *input gate*, which controls how much of the new information should be added to the memory. This gate is defined as:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{2.3.3}$$

and the candidate memory content is computed as:

$$\tilde{c}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c), \tag{2.3.4}$$

where W_i, b_i, W_c, and b_c are the corresponding weights and biases. The new candidate memory is filtered by the input gate activation through:

$$i_t \odot \tilde{c}_t, \tag{2.3.5}$$

which ensures that only selected parts of the new information are allowed into the cell state. The cell state is then updated by combining the retained previous memory and the filtered new memory:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{2.3.6}$$

where c_t is the updated long-term memory vector. The final step involves computing the *output gate*, which determines what part of the updated memory should be output to the next layer or time step:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{2.3.7}$$

where W_o and b_o are the weights and bias for the output gate. Finally, the hidden state h_t, also known as the short-term memory or output of the LSTM, is computed as:

$$h_t = o_t \odot \tanh(c_t), \tag{2.3.8}$$

which represents the externally visible output of the cell and is passed forward in both the temporal and layer dimensions.

Table 2.3: LSTM Internal Operations Aligned with Diagram Components.

Operation	**Formula and Description**	**Diagram Location in Figure 2.3 (if applicable)**
Forget Gate	Equation (2.3.1): Controls retention of the previous cell state c_{t-1}.	Left orange sigmoid block
Forget Path	Equation (2.3.2): Elementwise multiply with old memory content.	Yellow multiplication ($\times$) near top left
Input Gate	Equation (2.3.3): Selects which parts of the input should affect the cell state.	Middle orange sigmoid block
Candidate Cell State	Equation (2.3.4): Proposed new memory content.	Tanh block adjacent to input gate
Input Path	Equation (2.3.5): Filters candidate memory before updating cell state.	Yellow multiplication block next to tanh
Cell State Update	Equation (2.3.6): Combines old and new information into updated memory.	Central yellow addition node
Output Gate	Equation (2.3.7): Decides which part of the memory to expose as output.	Rightmost orange sigmoid block
Final Output	Equation (2.3.8): Outputs short-term memory based on filtered cell state.	Right-side yellow multiplication and tanh block

Figure 2.4 shows the block diagram of the data-driven approach used to predict the aerostructural behavior of the wing over time. To achieve this, the aeroelastic model of the BWB is first prepared in SHARPy based on predefined aeroelastic design parameters, represented by letters A and B in the figure. The modeled dynamical system is excited using control inputs, including gust profile velocities in the directions. A Pseudo-Random Binary Signal (PRBS) is selected to excite the system over time in such a way that, at certain moments, it exceeds the flutter speed for short periods, as indicated by letter C in the figure to build the training dataset(D). All aerostructural information, including structural deformation, aerodynamic forces, and the circulation strength of the vortex panels in the UVLM for each node, is considered as the state vector of the system. These outputs, concatenated with the inputs evolving over time, form the training dataset for LSTM and DMDc (E). Note that all inputs are required to be normalized before training the LSTM to ensure that all features are within the same scale, preventing any one feature from dominating the model. In this study, no dimensionality reduction was applied, and a full-rank DMDc (F) approach was used for

training. The detailed procedure for training DMDc is outlined in [12].

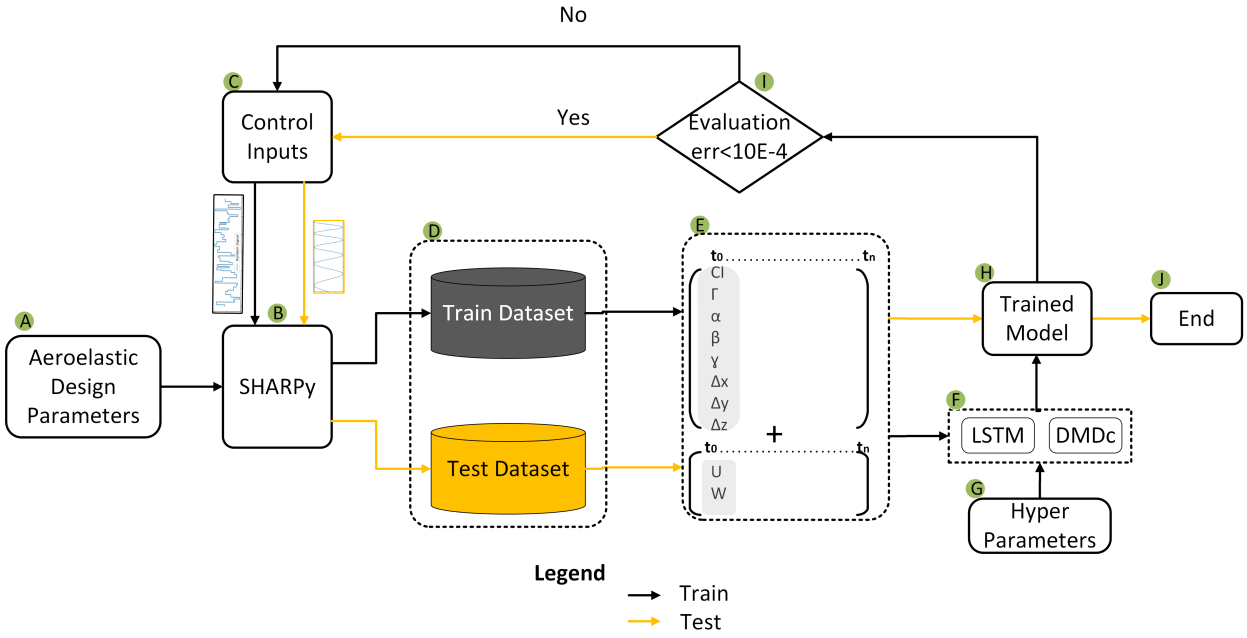


Figure 2.4: Block diagram of the data-driven aero elastic modeling.

Having trained the LSTM neural network and created the DMDc model using the training dataset(H), their prediction performance is evaluated on the same dataset(I). Figure 2.5 shows the evolution of loss versus epoch during the training phase of the LSTM model. If they show good agreement with the state-space evolution in time simulated by SHARPy, the algorithm proceeds to the next step to generate another dataset for testing DMDc and LSTM models. It is noted that the test dataset is not seen during the training of LSTM and DMDc. Finally, the trained model is implemented to predict the state of the dynamical system versus time.

Table 2.4: LSTM Model Setup.

Component	Parameter	Value
Model	Input Dimension	425
	Output Dimension	423
	Hidden Dimension	64
	Number of Layers	2
Training	Batch Size	32
	Loss Function	MSELoss()
	Optimizer	Adam
	Horizon	50
	Sequence length	200

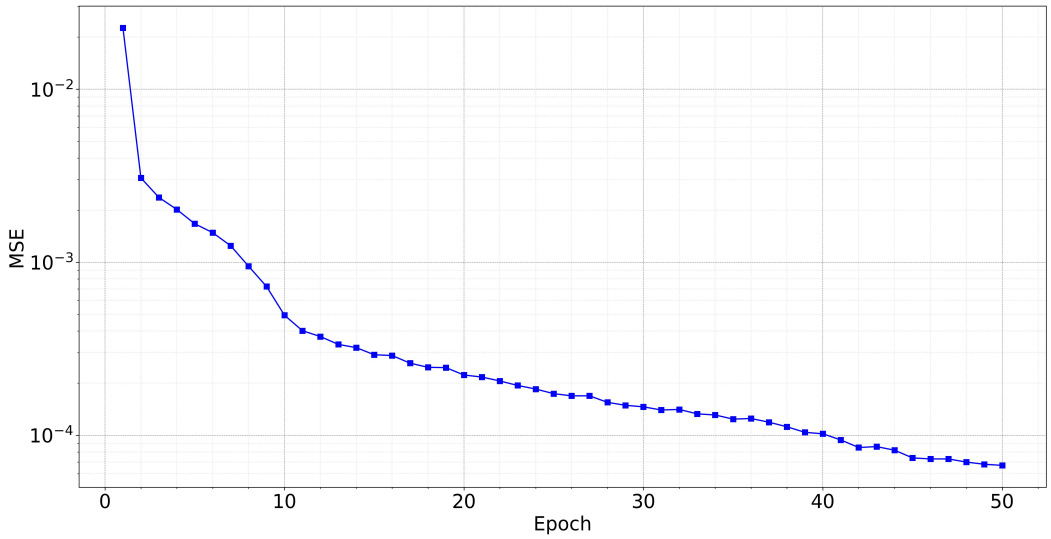


Figure 2.5: Evolution of LSTM model loss versus epoch over train dataset #1.

Figure 2.6 illustrates the workflow of data processing within the trained LSTM model. To make predictions, the LSTM takes a sequence of past data as input ($X_m \; to \; X_{m-s}$) shown in the Figure, which includes the scaled aeroelastic state-space variables of the system, such as displacements, velocities, or angles. This time-ordered sequence allows the LSTM to learn temporal dependencies in the system dynamics. Based on this learned pattern, the model outputs predictions of the system's future state-space over a specified prediction horizon ($X_{m+1} \; to \; X_{m+h}$).

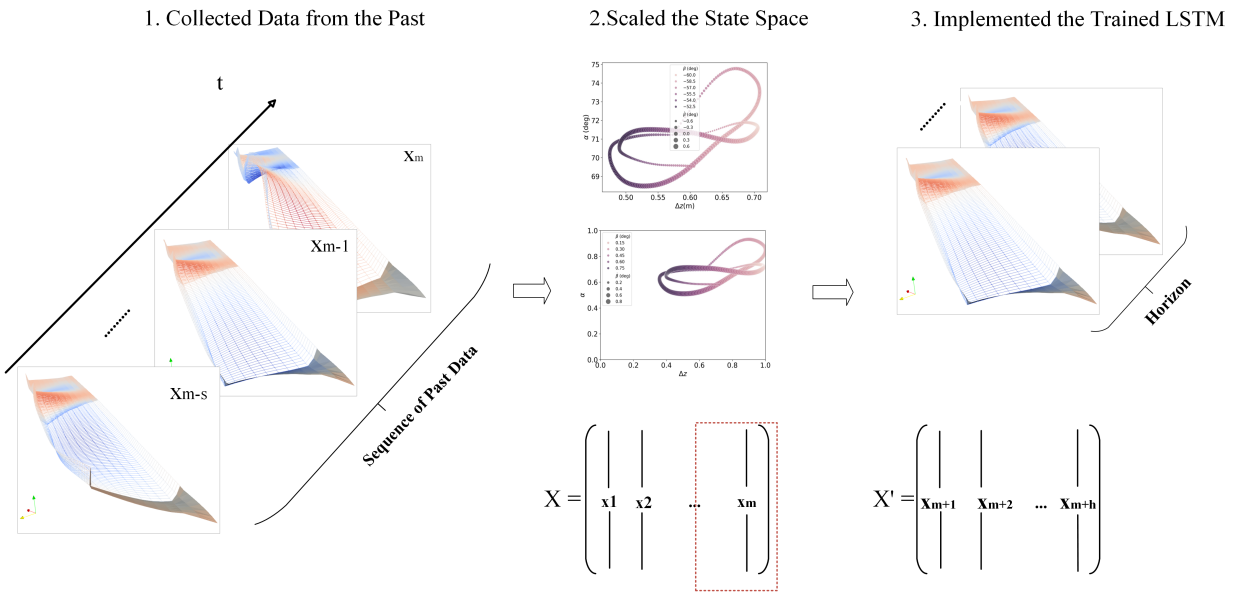


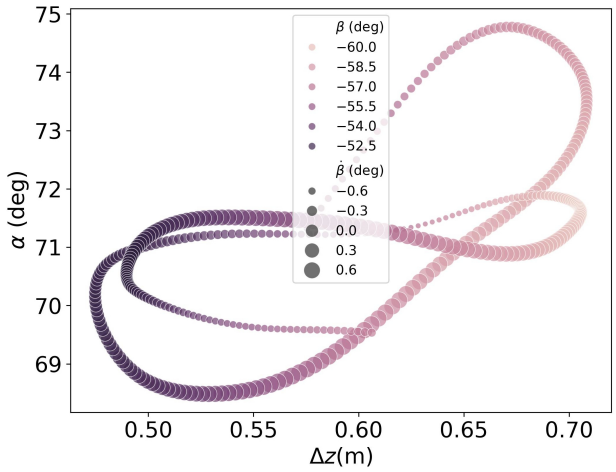
Figure 2.6: Structure of data processing in the trained LSTM model.

2.3.2 LSTM Design Considerations

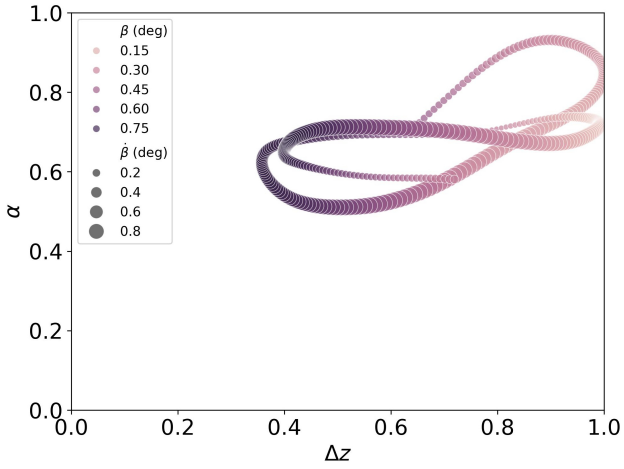
The position of instability in the dataset can matter depending on how LSTM is trained and the application we are focusing on. LSTMs are designed to capture sequential dependencies. Instabilities occurring at the beginning of the dataset can have a lasting effect throughout the sequence due to the memory mechanism. Instabilities occurring at the end of the dataset may not influence earlier parts of the sequence but will strongly affect predictions near the instability period. If the instability appears early, the LSTM might prioritize learning features related to instability and potentially ignore normal dynamics. This can bias the model to overfit the unstable behavior. If the instability is concentrated near the end, the model might focus on the transition to instability and learn to predict when instability is likely to occur, which can be only useful for forecasting such behavior. Therefore, to achieve an accurately trained LSTM model, the wing must exhibit instability at random times during the transient simulation period.

LSTMs are designed to learn temporal dependencies and dynamics. If the dynamics of the system (states corresponding to velocity) are consistent across the velocity range, the LSTM might extrapolate well even outside the exact training range.In this regard step functions are sharp transitions, which might be more challenging to model than smooth sine waves. If the LSTM successfully learned from step inputs, it is likely to handle smooth sine sweeps with ease. This is the reason why we chose PRBS to train the data-driven models.

Testing LSTM with unseen dataset requires not to scale the test data separately. It is essential to use the same scaler (fit on the training data) to scale both the training and testing datasets. Indeed, the model is trained on data scaled using the training set statistics (e.g., min, max for *MinMaxScaler* or mean, std for *StandardScaler*). If the test data are scaled independently, the test data will be transformed differently, causing a mismatch in the model's expected input distribution. Moreover, from a real-world application perspective, when a trained model is deployed, future test data is not accessible during training. Using a different scaler for the test data introduces a scaling mismatch, resulting in inconsistent predictions and incorrect evaluation. Figure 2.7 shows the original and scaled data for important structural variables in the state space for only a selected hysteresis loop among transient data points .



(a) Original



(b) Scaled

Figure 2.7: Representation of original and normalized state space for an arbitrary hysteresis loop.

When using an LSTM with a horizon, the model generates predictions for future time steps that were not directly part of the input sequence. However, there are key details to understand about what this means. There are two common approaches when iterating through data for multi-step (horizon-based) forecasting:

- Sliding Window Approach (Overlapping Windows): In this method, as shown in Table 2.5 the next input sequence shifts forward by one step, maintaining overlap between consecutive sequences. This approach, which is selected for this project, provides many overlapping training samples, helping the model learn dependencies better. It also ensures the model learns transitions between sequences.

Table 2.5: Overlapping Window Approach for Sequence-to-Sequence LSTM Training ($x\_seq\_length = 5$, $y\_seq\_length = 3$).

Iteration	**Input Sequence** (x)	**Prediction Sequence** (y)
1st	$[t_0, t_1, t_2, t_3, t_4]$	$[t_5, t_6, t_7]$
2nd	$[t_1, t_2, t_3, t_4, t_5]$	$[t_6, t_7, t_8]$
3rd	$[t_2, t_3, t_4, t_5, t_6]$	$[t_7, t_8, t_9]$

- Non-Overlapping Windows: The next sequence starts right after the previous prediction window. This approach, presented in Table 2.6 is faster training since it skips redundant overlapping sequences. It is more efficient if future dependencies are long-range. Note that in terms of data-driven aeroelastic modeling, both short and long term dependencies are needed to be considered. Therefore, this approach is not selected in LSTM design.

Table 2.6: Non-Overlapping Window Approach for Sequence-to-Sequence LSTM Training ($x\_seq\_length = 5$, $y\_seq\_length = 3$).

Iteration	**Input Sequence** (x)	**Prediction Sequence** (y)
1st	$[t_0, t_1, t_2, t_3, t_4]$	$[t_5, t_6, t_7]$
2nd	$[t_8, t_9, t_{10}, t_{11}, t_{12}]$	$[t_{13}, t_{14}, t_{15}]$
3rd	$[t_{16}, t_{17}, t_{18}, t_{19}, t_{20}]$	$[t_{21}, t_{22}, t_{23}]$

The accuracy of multi-step forecasting depends on the relationship between *seq_length* (length of past input sequence) and *horizon* (number of future time steps predicted). The trade-off is between model memory (context) and error propagation. If the model gets longer past context but predicts fewer future steps, there will be more reliable predictions because the model has more past information to learn patterns. Moreover, this model will be a short-term predictions only, meaning the

model can't forecast far into the future.

A longer seq_length generally enhances the LSTM's predictive accuracy by incorporating more historical data, which is particularly beneficial for systems with long-term dependencies. This extended context allows the model to capture underlying trends and patterns that influence future states. However, an excessively long sequence can introduce noise or redundant information, potentially leading to overfitting and increased computational complexity.

In the context of MPC, selecting the appropriate seq_length is crucial, as it directly impacts prediction accuracy and, consequently, control performance. Since MPC relies on accurate future state predictions to optimize control actions over its prediction horizon, a poorly chosen seq_length can degrade performance. If the system dynamics exhibit short-term dependencies, increasing seq_length may not provide significant benefits and could unnecessarily burden computation. Conversely, for systems with long memory effects, a short seq_length may lead to inaccurate state predictions, impairing the controller's ability to make effective decisions. Moreover, stability and robustness in MPC are closely tied to prediction reliability. A suboptimal seq_length can cause instability in control actions, leading to overly aggressive or ineffective adjustments. Therefore, balancing seq_length is essential to ensure accurate forecasting while maintaining computational efficiency and control stability.

The prediction horizon in MPC is a crucial parameter that must be carefully chosen based on the system dynamics, control objectives, and computational limitations. While it may seem beneficial to train an LSTM model to predict as far ahead as possible, this approach involves trade-offs. The horizon length N in MPC is typically determined by several factors. System dynamics play a key role—if the system exhibits slow changes, a longer horizon is necessary to capture meaningful effects of control actions, whereas fast-changing systems require shorter horizons to remain responsive. Additionally, the prediction horizon should be long enough to model the system's response to control inputs effectively. However, computational feasibility is another critical factor. Extending the prediction horizon increases the complexity of both LSTM inference and MPC optimization, potentially making real-time control impractical. Therefore, selecting an optimal horizon length involves balancing prediction accuracy, computational efficiency, and the system's dynamic behavior.

Data-driven aeroelastic modelling plays a crucial role in predicting the state space for an LSTM-based MPC controller over a specified horizon. The trained model is well-suited for MPC, as it can effectively learn complex temporal dependencies from data, making them ideal for modelling aerostructural systems where the state evolution depends on past states and inputs. Indeed, when

the BWB encounters flutter, the control unit equipped with MPC utilizes an LSTM to forecast the system's state and subsequently optimizes the control inputs (here, airspeed) for future time steps to minimize a certain cost function, ensuring that the BWB can effectively manage flutter.

Chapter 3

Aerostructural Modeling of A Flexible Rectangular Wing Using SHARPy

This chapter presents the development and validation of a nonlinear, low-fidelity aeroelastic model for a flexible cantilevered wing near flutter conditions using the SHARPy simulation framework. The aim is to investigate unsteady aerostructural dynamics, capture nonlinear phenomena such as Limit Cycle Oscillations, and evaluate the system's transient response under gust excitation. The chapter begins with an overview of SHARPy's capabilities for simulating static and dynamic aeroelastic behaviors and highlights the major steps involved in setting up transient simulations, including aerodynamic and structural modeling, solver configuration, and time integration. A benchmark case based on the Pazy wing is modeled and validated against experimental wind tunnel data. The study addresses several technical challenges, including solver divergence near flutter speed, the impact of geometric nonlinearity, and the accurate implementation of time-varying gust profiles. Results show good agreement between simulation and experiment, and demonstrate SHARPy's ability to capture key nonlinear aeroelastic responses. The outcomes support the use of SHARPy as a robust tool for exploring flexible wing behavior and provide a foundation for future integration with data-driven control systems.

3.1 Introduction

It is expected that highly flexible aeroelastic structures will become ubiquitous in future transportation and energy systems, enabled by advanced materials and emerging morphing wing technologies [40]. Flexible wings are highly pron to failure due to the linear or nonlinear flutter phenomenon. Typically involving wings, tails, and control surfaces, flutter manifests as an unstable, self-excited vibration where the structure absorbs energy from the air stream, often leading to catastrophic structural failure.

The main goal of this chapter of my thesis is to build a nonlinear low-fidelity aero-elastic transient modeling to provide a fast and accurate representation of aerostructural dynamics of a cantilevered aircraft wing near flutter speed using SHARPy. Furthermore, this project also aims to address key challenges in performing non-linear transient aerostructural simulations to see nonlinear phenomenon such as Limit Cycle Oscillation (LCO) due to large deformation in SHARPy. Note that configuring solvers to handle coupled aeroelastic behaviors with large nonlinear deformations demands extensive parameter testing to ensure accuracy and stability in this project. Modeling aeroelastic systems requires precise setup of geometries, solution configurations, and solver tuning in SHARPy's Python environment to detect nonlinear phenomena.

The state space representation of the cantilevered wing requires capturing the system response to specific input air velocity signals. However, SHARPy is limited to providing transient simulations at a fixed speed. One of the main challenges in this research is answering this question how to vary the velocity profile over time. To address this, I will activate gust profiles in different directions as input to the dynamical system to study the effects of varying velocity on wing stability. This adds more complexity in terms of unsteady aerodynamics. I anticipate that with certain gust profiles reaching flutter speed, wing instability can be observed.

The first step of this chapter is modeling the wing configuration in SHARPy including structural and aerodynamic modelling setup using the pre-designed information of the wing. These steps are done through configuring the FEM and Aerodynamic modules in Python. Next, a modal analysis using SHARPy will be conducted to determine the flutter speed for deflected wing, highlighting the need for a reduced-order model for efficient frequency-domain analysis. Note that, a comprehensive database of the aircraft's aerostructural characteristics also will be developed through transient simulations near flutter speed. Finally in the post processing phase, we will demonstrate the time-evolving state of the aircraft, including wing deformation and aerodynamic force and moment distribution along the wingspan, both before and after reaching flutter speed. Note that, in order to

visualize the results, *Paraview* is used to read all aerostructural results extracted from SHARPy.

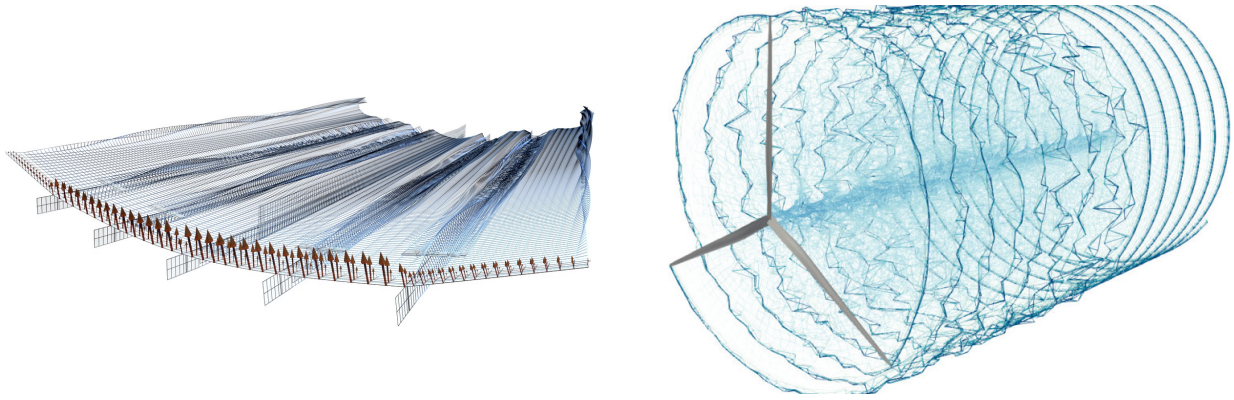
3.1.1 SHARPy Package Features

SHARPy is a modular and flexible architecture which is designed to simulate complex aeroelastic phenomena, particularly for highly flexible structures such as modern aircraft, flying wings, and wind turbines. The package provides various analysis tools to study the interplay between structural mechanics, aerodynamics, and flight dynamics including:

- **Static Aerodynamic, Structural, and Aeroelastic Analysis:** SHARPy supports static analyses that consider fuselage effects, providing steady-state insights into the aerodynamic and structural behavior of aeroelastic systems.
- **Trim Condition Calculation:** SHARPy can compute trim conditions, ensuring that the system remains in stable and balanced flight for given aeroelastic configurations.
- **Nonlinear, Dynamic Time-Domain Simulations:** SHARPy offers nonlinear, dynamic simulations over time, adaptable to various conditions such as:
  - **Prescribed Trajectories:** Forcing the structure to follow a predefined path.
  - **Free Flight Dynamics:** Simulating the natural response of the system under no external constraints.
  - **Dynamic Follower Forces:** Analyzing forces that move with the structure, such as thrust or other applied loads.
  - **Control Inputs:** Including thrust modulation and control surface deflections, which allow for comprehensive control system analysis.
  - **Time-Domain Gusts:** Simulating both spanwise constant and non-constant gusts, as well as full 3D turbulent fields, enabling low-fidelity turbulence modeling.
- **Multibody Dynamics:** SHARPy handles complex multibody configurations with movable parts. It supports:
  - **Wind Turbines:** Modeling blade articulation and flexibility, shown in Figure **??**.
  - **Hinged Aircraft:** Systems with movable components such as folding wings.
  - **Catapult-Assisted Takeoffs:** Modeling the dynamics of aircraft launches.
- **Linear Analysis:** SHARPy includes linearization techniques for analyzing system behavior around nonlinear equilibrium points, offering:

  - **Frequency Response Analysis:** Studying how the system responds at different frequencies.
  - **Asymptotic Stability Analysis:** Assessing long-term system stability.

- **Model Order Reduction:** SHARPy incorporates model order reduction techniques to enhance computational efficiency:
  - **Krylov-Subspace Reduction Methods:** Reducing the problem size while maintaining essential dynamics.
  - **Balancing Reduction Methods:** Optimizing the model to preserve control energy and observability.



(a) Flying wings **(b)** Wind turbines

Figure 3.1: SHARPy applications [29]

Note that SHARPy's modular design enables the simulation of complex aeroelastic cases, especially for very flexible aircraft. The structural solver supports complex beam arrangements with geometrical nonlinearities. The UVLM solver is capable of modeling deformed lifting surfaces excluding aerodynamic stall model. It incorporates various wake modelings, making SHARPy suitable for low-fidelity aeroelastic simulations. Table 3.1 shows the advantages and disadvantages of using SHARPy for aeroelastic simulation.

Table 3.1: Pros and Cons of SHARPy

Pros	Cons
Open-Source	Limited High-Fidelity Aerodynamics
Modular Framework	Potential flow solvers
Easy to integrate new solvers	Python-based implementation can be slower
Nonlinear Capabilities	Documentation Gaps
Multiphysics Integration	Boundary Condition Limitations
Steady and Transient Analyses	Limited handling of complex geometries
	No Built-In Structural Optimization

3.2 Steps in SHARPy Modal Analysis

SHARPy includes many modules for various analyses, but to compute flutter in highly flexible, clamped wings, only specific ones are used. The process involves solving the nonlinear static aeroelastic equilibrium, then separately linearizing the structural and aerodynamic systems, reducing them using modal and Krylov-based methods [41], and finally analyzing the stability of the resulting reduced system [42].

The structural model in SHARPy uses a geometrically exact 1D 3-nodded beam formulation. The nonlinear equations are derived using Hamilton's principle and take the general form:

$$\mathcal{M}(\boldsymbol{\eta})\ddot{\boldsymbol{\eta}} + \boldsymbol{Q}_{\text{str}}(\boldsymbol{\eta}, \dot{\boldsymbol{\eta}}) + \boldsymbol{Q}_{\text{int}}(\boldsymbol{\eta}) = \boldsymbol{Q}_{\text{ext}}(\boldsymbol{x}, \boldsymbol{\eta}) \tag{3.2.1}$$

Here, $\mathcal{M}$ is the mass matrix, and $\boldsymbol{Q}$ represents the discrete external (aerodynamic), gyroscopic, and stiffness forces. The flexible degrees of freedom are defined in a body-attached reference frame $\mathcal{F}_b \in \mathbb{R}^{6(N_{\text{node}}-1)}$, which includes local displacements and rotations. The rotations are parameterized using a Cartesian rotation vector (CRV).

To analyze the system dynamics, the nonlinear equations are linearized about a steady-state reference defined by fixed values $(\boldsymbol{\eta}_0, \dot{\boldsymbol{\eta}}_0)$, assuming no acceleration, i.e., $\ddot{\boldsymbol{\eta}}_0 = 0$. Small deviations from this equilibrium, denoted by δ, lead to the following linearized equation [43]:

$$\mathcal{M}(\boldsymbol{\eta}_0)\ddot{\boldsymbol{\eta}} + \mathbf{C}(\boldsymbol{\eta}_0, \dot{\boldsymbol{\eta}}_0)\dot{\boldsymbol{\eta}} + \mathbf{K}(\boldsymbol{\eta}_0, \dot{\boldsymbol{\eta}}_0)\boldsymbol{\eta} = \mathbf{G}_{\text{ext}}(\delta\boldsymbol{x}, \delta\boldsymbol{\eta}) \tag{3.2.2}$$

In this expression, **C** and **K** represent the linearized damping and stiffness matrices. The system is then reduced by projecting it onto modal coordinates associated with the equilibrium configuration and retaining only the dominant dynamic modes.

3.3 Steps in SHARPy Transient Analysis

Transient simulation in the aeroelastic context refers to the time-dependent analysis of the dynamic response of flexible structures, such as aircraft wings, under varying aerodynamic and structural loads. Unlike steady-state simulations, which consider a system's behavior at equilibrium, transient simulations capture the evolution of the system over time, making them essential for studying phenomena like flutter, gust interactions, and other time-varying aerodynamic effects. These simulations are critical for understanding how an aircraft's structural components interact with aerodynamic forces during flight, particularly when subjected to rapid changes in conditions such as airspeed, angle of attack, or external disturbances like gusts. The major steps in conducting a transient aerostructural simulation typically include steps described below.

3.3.1 Problem Definition

The first step in performing an aerostructural simulation is to model the geometry of the structure (e.g., aircraft wing or body) and generate a computational mesh suitable for both aerodynamic and structural analysis. Next, it is essential to specify the structural properties, such as Young's modulus, density, and Poisson's ratio, for the materials used in the simulation. This is followed by defining boundary conditions for both the aerodynamic and structural components, such as fixed supports or external forces. These steps are crucial for ensuring accurate simulation results.

3.3.2 Aerodynamic Model Setup

Aerodynamic modeling requires selecting the flow solver, providing geometry mesh, and boundary condition. For modeling transient simulations (e.g., gust or turbulence), it is necessary to set up the time-dependent flow . In this project different gust profiles are examined to study the aero structural behavior of the wing. Finally, mesh movement enables mesh deformation allowing the fluid domain to follow the structure's deformation. In SHARPy, defining aerodynamic panels, representing the lifting surface for the geometry, is required for its unsteady solver, *StepUVLM*. Moreover, UVLM requires defining aerodynamic panels for the wake of the wing.

Figure 3.2 illustrates the cantilevered wing modeled in SHARPy. The image depicts the aerodynamic panel divisions in the chordwise and spanwise directions. Additionally, to account for

wake roll-up effects, relatively long panels are used to model the wake. It is worth noting that these divisions can be adjusted during a mesh refinement study; however, since smaller grid sizes result in higher computational costs, the simulation continues with the same setup as the Pazy wing described in the studies conducted by AePW3 [33].

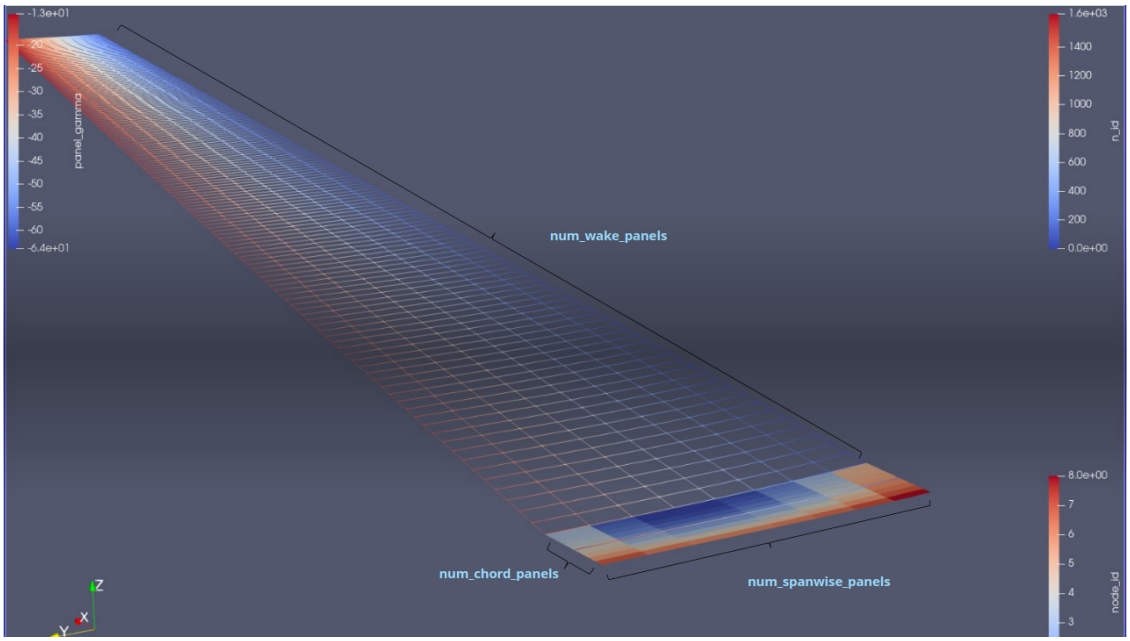


Figure 3.2: Pazy wing aerostructual model in SHARPy

3.3.3 Structural Model Setup

First, it is necessary to introduce SHARPy Frame of Reference(FoR) in configuring the structural model. FoR , depicted in Figure 3.3 is an important point, because all the inputs that move with the beam are in material FoR. For example: follower forces, stiffness, mass, lumped masses. The material frame of reference is noted as B. Essentially, the x component is tangent to the beam in the increasing node ordering, z looks up generally and y is oriented such that the FoR is right handed. SHARPy's beam model is based on Finite Element Method (FEM) for discretization and geometrically nonlinear formulations (three-nodded beam theory). Moreover, it couples its structural model with UVLM to capture aerodynamic loads.

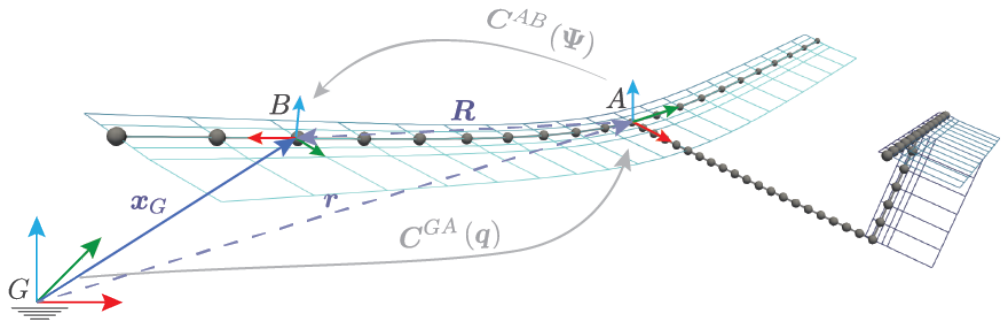


Figure 3.3: Demonstration of the frame of reference in SHARPy for an arbitrary airplane [30]

3.3.4 Aerodynamic Solver

StepUVLM provides an aerodynamic only simulation in time by time stepping the solution. The type of aerodynamic solver is parsed as a setting. We choose DynamicUVLM as a suitable flow solver for detailed flow simulations or a potential flow model like the Vortex Lattice Method for quicker, low-fidelity results. The solver setup code developed in Python is mentioned in Appendix A.3.

3.3.5 Structural Solver

NonLinearDynamic solver provides an interface to the structural library *xbeam* and updates the structural parameters for every time step of the simulation. This solver is called as part of a standalone structural simulation, mentioned in Appendix A.3.

3.3.6 Aerostructural Coupling

The *DynamicCoupled* solver couples the chosen aerodynamic and structural solvers to advance the aeroelastic system's solution over time. To use the *DynamicCoupled* solver, an instance of the *StaticCoupled* solver must be included in the SHARPy solution flow when defining the problem case to reach steady state equilibrium. Input data can be received from external controllers, and output data can be sent using the SHARPy network interface, which is configured through the networks setting of this solver.

3.3.7 Time Integration Solver

In SHARPy, *NewmarkBeta* and *GeneralisedAlpha* are time integration methods used to solve the equations of motion for dynamic simulations, particularly in aeroelastic problems. The *Newmark-Beta* method is an explicit or implicit time-stepping algorithm used for solving the equations of

motion in dynamic systems. It is often used in structural dynamics and aeroelastic simulations where a reliable and stable method is needed for time integration. *NewmarkBeta* is simpler and is often used when moderate accuracy and stability are sufficient. It is easier to implement and is commonly used in smaller, less complex systems. *GeneralisedAlpha* provides better control over high-frequency modes and is often chosen for more complex systems, especially those involving fluid-structure interaction, where numerical stability can be challenging. Since higher frequency modes are not important in this research, since flutter is a low frequency phenomenon by nature, *NewmarkBeta* is selected in this project.

3.3.8 Simulation

- **Initialization**: Initialize the simulation with initial conditions for the flow field and structural position (e.g., stationary or specific displacement).
- **Execution**: Start the *StaticCoupled* to compute deformed equilibrium position under steady aerodynamic loading. Next, start transient simulation by setting *DynamicCoupled*, allowing the system to evolve over time. At each time step, the aero and structural solvers exchange data based on the defined coupling method.
- **Monitor Convergence**: Check convergence at each time step using the output screen and ensure that loads, displacements, and flow fields are updating correctly.

The developed code containing commands to run the simulation is mentioned in Appendix A.4.

3.3.9 Post-Processing and Analysis

- **Result Extraction**: Extract key results such as pressure distributions, structural displacements, stresses, and aerodynamic forces over time in text file format. Use post-processor modules including *AerogridPlot*, and *BeamPlot* in SHARPy to visualize the results in Paraview.
- **Response Evaluation**: Analyze transient responses like oscillations, damping behavior, and deformation patterns in *Paraview*. For gust response, check how the structure reacts to changes in aerodynamic loads.
- **Validation and Comparison**: Compare simulation results with experimental data (if available) or theoretical predictions to validate the simulation.

In terms of post processing, SHARPy can deliver aerostructural information in text files. For instance, structural information including displacement, rotation, and derivatives are called through this command mentioned in the Appendix A.3: structure_variables = [' pos ' , 'psi ' , ' psi_dot '].

3.4 Pazy Wing Benchmark Model

In this section the Pazy Wing is described with a focus on its main geometric, design, and manufacturing features, presented in Table 3.2. The Pazy wing is a flexible wing designed for wind tunnel aeroelastic experiments, serving as a benchmark for flexible wing research. It has a chord length of 100 mm, a span of 550 mm, and uses a NACA 0018 airfoil rib profile. The wing is covered with Oralight foil, a polyester commonly used in radio-controlled drones. A 300 mm long, 10 mm diameter wing-tip rod, also 3D printed, is used to attach weights that adjust the dynamic properties and flutter speed. The total mass of the wing is 0.32 kg, including the wing-tip rod, but excluding the base used for attaching it to the wind tunnel floor.

Table 3.2: Pazy Wing Properties [32]

Property	**Value**	**Unit**
Semi-span	0.55	m
Chord (without tip rod)	0.1	m
Reference surface area (without tip rod)	0.055	m^2
Clamping section height	0.021	m
Clamping section diameter	0.12	m
Tip rod length	0.3	m
Tip rod diameter	0.01	m
Airfoil	NACA 0018	-
Chassis / Rib structure	Nylon (Polyamide 12), 3D printed	-
Covering	Oralight (polyester shrink film, 12 μm)	-

3.5 Technical Challenges and Special Features

Providing a transient aerostructural analysis near the flutter speed poses several challenges, including solver divergence, predicting the threshold of unstable conditions, gust profile setup, and

computational time. Incorporating the effects of geometric deformation during the modal study is also important. This is addressed by activating the *StaticCoupled* solver before running the *Modal* solver. Another challenge involved the inability to capture a dynamic response due to changes in the gust profile, as shown in Figure 3.4. This issue was resolved by setting *relative_motion : True* in the solver configuration to account for a non-stationary gust profile applied to the clamped cantilevered wing in a *no free-flight* condition. Figure 3.5 demonstrates that the wing can only be excited by a gust profile beyond the flutter speed, indicated by the dotted line. This issue arose because the initial geometry had a zero angle of attack, resulting in no aerodynamic excitation before reaching the flutter speed. This issue was fixed, and the wing experienced expected oscillations as it became excited by the gust profile, as shown in Figure 3.6. Note that it is crucial to initiate the gust profile excitation only after the wing reaches a steady-state condition. This will be considered for the cantilevered and ORCA wing during dataset generation.

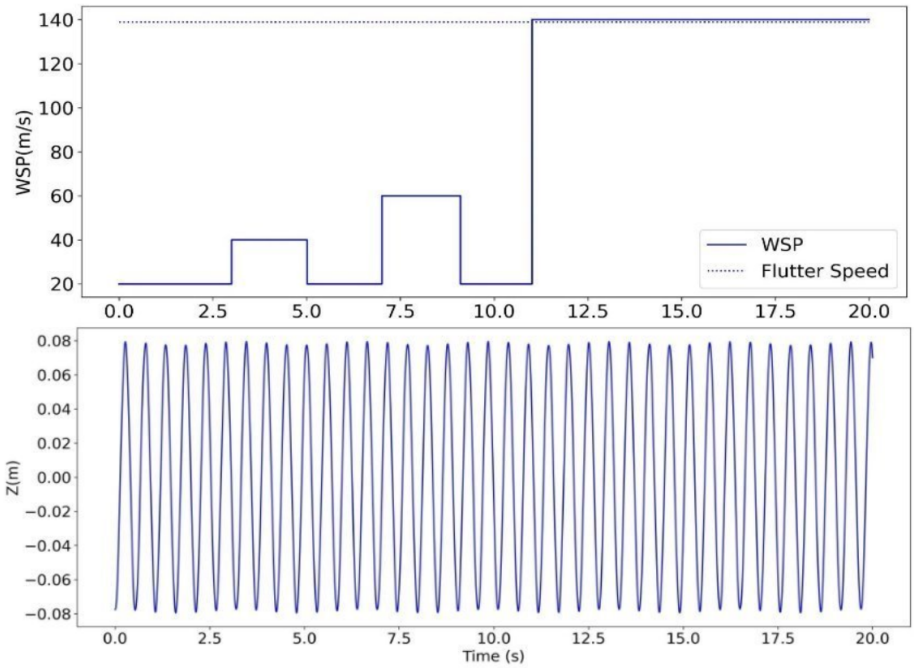


Figure 3.4: Dynamic response issue: No excitation due to stationary gust input, resolved using *relative_motion = True*.

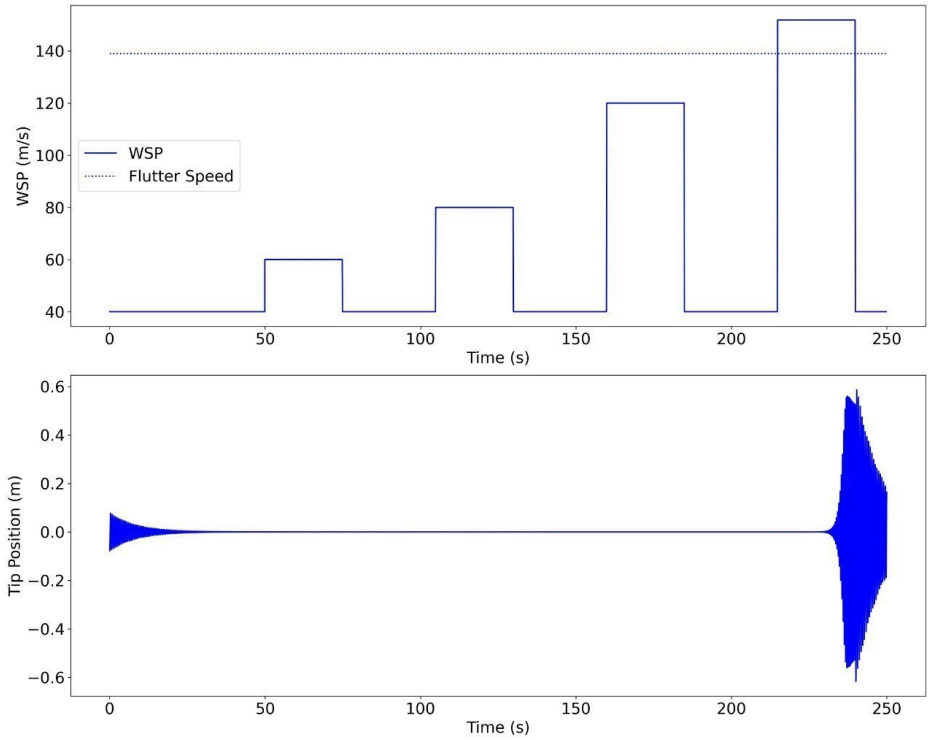


Figure 3.5: Onset of flutter only beyond flutter speed due to zero initial AoA, requiring high-speed gust input.

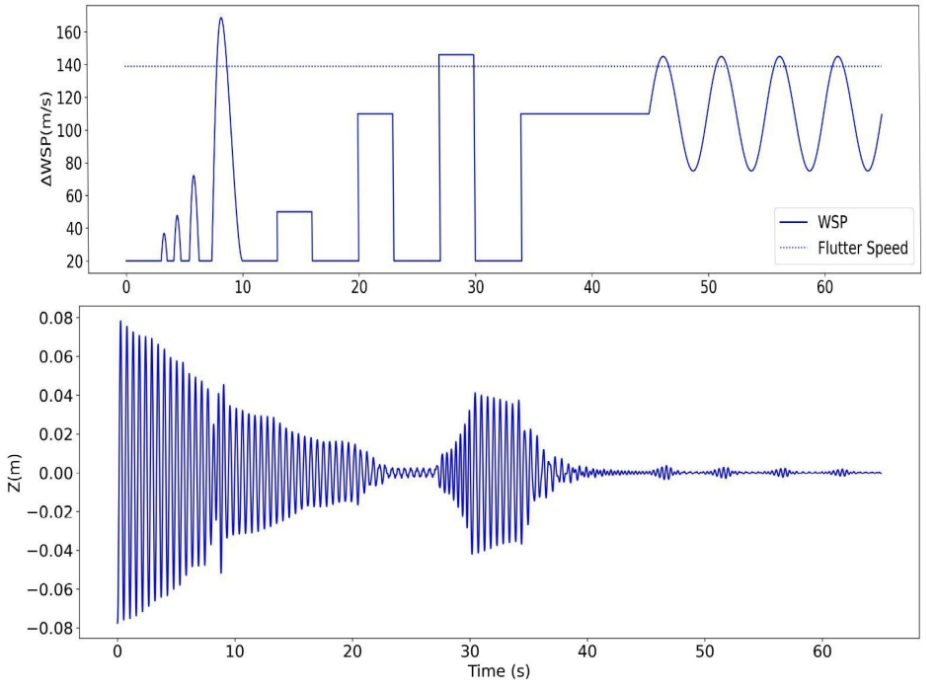
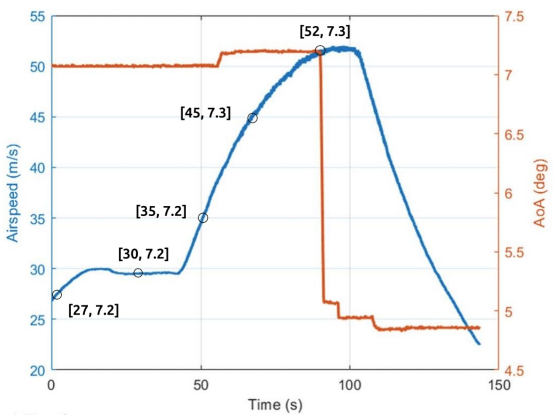


Figure 3.6: Need for steady-state convergence before applying gust excitation in transient simulations.

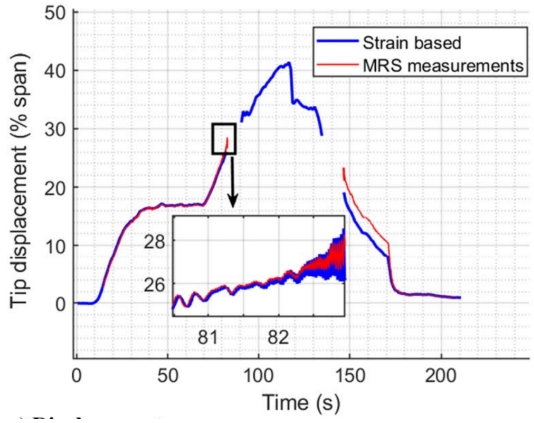
3.6 Results

One of the main objectives of this project is to validate the transient simulation against well-known experimental tests. The Pazy wing serves as an ideal model for this purpose, as numerous recently published experimental studies provide detailed insights into its aerostructural behavior. [32, 33, 42]. Figure 3.7 presents the experimental results for the Pazy wing under varying angles of attack (AoA) and airspeed in a wind tunnel. Specifically, wing tip deformation was measured using strain-based methods and the Modal Rotation Method (MRM). Figure 3.7-B displays sample data from Test 1, shown in Figure 3.7-A, obtained from a single strain sensor located 127 mm from the root on the main spar. The data is plotted as a function of time alongside the corresponding airspeed variation. Since simulating the Pazy wing for 150 seconds in SHARPy requires more than two days of computational time, selected sample points, marked in Figure 3.7-A, were chosen to perform 1-second transient simulations. For each sample point, the wing tip deformation was recorded after the system reached a steady-state condition. The results of these 1-second simulations for the selected sample points are presented in Figure 3.8 and Figure 3.9.

Before analyzing the transient response of the wing near flutter, it is essential to determine an accurate flutter speed. This can be achieved by performing a modal analysis in SHARPy and examining the eigenvalues of the aeroelastic system. Figure 3.10 illustrates the results of this analysis. The damping ratio of the aeroelastic system exhibits a characteristic behavior as a function of airspeed, as shown in Figure 3.10B. When flutter occurs, the damping ratio decreases to zero and becomes negative for the unstable mode, indicating that oscillations grow over time. Simultaneously, the real part of the eigenvalues, shown in Figure 3.10A, crosses into the positive region, signifying the onset of dynamic instability. For stability, the damping ratio of all modes must remain positive, meaning oscillations decay over time. In Figure 3.10B, it is evident that only one mode crosses the horizontal axis, indicating that only the first bending mode becomes unstable under these conditions. As airspeed increases, aerodynamic forces reduce the damping ratio of certain modes, potentially leading to instability. Some modes may initially appear stable but transition to unstable as airspeed increases. This behavior is exemplified by the first bending mode in Figure 3.10B, which transitions from a stable to an unstable condition.

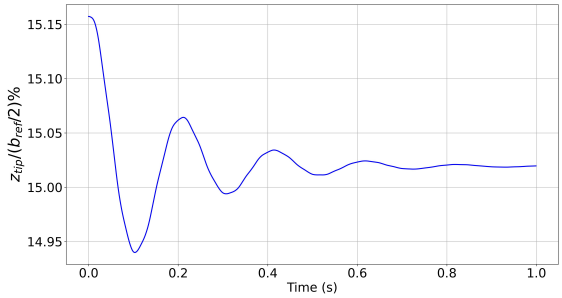


(a) A: Air speed and AOA variation

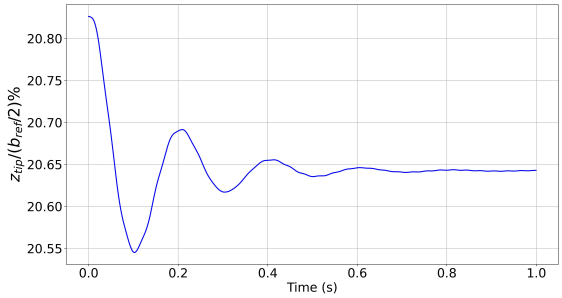


(b) B: Wingtip vertical displacement

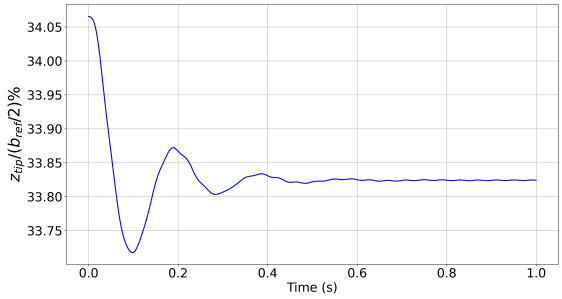
Figure 3.7: Experimental results of the Pazy wing measured by FBG and MRS [32]



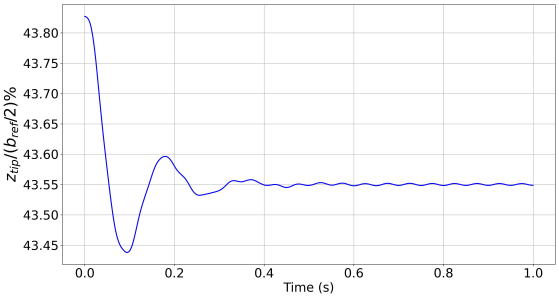
(a) [V=30,AOA=7.2]



(b) [V=35,AOA=7.2]



(c) [V=45,AOA=7.2]



(d) [V=52,AOA=7.2]

Figure 3.8: Wingtip displacement of the Pazy wing at different speeds

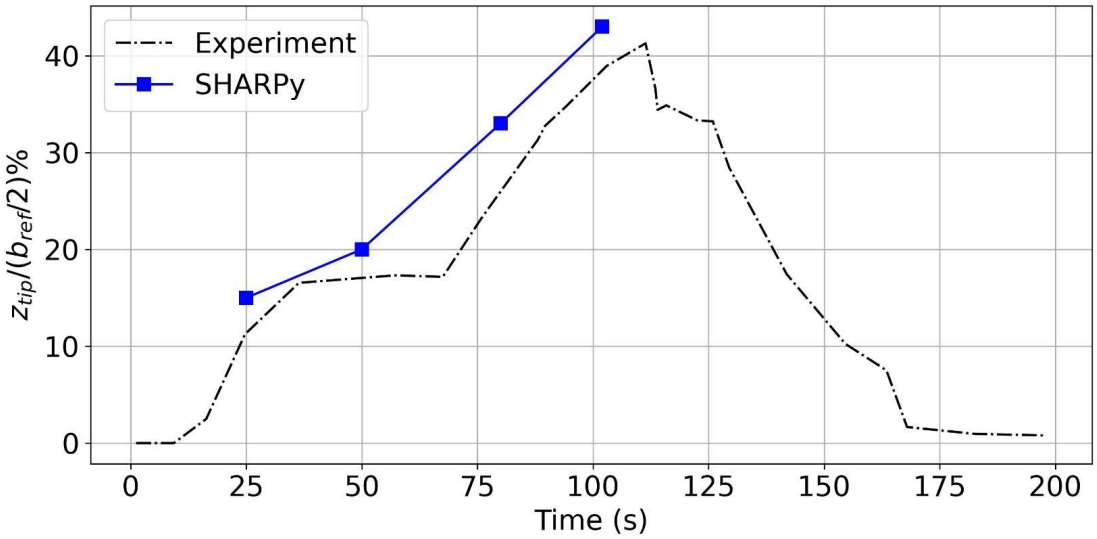
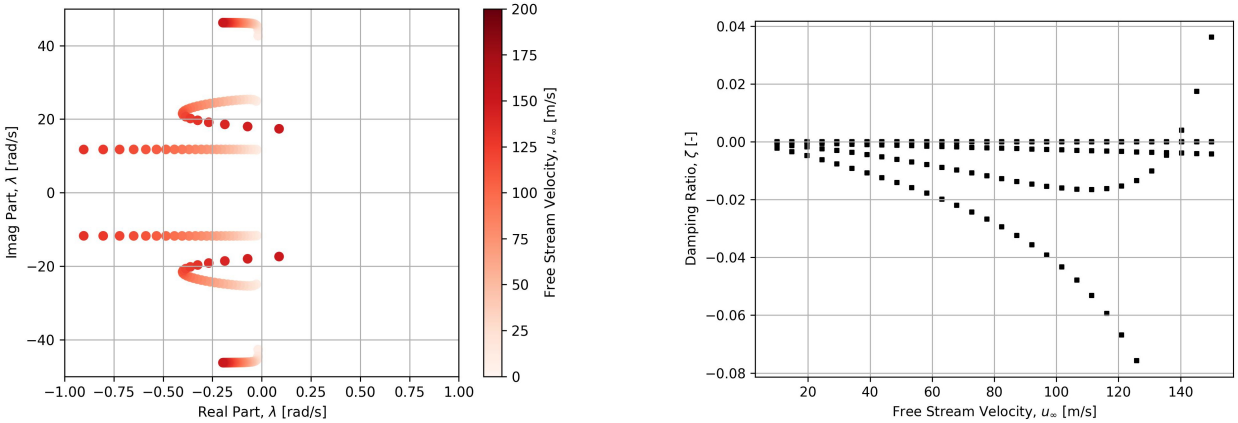


Figure 3.9: Comparison of wing tip displacement calculated by SHARPy and measured in the experimental article [32]



(a) Test A

(b) Test B

Figure 3.10: Aeroelastic modal analysis of the cantilevered wing

Figure 3.11 to 3.14 illustrate the dynamic response of the cantilevered wing to various gust profiles applied in the X-direction. In Test A, the gust profile excites the wing using a combination of step functions and sine waves, remaining below the flutter speed, indicated by the dotted line in the figure. After the system reaches a steady-state condition around 25 seconds, the excitation begins at 50 seconds, resulting in an underdamped yet stable oscillation. The mean amplitude of the wing tip oscillations increases by approximately 0.25 units compared to the wing's initial position. This increase is attributed to the positive angle of attack, which generates additional lift due to the higher velocity at 50 seconds.

Test B demonstrates a scenario where the airspeed exceeds the flutter speed for specific step functions. As shown, the wing tip amplitudes grow exponentially around 170 seconds when the airspeed surpasses the flutter speed. However, the amplitudes begin to decay after the airspeed is reduced to an underdamped condition (70 m/s).

Test C involves excitation through step functions applied both below and beyond the flutter speed. The distinct phenomenon observed here is LCO. It is an aeroelastic phenomenon in which a structure, such as an aircraft wing, undergoes sustained, periodic oscillations due to the interaction of aerodynamic, structural, and inertial forces. Unlike flutter, which can result in destructive divergent oscillations, LCO represents a stable condition where the oscillation amplitudes reach a steady value. This behavior is illustrated in Figure 3.15A within the time frame of 215 to 240 seconds.

Additionally, transient response delay is another non-linear phenomenon that can occur for a certain geometry. There might be, indeed, a time shift between the step function velocity input and the exponential growth in the wingtip amplitude (flutter), shown in Figure 3.15B, which is due to the dynamic response of the system. Flutter is a coupled phenomenon involving aerodynamic, elastic, and inertial forces. After the step velocity input, it takes some time for sufficient energy to transfer into the flutter mode, causing the exponential growth in amplitude.

Since SHARPy can capture nonlinear effects in aerostructural simulations, it is expected to observe flutter under certain excitations. Test D demonstrates a scenario where the wing is excited with step functions beyond the flutter speed, resulting in the onset of flutter corresponding to each step.

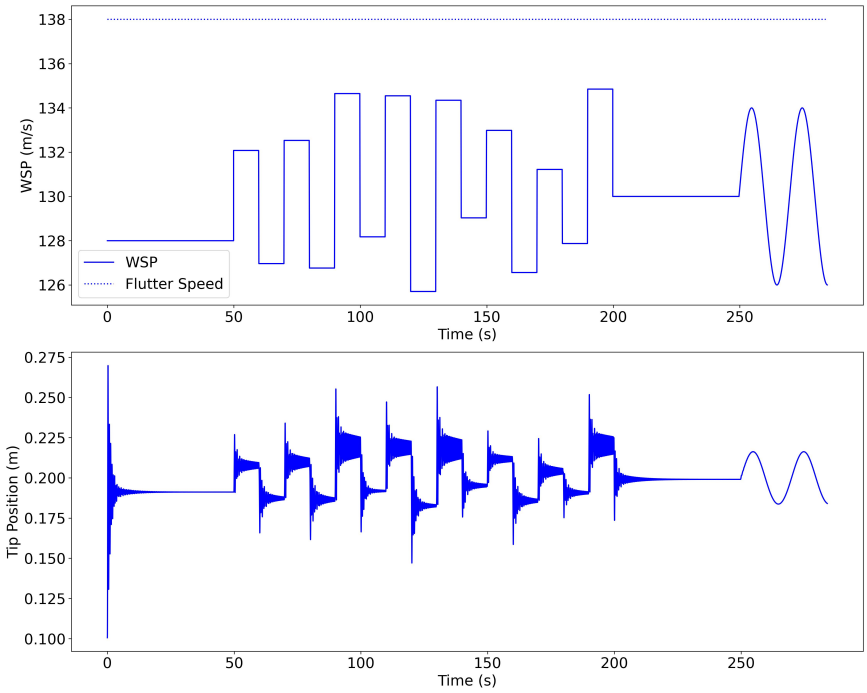


Figure 3.11: Dynamic response of the wing for Test A gust excitation

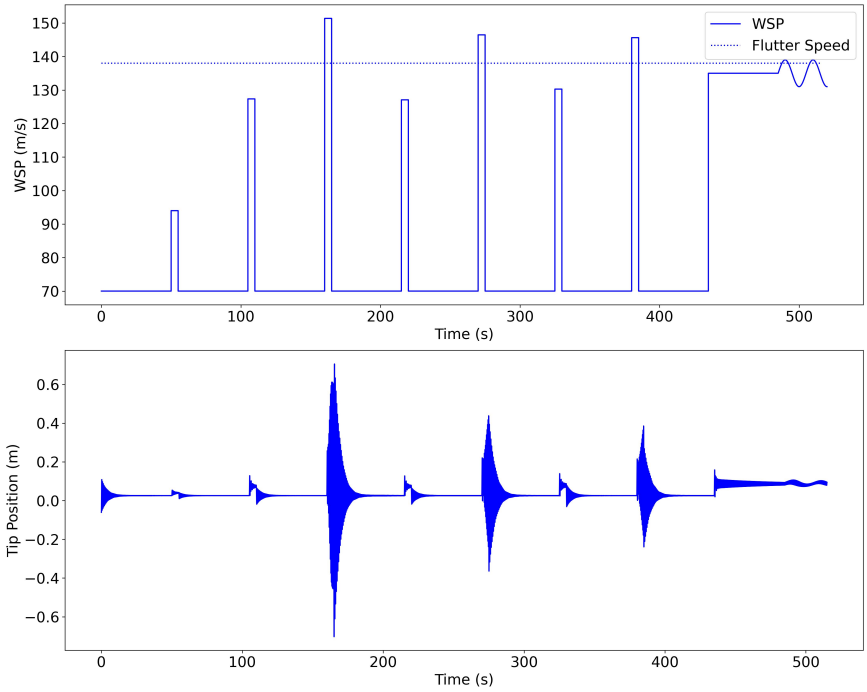


Figure 3.12: Dynamic response of the wing for Test B gust excitation

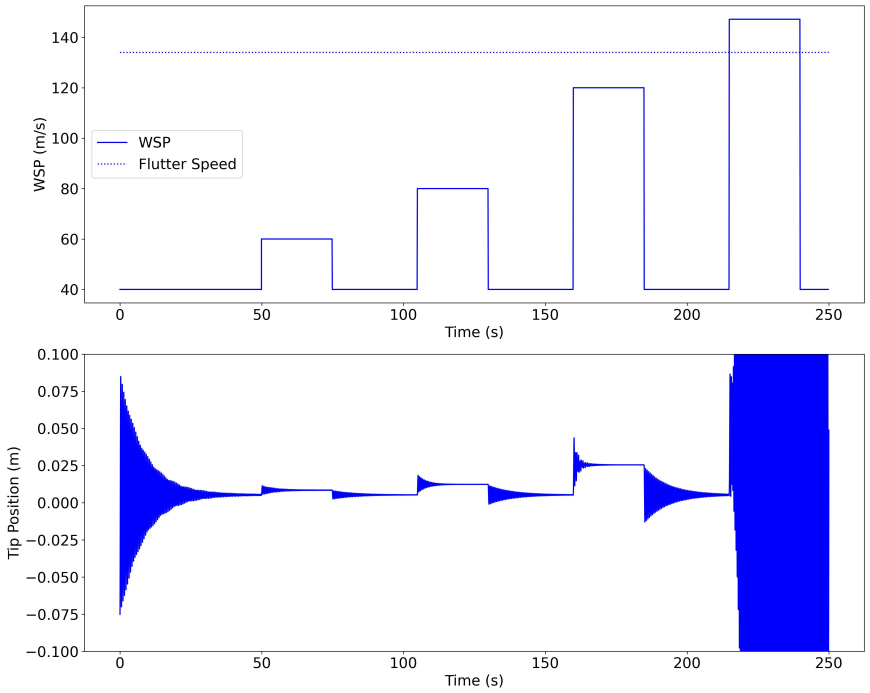


Figure 3.13: Dynamic response of the wing for Test C gust excitation

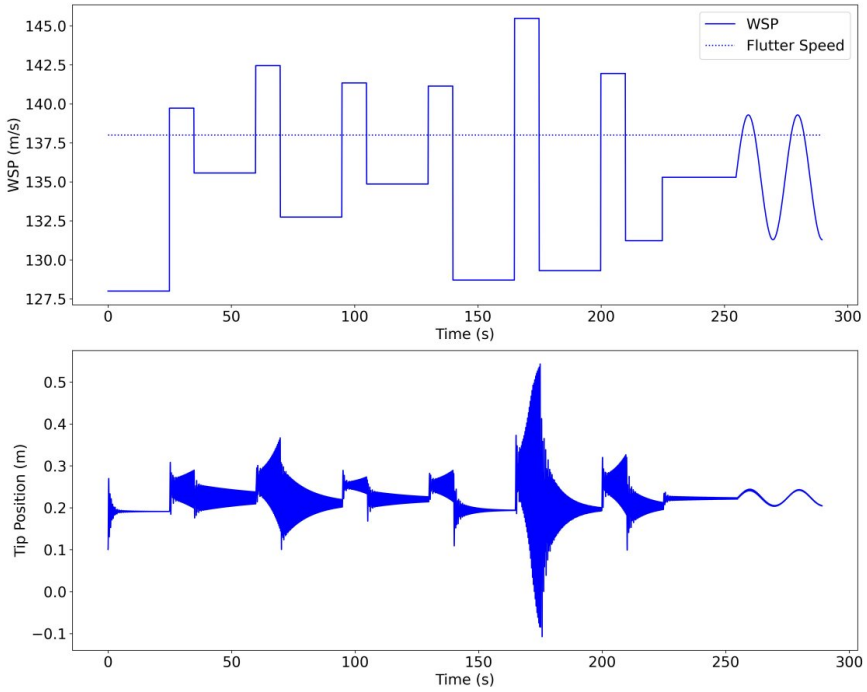
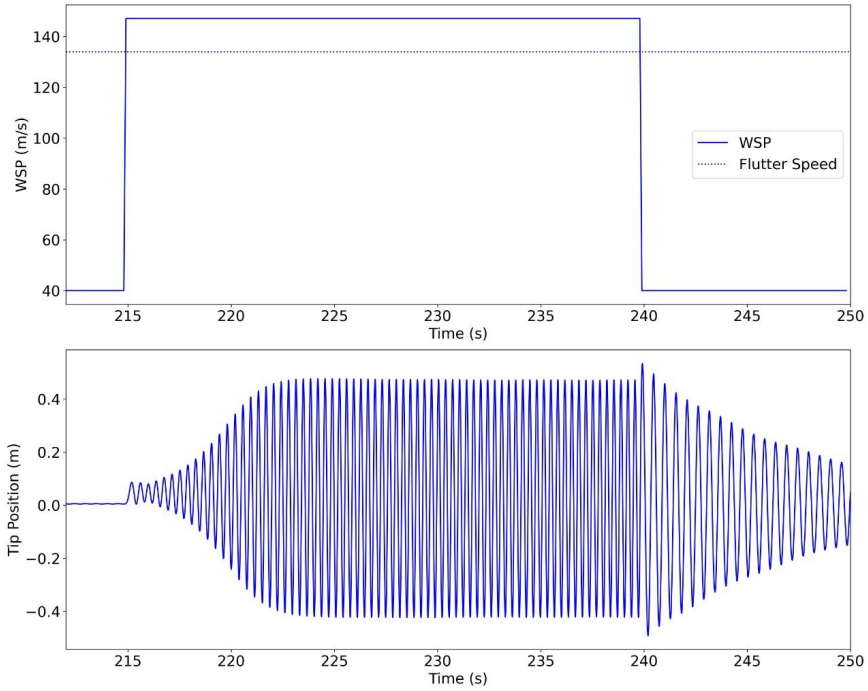
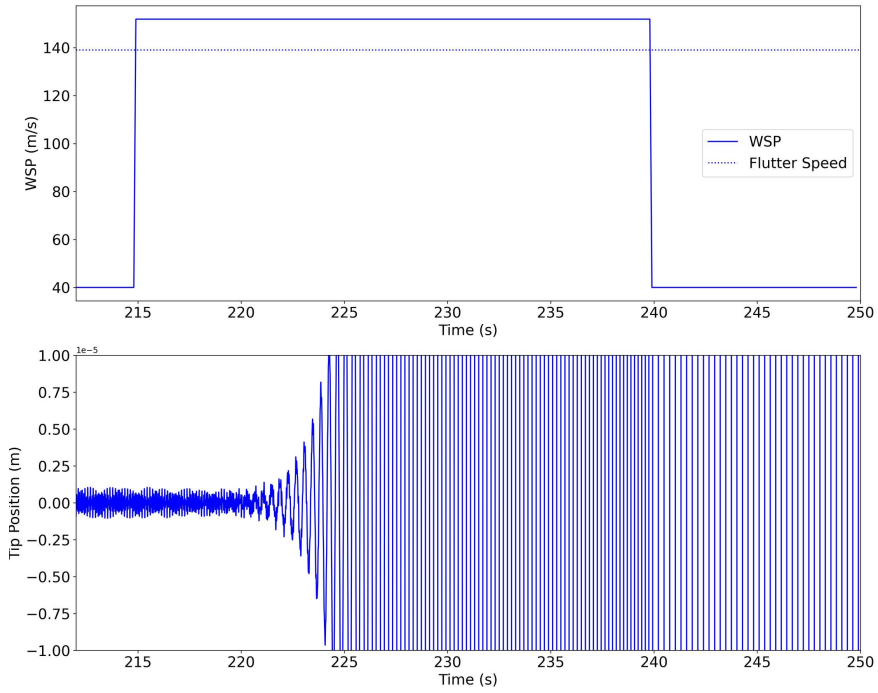


Figure 3.14: Dynamic response of the wing for Test D gust excitation



(a) A:LCO of the wing captured from Test C



(b) B:Transient response delay for Test C in Figure **??**

Figure 3.15: Non-linear phenomenon captured in the transient simulation of the cantilevered wing

SHARPy does not offer a graphical interface for visualizing wing structural deformation, aerodynamic loads, or other results directly. However, it provides Python modules that can extract

aerostructural information and export it as text files. These text files can be imported into visualization tools like *Paraview* to analyze the results graphically. For example, Figure 3.16 shows the structural deformation of the wing and the wake at a specific moment corresponding to the third step in Test B from Figure 3.12 .

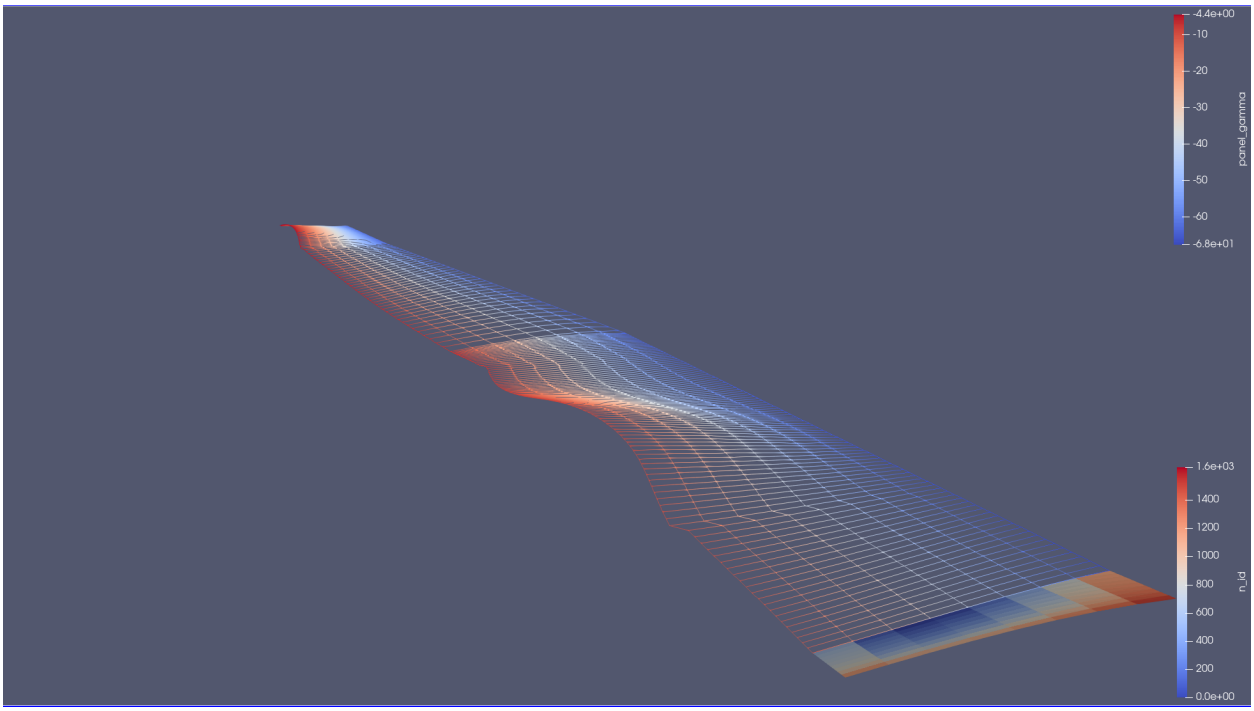


Figure 3.16: Demonstration of the deformed geometry and wake

3.7 Summary

This project focuses on using SHARPy, an aeroelastic simulation tool, to model the transient dynamics of a cantilevered wing near flutter speed. The Pazy wing, a benchmark model known for its experimental data, was chosen for validation. Key challenges included incorporating nonlinear deformations, addressing solver divergence, and capturing gust responses. The study successfully implemented transient simulations by configuring aerodynamic and structural solvers, validating results with modal analysis to identify flutter speed. The simulation captured phenomena like LCO, demonstrating SHARPy's capability to handle unsteady aerodynamics and nonlinear aerostructural coupling efficiently.

Results show that SHARPy accurately replicates experimental observations of the Pazy wing's behavior under various conditions, including flutter and gust profiles. The simulations revealed a transition from stable oscillations to flutter as airspeed increased, with damping ratios crossing into

instability. Comparisons with experimental data demonstrated strong agreement in wingtip displacement and flutter speed predictions. Additionally, the study highlighted SHARPy's ability to simulate complex aeroelastic phenomena, such as LCO, and transient response delay, under specific gust profiles, providing insights into wing stability and dynamic responses near flutter conditions. These results confirm the tool's utility for designing advanced control algorithms and analyzing flexible structures.

The SHARPy setup in Python involves configuring its modular solvers for structural, aerodynamic, and coupled simulations. For structural modeling, the NonLinearDynamic solver handles the dynamics of flexible structures, while the DynamicUVLM aerodynamic solver models unsteady aerodynamic forces. These solvers are coupled using the DynamicCoupled solver, which ensures accurate aerostructural interaction. The setup process includes defining the wing geometry, structural properties (e.g., stiffness and mass), boundary conditions, and aerodynamic panels for unsteady simulations. Time integration methods like NewmarkBeta provide efficient handling of low-frequency phenomena like flutter. Post-processing modules enable result extraction and visualization using tools like ParaView for analyzing wing deformation, aerodynamic loads, and structural behavior.

The balance of features makes SHARPy well-suited for research and preliminary design, especially for flexible high aspect ratio wing structures considering non-linearity due to large deformation, but it may require supplementary tools for low-fidelity or highly customized studies. SHARPy is ideal for scenarios where computational efficiency is critical, and nonlinear effects are significant, but it may not be the best choice for high-fidelity or fully 3D simulations requiring detailed flow resolution. SHARPy may not perform well with compact or low aspect ratio wings where plate or shell models are more appropriate. It is limited to beam-like structures for the structural model and not ideal for components needing shell or solid modeling.

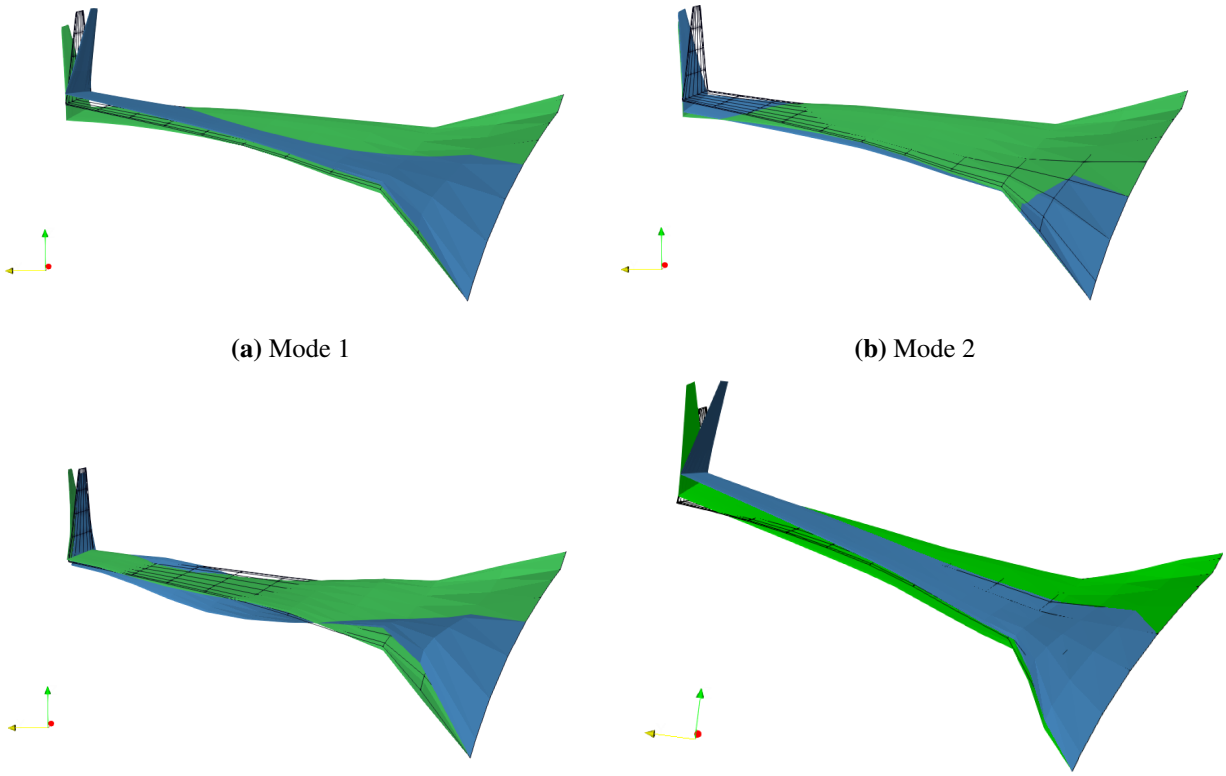
Chapter 4

Data-Driven Aeroelastic Modeling of the BWB Aircraft Wing

This chapter develops and evaluates data-driven models for predicting the aeroelastic response of a flexible cantilevered wing based on the ORCA configuration. Flutter speed is first identified using SHARPy modal analysis. The system is then excited near and beyond flutter using PRBS and sine-sweep velocity profiles to generate training datasets. Two modeling approaches—DMDc and LSTM—are trained and tested on these datasets. Results show that LSTM outperforms DMDc, especially near flutter, achieving less than 2% error within the training velocity range. Accuracy declines when extrapolating, but LSTM remains over 6000 times faster than SHARPy, making it suitable for real-time use. Finally, LSTM's integration into a MPC framework is conceptually demonstrated, highlighting its potential for flutter suppression and flight envelope enhancement.

4.1 Modal Analysis of the Wing

The first four zero-speed modes of the cantilevered wing were first determined using SHARPy at a referenced speed of 20m/s and are shown in Figure 4.1. The first and second modes represent bending and torsion deformations, with increasing complexity in displacement. The third and fourth modes correspond to torsional motion, with the second bending mode showing a less intricate twisting pattern.



(a) Mode 1

(b) Mode 2

(c) Mode 3

(d) Mode 4

Figure 4.1: First four elastic mode shapes of the BWB wing

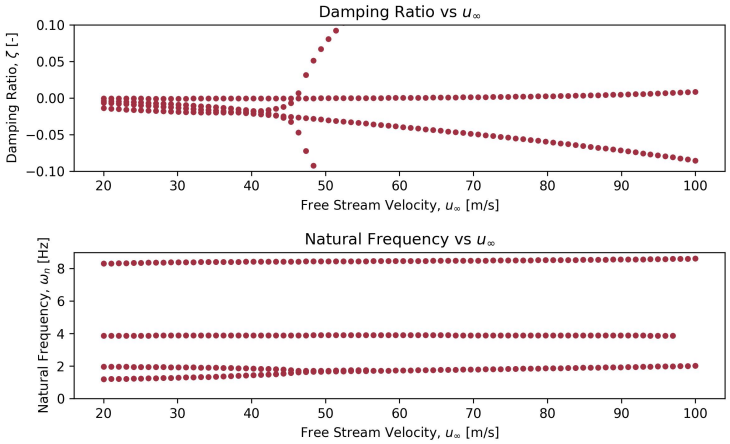
Table 4.1: Elastic modal analysis results for the first four modes

Mode Num.	f_n [Hz]
Mode 1	1.13
Mode 2	1.98
Mode 3	3.87
Mode 4	8.47

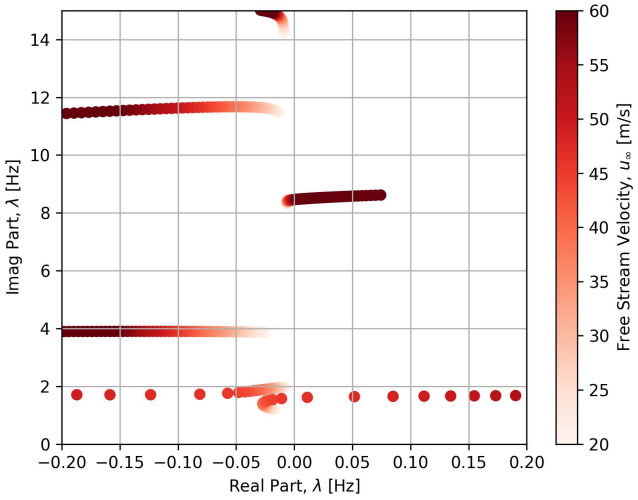
Next, the aeroelastic modes were evaluated at different airspeeds to determine the flutter speed. As mentioned before SHARPy employs the UVLM, which is an inviscid and incompressible flow solver. As such, simulations near the flutter speed—where compressibility effects may become significant—may not be fully accurate. Additionally, at higher speeds, the wing can extract more energy from the airflow, leading to increased deformation or sudden structural acceleration. Therefore, it is important to ensure that the flutter speed remains sufficiently low.

Increasing the wing's stiffness in bending and torsion (EI, GJ) raises the flutter speed around 200 m/s; however, an overly flexible wing can cause numerical divergence in the solver due to excessive deformation. One effective approach to maintaining convergence is to reduce the flutter speed by adding a lumped mass at the wing tip.

Note that only the first eight modes of the system were requested from SHARPy. The first bending and torsional modes with frequencies of approximately 1.13 Hz and 1.98 Hz, respectively, coalesce at around 45 m/s and the damping ratio of the mode becomes positive. This indicates the onset of a bending-torsion flutter mechanism.



(a) V–g–f plot of aeroelastic modes for the BWB wing



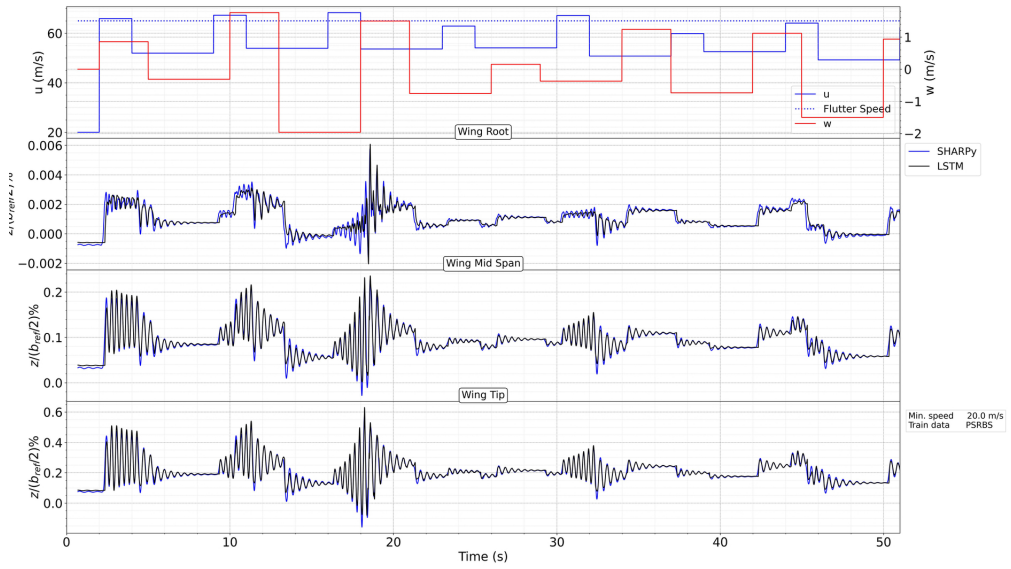
(b) Root locus of aeroelatic modes for the BWB wing

Figure 4.2: Aeroelastic modal analysis for the BWB wing with lumped mass at the wing tip

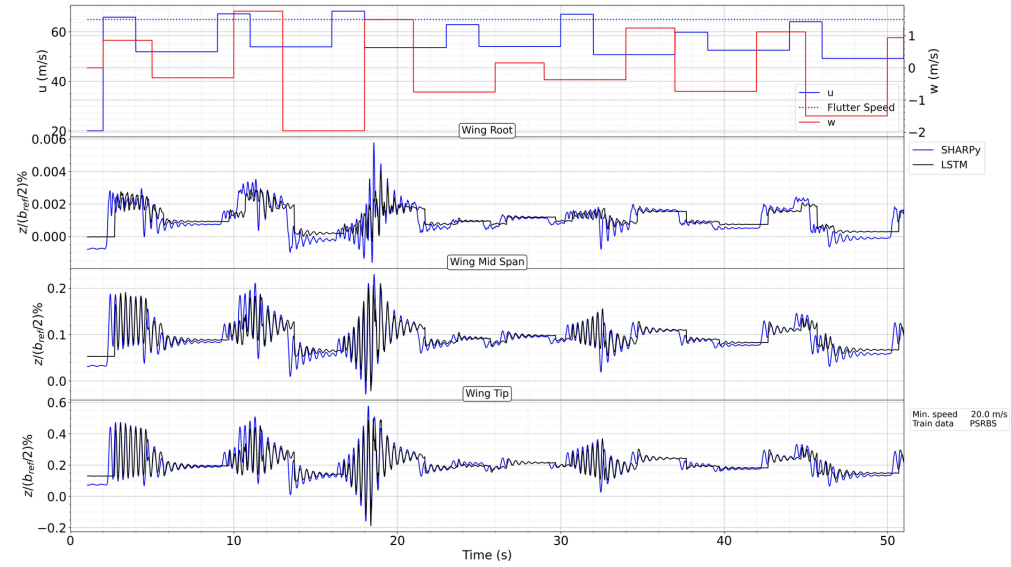
4.2 Data-Driven Aerostructural Modeling

Given the range of flutter speeds (42 to 45 m/s, as shown in Figure 4.2), the dynamical system is excited using various input signals to record its dynamic response. These responses are used to construct training and test datasets suitable for LSTM and DMDc models. One important hyperparameter during training is the prediction horizon length, which refers to the number of future time steps that the method is expected to predict the system's state. For DMDc, the horizon length is typically limited to a single time step, as discussed in detail in [13]. In contrast, the horizon length for LSTM can vary. Interestingly, the prediction performance of the LSTM model tends to decrease as the prediction horizon increases, especially when using a fixed sequence length of past data and

a specific training dataset. As shown in Figure 4.3, selecting an appropriate prediction horizon for the LSTM depends on how quickly the dynamical system responds to input excitations [44]. As the horizon length increases, the accuracy of the trained LSTM decreases when the sequence length is kept constant at 100 time steps.



(a) SHARPy vs. LSTM ($h = 10$)



(b) SHARPy vs. LSTM ($h = 200$) — extended horizon

Figure 4.3: Effects of change of the horizon length on the prediction performance of LSTM

Figure 4.4 illustrates how the LSTM, using a one-time-step prediction horizon—similar to the full-rank DMDc—predicts the wingtip displacement of the BWB aircraft for the second test dataset. As shown in the figure, the LSTM shows acceptable agreement with SHARPy, especially near the

flutter condition (around 10s), and performs better than DMDc under the same one-timestep horizon. To ensure effective MPC control, it is important to assess the LSTM performance over multi-step prediction horizons. Figure 4.5 to 4.8 illustrate the prediction performance of the trained LSTM and DMDc models across different datasets considering a prediction horizon of 50 timesteps, i.e., at each timestep, the LSTM model predicts the state vector for a window of 50 steps. Since the state vector dimension is high, the wing deformation in the z direction at three different spanwise locations is considered to compare the performance of LSTM and DMDc. To provide a clear comparison with the actual state, the predicted values shown correspond to the last element within each prediction horizon. This choice is based on the expectation that the final predicted parameters within each horizon may exhibit the highest error relative to the actual signal. Moreover, this choice enables a consistent comparison with DMDc, whose prediction horizon is limited to a single time step.

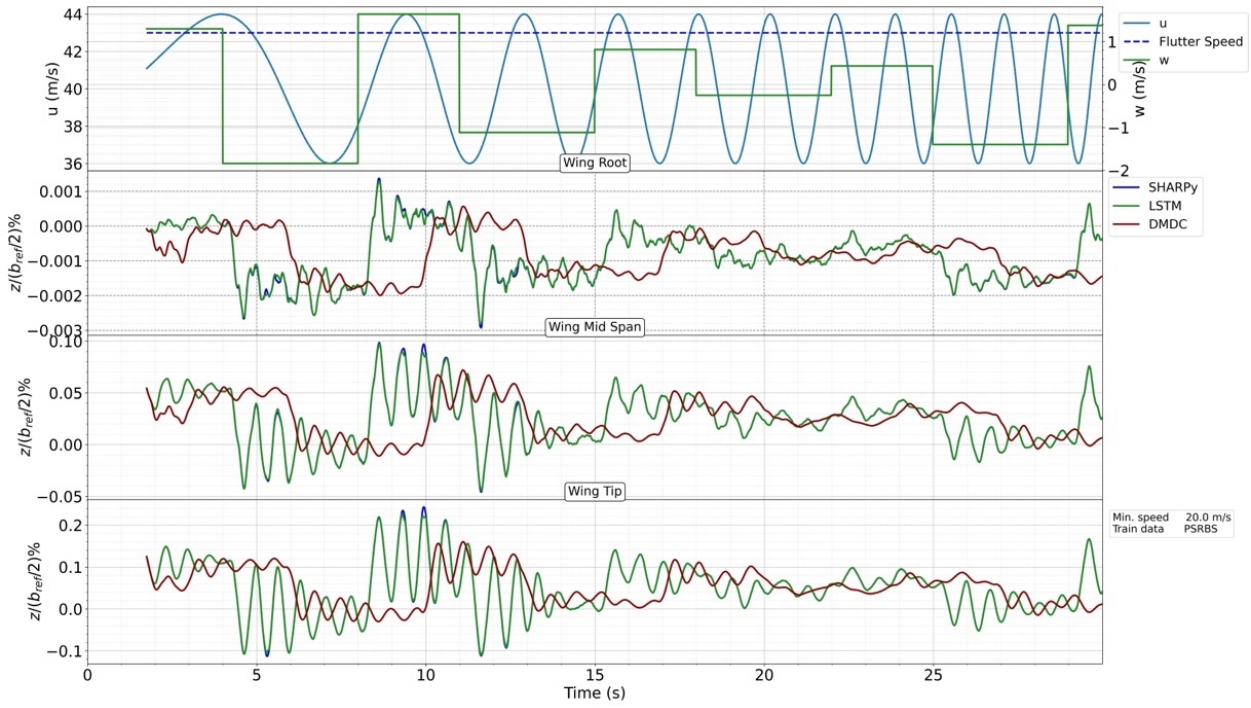


Figure 4.4: Evolution of wing transverse displacement over time at the tip, midspan, and root sections, as simulated by SHARPy and predicted by DMDC and LSTM (h=1)

The training dataset consists of the wing's state vector excited by a PRBS signal applied to velocities in different directions, as shown in Figure 4.5. Since the onset of flutter occurs at 45 m/s and the aim is to excite the system around this speed, the velocity range is set between 20 and 50 m/s for the training dataset. Furthermore, we present the state of the system at three different locations across the wingspan to evaluate the performance of the data-driven methods. As seen in the figure, the LSTM model exhibits a strong agreement with the actual data from SHARPy, outperforming

DMDc at all locations across the span during validation. The wing root deformations could be excluded from the state vector before training as the displacements were relatively small. Nevertheless, the LSTM model still performed well with such low-amplitude signal.

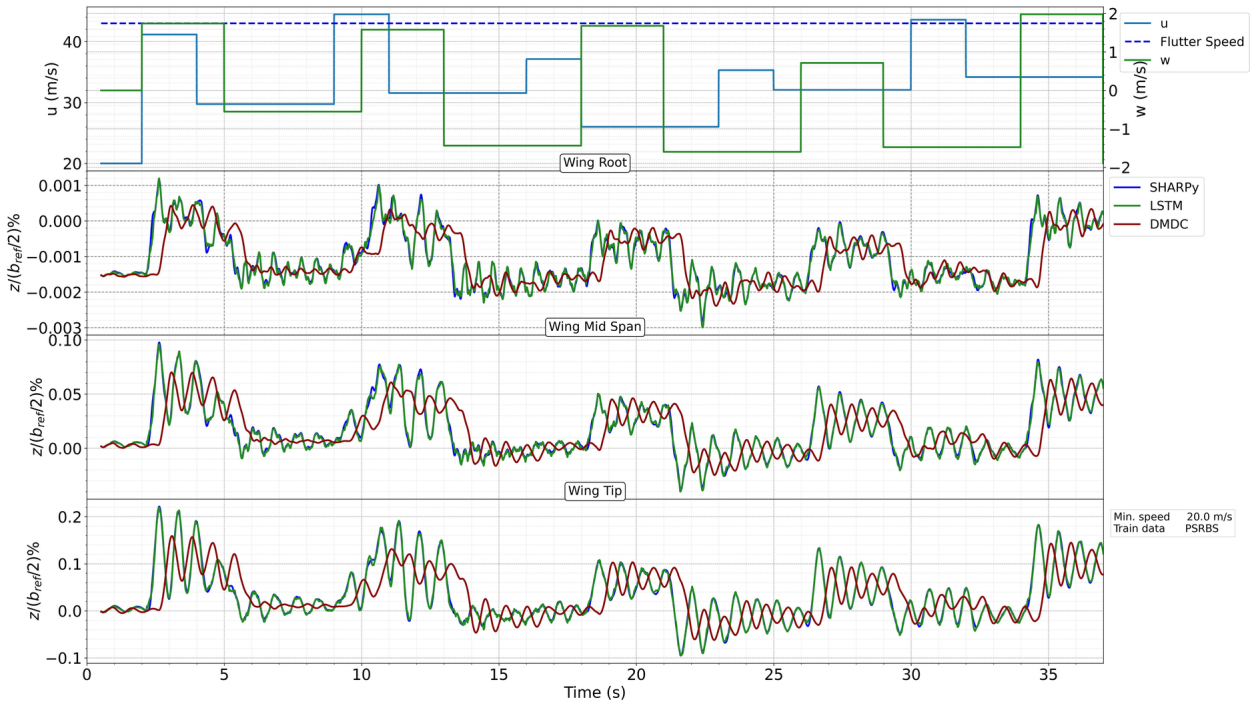


Figure 4.5: Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the train dataset

Testing the forecasting accuracy of data-driven models using test datasets that were not seen during training is essential for evaluating the generalization performance of the model [45–47]. To achieve this, the performance of the trained models is evaluated on two different test datasets. Dataset #1 is constructed using the same family of excitation PRBS signals in SHARPy, while maintaining the minimum velocity within the range of observed velocities in the training dataset. However, the maximum velocity is set to 55 m/s to examine whether the LSTM can extrapolate the system state further beyond the flutter onset. Further increase in the maximum velocity led to divergence in the SHARPy solution due to a high deformation of the wing structure of more than 30% of wingspan. As shown in Figure 4.6, the LSTM model successfully captures the overall trend of the reference data, even when the BWB wing exceeds the flutter speed at certain time instances, such as t = 30s.

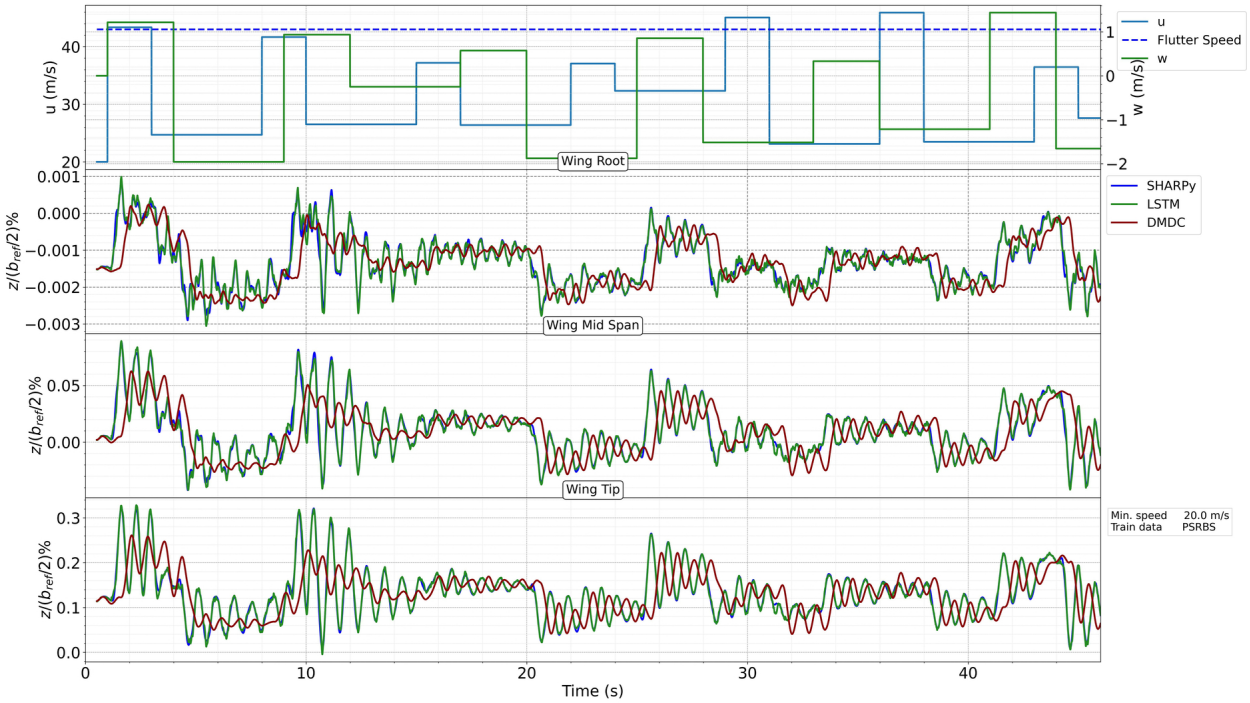


Figure 4.6: Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the test dataset #1

In addition to providing a deeper understanding of the system's behavior under different excitation conditions, the prediction performance of the data-driven models is evaluated using a third dataset generated by SHARPy, where the system is excited with a sine sweep function shown in Figure 4.7. The excitation signal range is between 5 and 55 m/s, extending beyond the velocity range in the training dataset, which was between 20 and 50 m/s. This enables a more comprehensive analysis of the system dynamics, including how the response evolves across different time scales. As shown in the figure, when the system exceeds the flutter onset point, DMDc fails to capture the system dynamics accurately. Although the LSTM model demonstrates good agreement with the SHARPy simulation, it underestimates the wingtip position over time as the frequency of the input sine sweep increases.

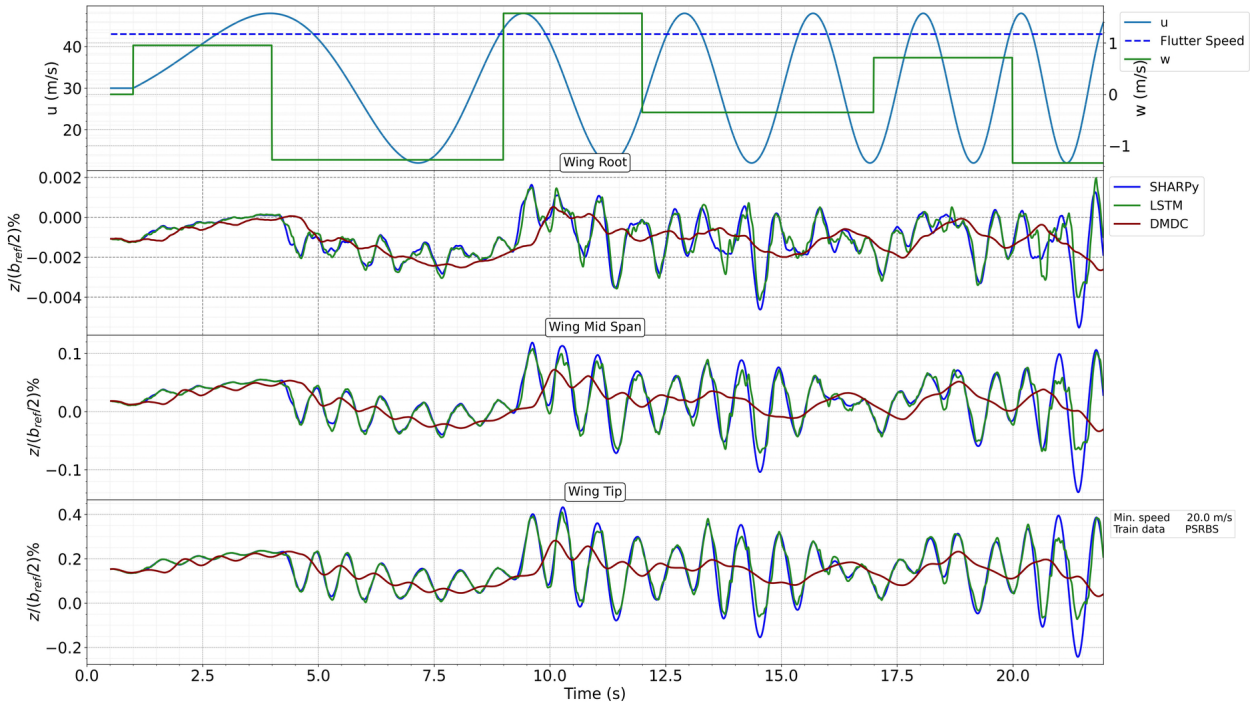


Figure 4.7: Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the test dataset #2

Figure 4.8 shows the results when the sine-sweep excitation remains within the range of the training dataset. Compared to Figure 4.7, as the frequency of excitation increases, the LSTM model is able to predict the behavior of the dynamical system with more accuracy. The key conclusion is that the LSTM model demonstrates robust performance under the tested conditions (Figure 9d). Even as the excitation frequency increases, the model does not underestimate the actual signal when the excitation remains within the range used for training. This suggests that the LSTM is effectively capturing the dynamic behavior of the system and can generalize well within the operating range, providing reliable predictions even under varying excitation frequencies.

Rotation angles about and at different spanwise locations were also predicted using the two models. As shown in Figure 4.9 to Figure 4.12, DMDc either underestimated or overestimated the evolution of the state over time, failing to accurately capture the system dynamics. In contrast, LSTM was able to accurately predict wing rotational states near the flutter speed. This shows that LSTM can provide a more efficient and accurate approach to control of aeroelastic instabilities compared to DMDc.

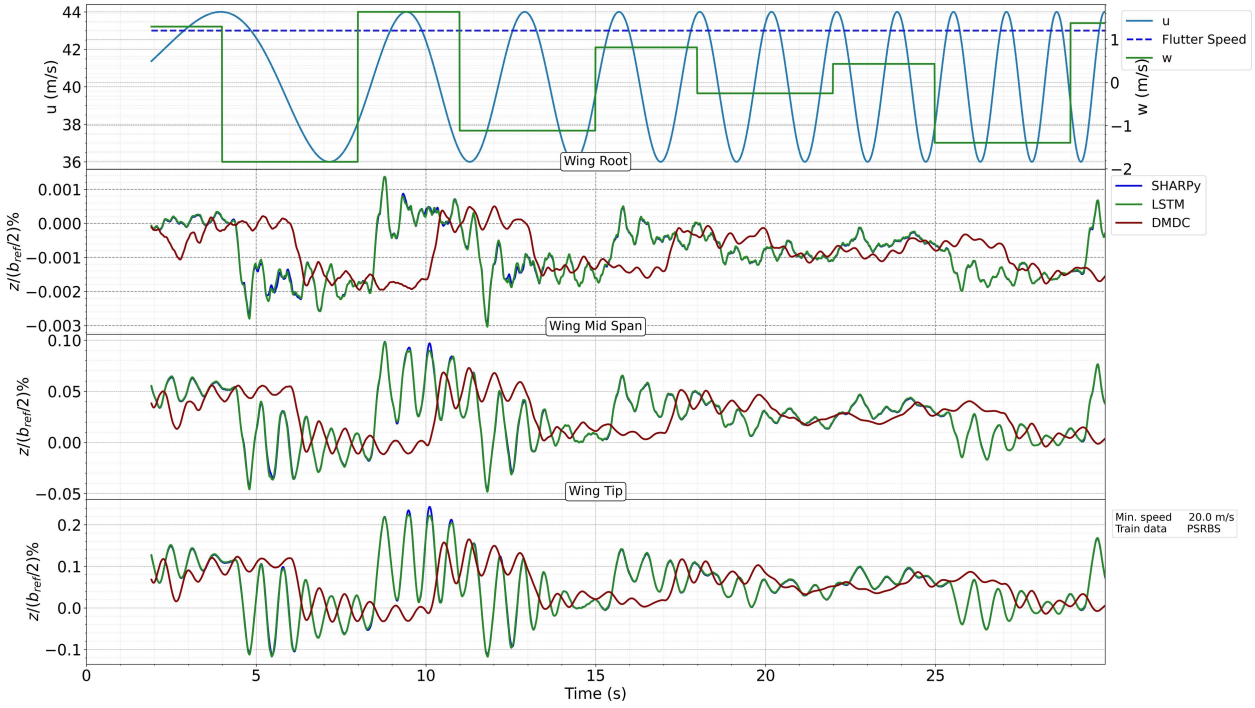


Figure 4.8: Evolution of wing displacement over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDc, and LSTM using the test dataset #3

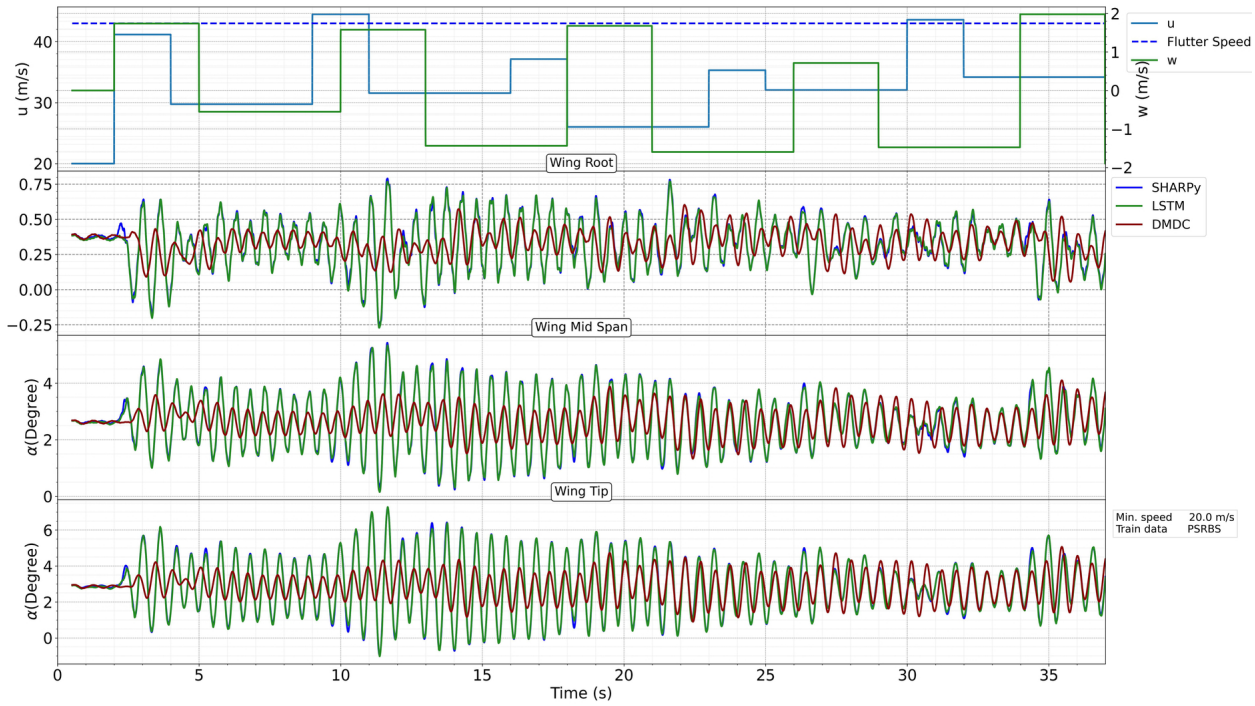


Figure 4.9: Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the train dataset

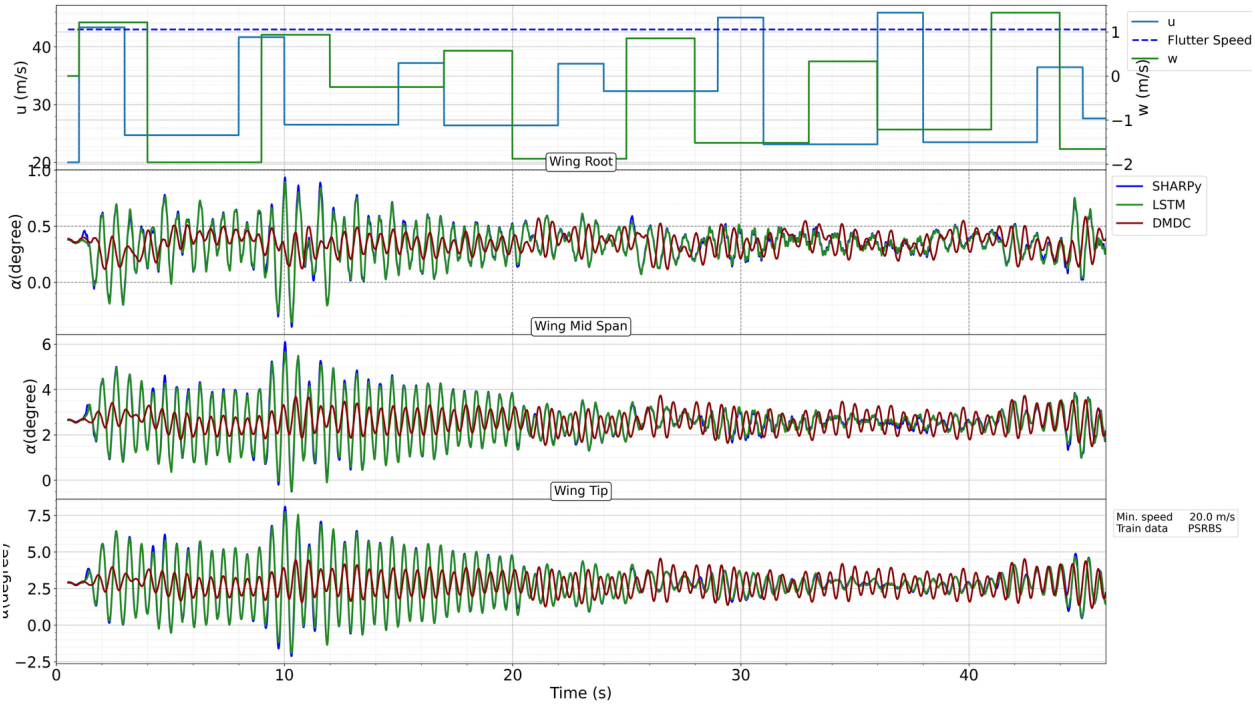


Figure 4.10: Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #1

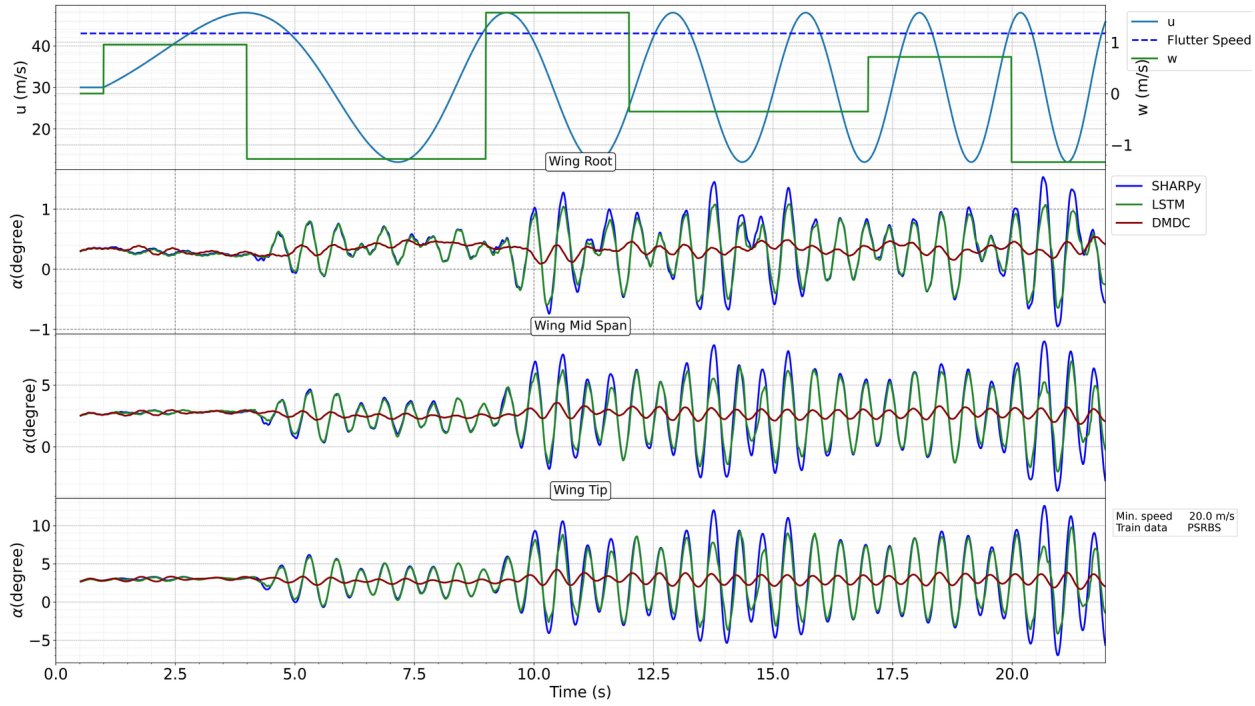


Figure 4.11: Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #2

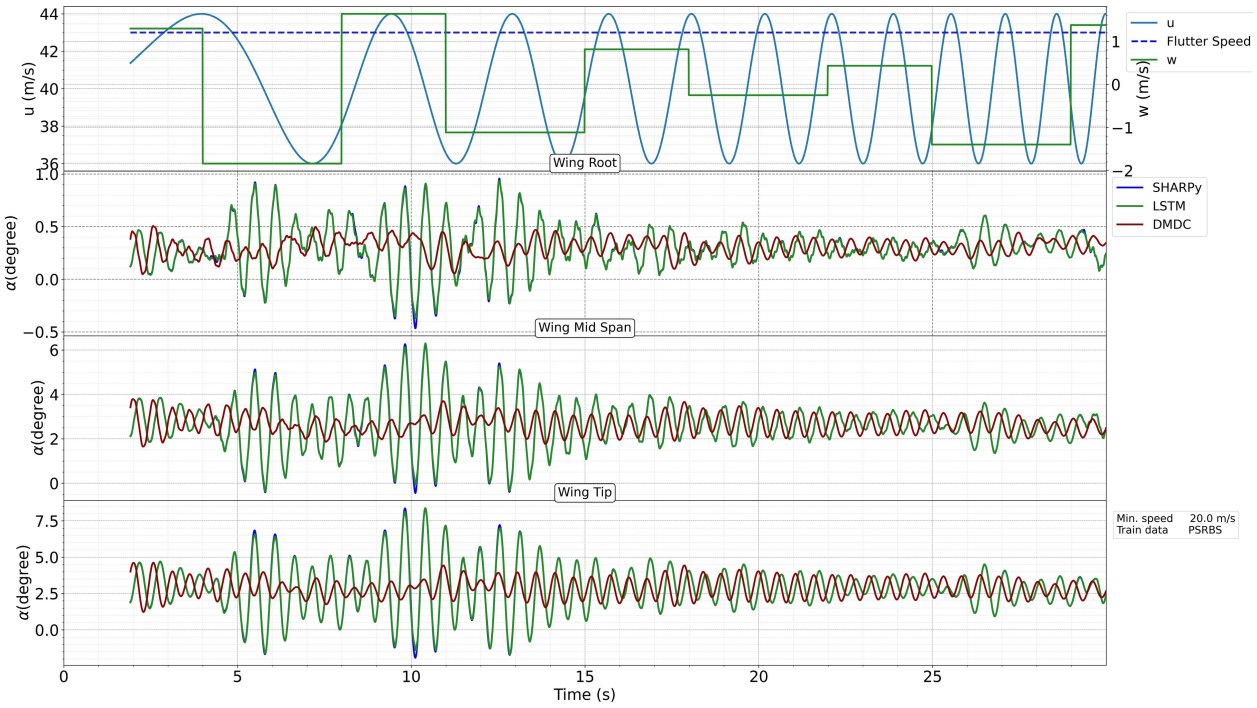


Figure 4.12: Evolution of wing torsion angle around Y over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #3

A time-domain error analysis of the LSTM-predicted bending and torsional states was also carried out. Wing tip deformations were selected—including displacement, rotation, and rotational derivatives in three directions. To perform a time-dependent error analysis, since the LSTM predicts the system state over a fixed horizon (50 timesteps) at each time step, the maximum prediction error within each horizon was computed over time for each state during both validation and testing. We conducted this analysis for all datasets highlighting errors greater than 5% in red.

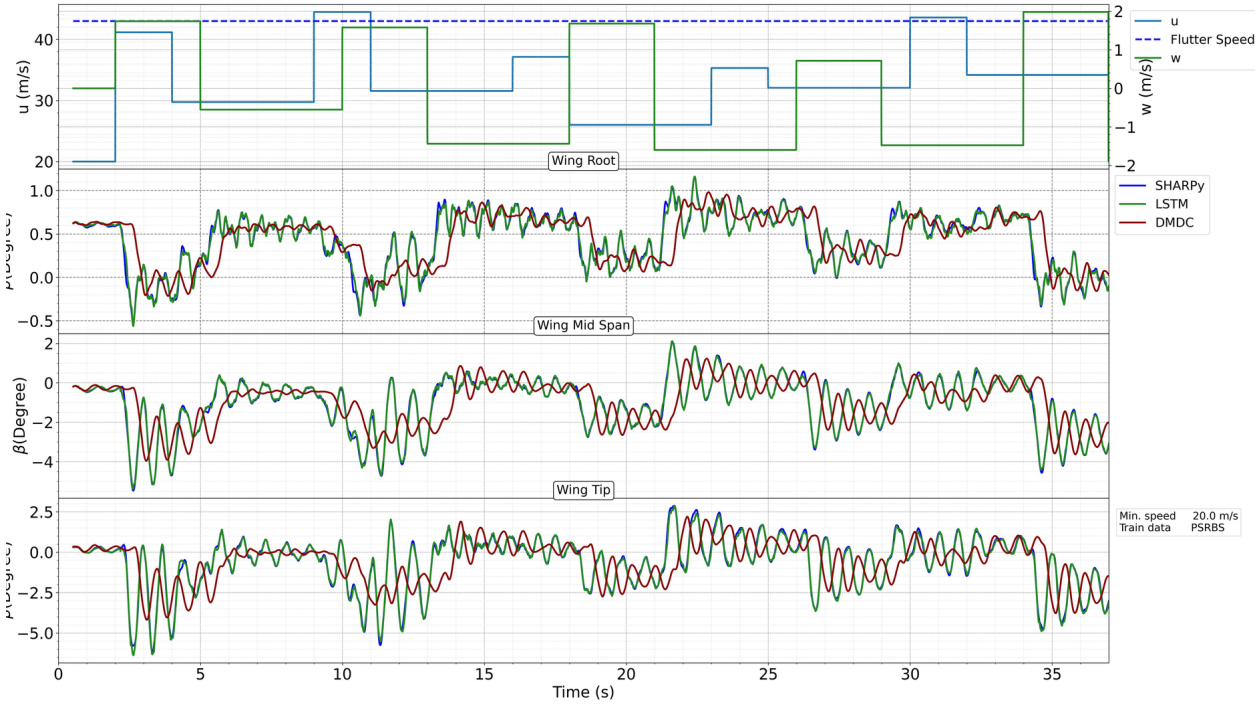


Figure 4.13: Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the train dataset

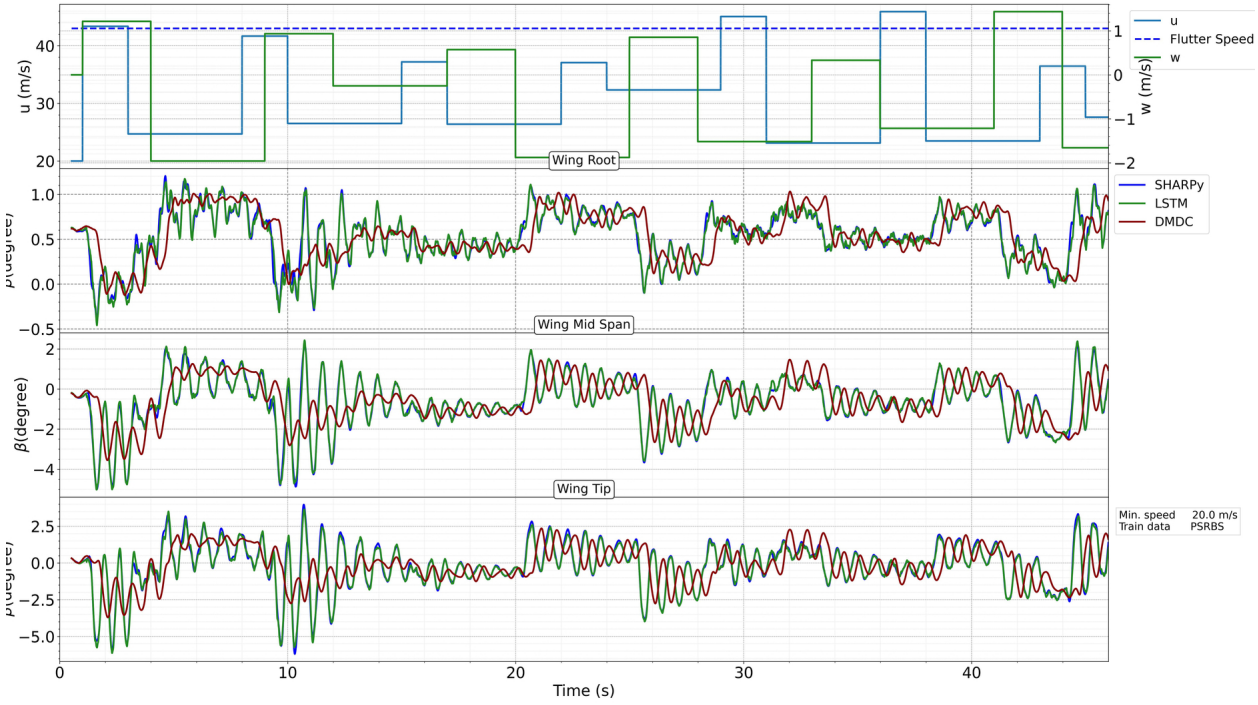


Figure 4.14: Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #1

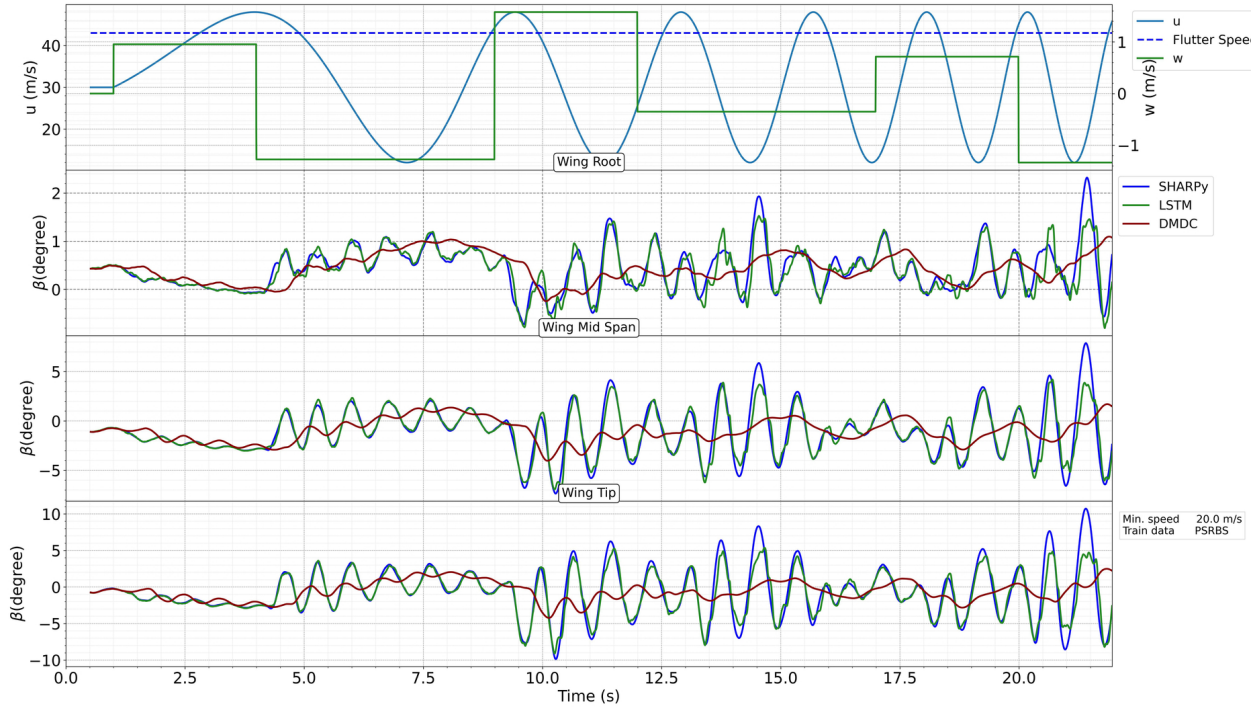


Figure 4.15: Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #2

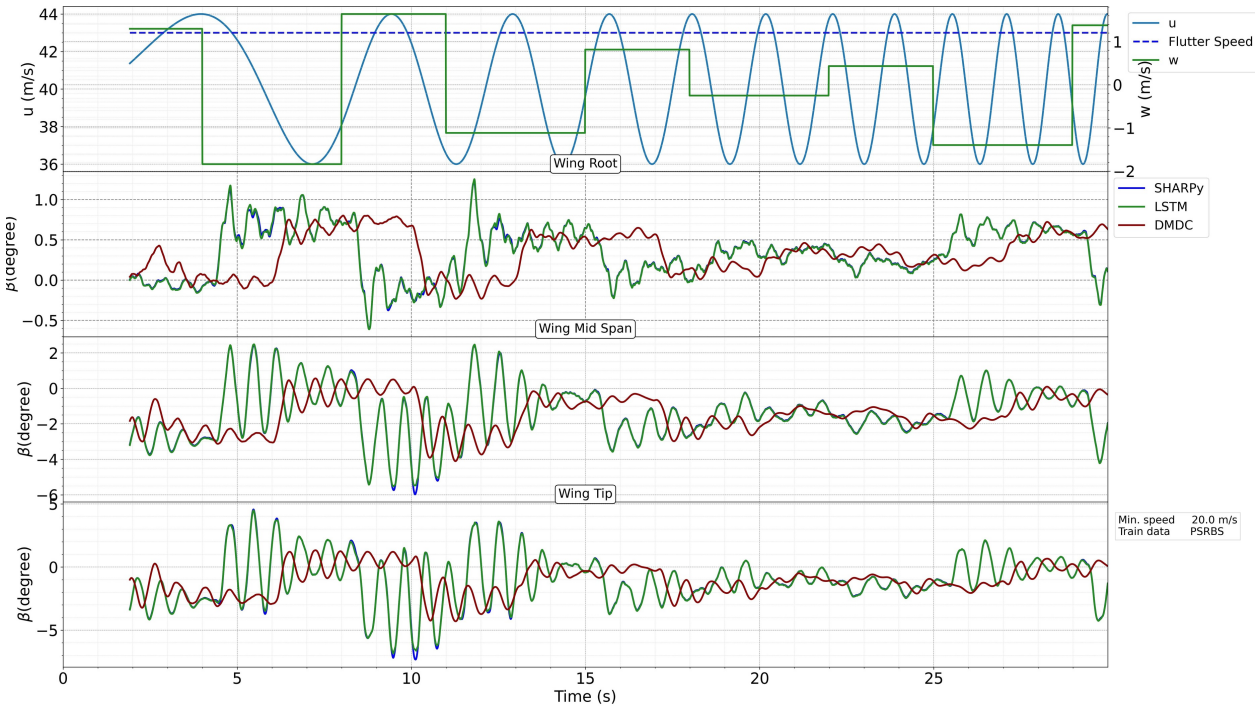


Figure 4.16: Evolution of wing torsion angle around X over time at the tip, mid-span, and root positions, as modeled by SHARPy, DMDC, and LSTM for the test dataset #3

The trained LSTM model exhibits an error of less than 1% for all structural variables, as shown in Figure 4.17. This low error indicates that the LSTM effectively captures the dynamics of the system during validation.

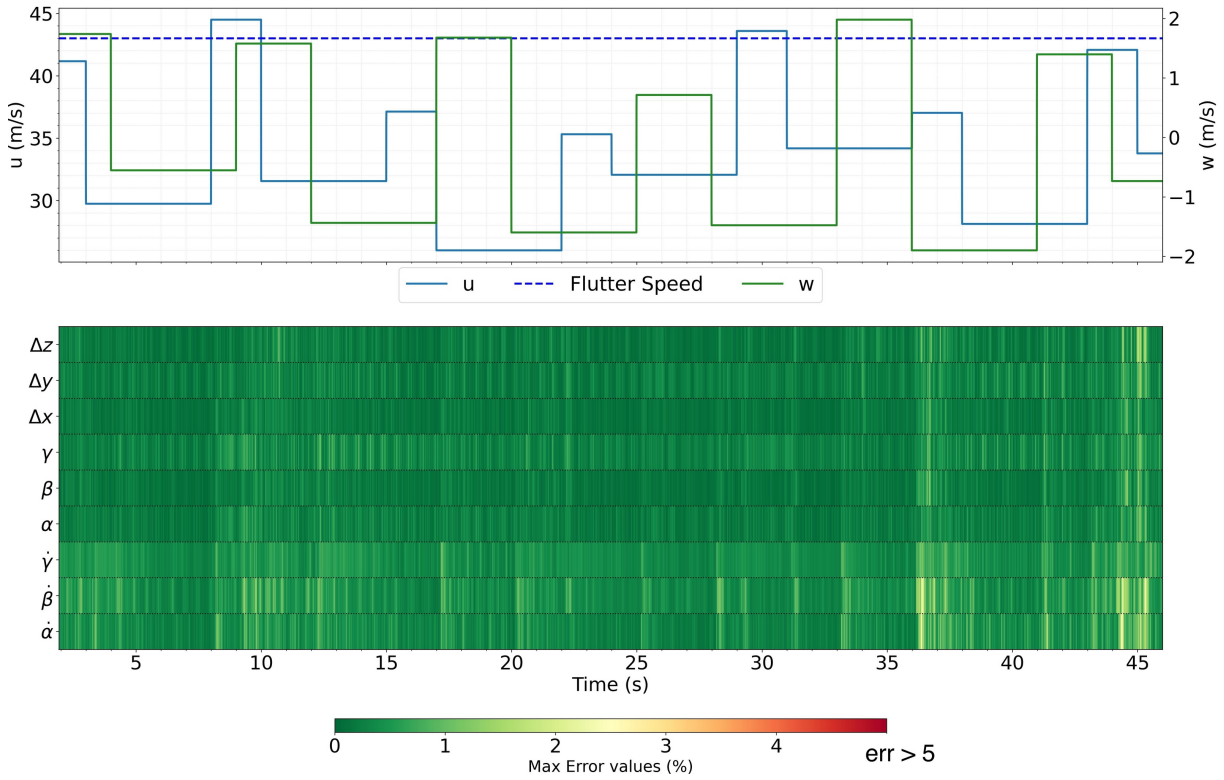


Figure 4.17: Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the train dataset

However, as illustrated in Figure 4.18, at around 10 s, where the dynamical system is excited using a new PRBS (while maintaining the excitation range of the training data: 20 to 50 m/s), the LSTM shows an error of approximately 2% for all structural parameters during testing. This is because the system is close to the flutter onset point and LSTM shows more error when the dynamical system is unstable. Note that the system experiences the same situation around 30 and 37 seconds. However, the LSTM does not show a high error for all structural parameters, as it encounters a lower angle of attack due to lower values in the normal velocity component.

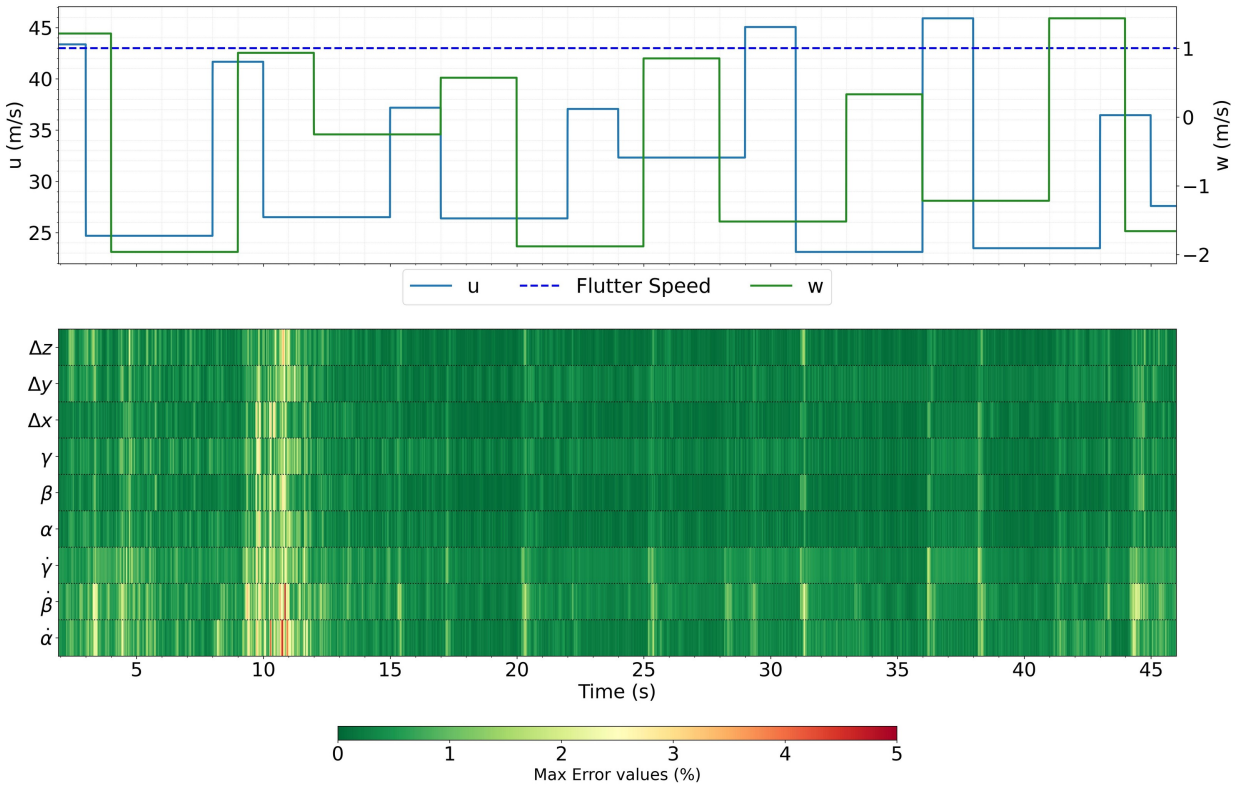


Figure 4.18: Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the test dataset #1

To further test the potential of the LSTM, we studied the time-dependent errors for the system excited by the two different sine-sweep functions. Figure 4.19 indicates that the prediction error exceeds the acceptable threshold, 5%, during certain periods when the airspeed decreases to values lower than the minimum speed in the training dataset (20 m/s). In terms of circulation over the aerodynamic panels of the wing, the LSTM also does not demonstrate good agreement at selected time steps in the testing dataset #2, as shown in Figure 13.

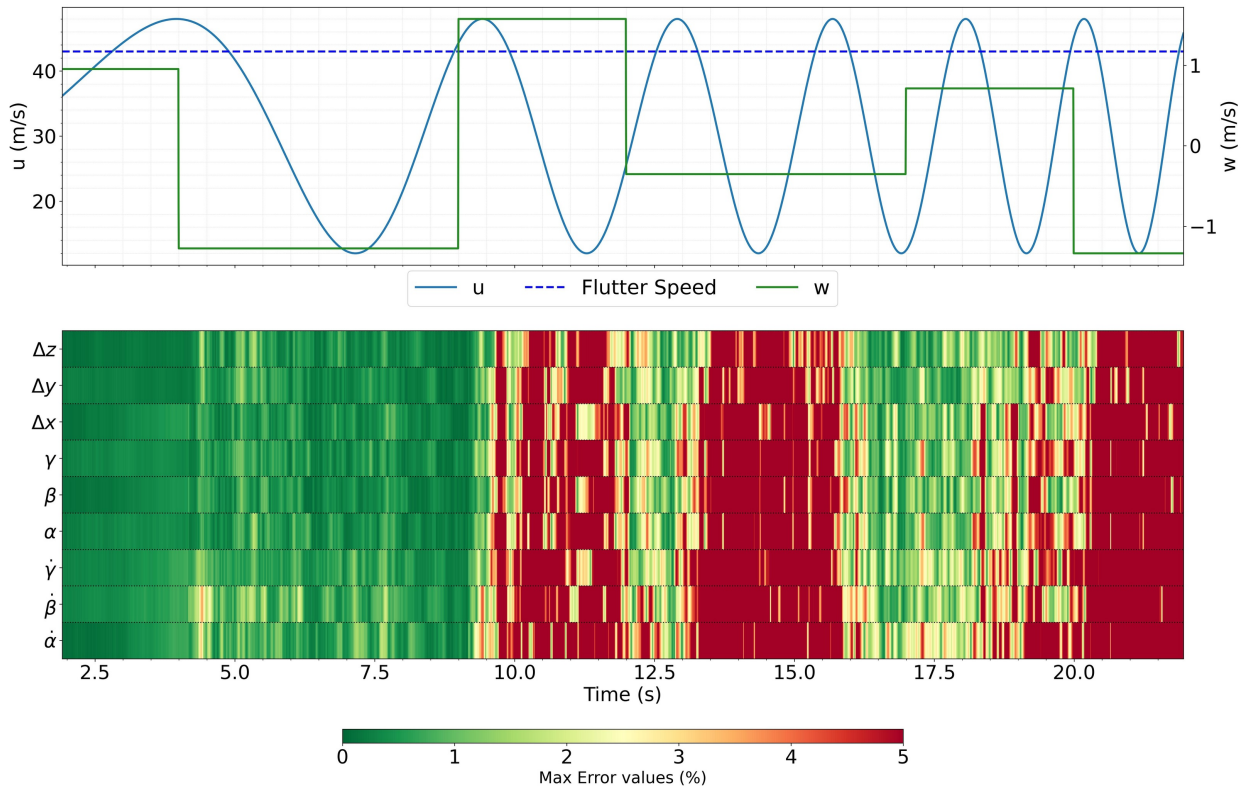


Figure 4.19: Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the test dataset #2

This suggests that the trained LSTM cannot accurately predict the system's state for speeds that fall outside the velocity range during training. However, as shown in Figure 4.20, the LSTM demonstrates good agreement when the other sine sweep excites the dynamical system within the same range of velocities as the training data in general. In this test, when the system exceeds the flutter speed at a positive effective angle of attack—around 8 to 10 s, the LSTM exhibits errors of less than 2%. It only shows errors exceeding 5% for displacement along the x and y axes, and only for a brief period, as indicated in red.

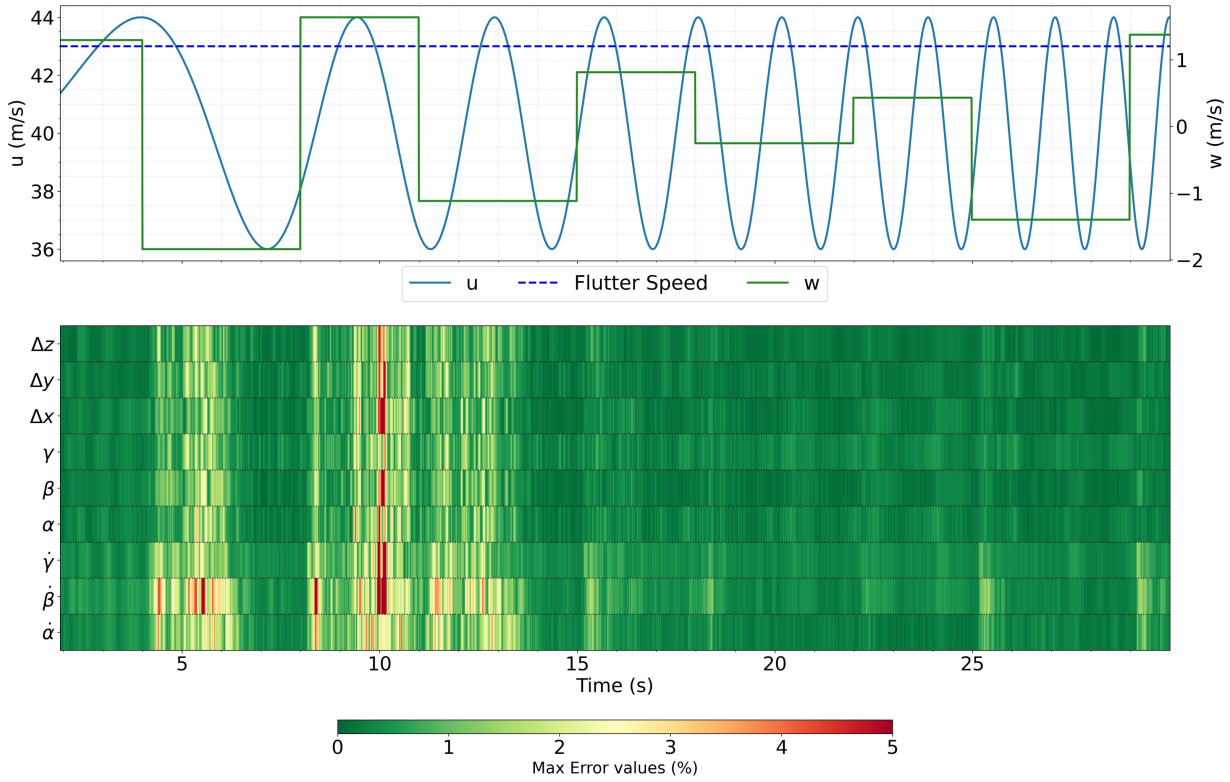
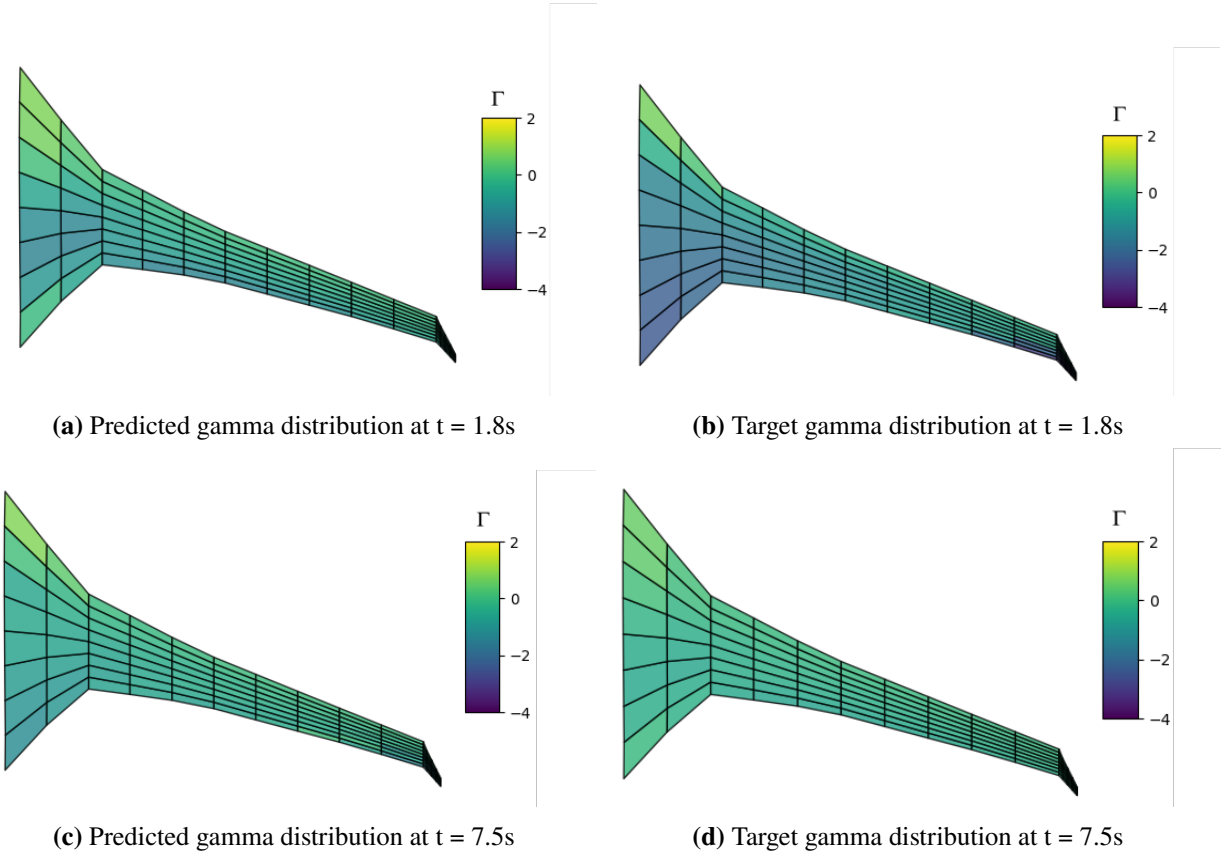


Figure 4.20: Time-domain error analysis of structural variables predicted by LSTM versus SHARPy for the test dataset #3

Generating each dataset in SHARPy, which includes 50 s of training data and 30 s of testing data, takes approximately 5 and 4 hours, respectively. Therefore, predicting a complete horizon (50 timesteps) using SHARPy takes 62 s, whereas the LSTM model, trained as a surrogate predictor, takes under 0.01 s. The time needed for training the LSTM model with 50 epochs was around 30 min in a computer equipped with AMD Ryzen 5 2600X Six-Core processor. This represents a significant computational efficiency improvement, making the LSTM highly suitable for real-time applications like MPC. While SHARPy ensures high accuracy by solving complex physical equations, the LSTM leverages pre-trained data-driven approximations, achieving rapid predictions at a fraction of the computational cost.



(a) Predicted gamma distribution at t = 1.8s

(b) Target gamma distribution at t = 1.8s

(c) Predicted gamma distribution at t = 7.5s

(d) Target gamma distribution at t = 7.5s

Figure 4.21: Comparison of the gamma distribution of aerodynamic panels for test dataset #2 LSTM Prediction vs. SHARPy

MPC is a powerful control strategy that operates by predicting the future behavior of a system over a defined prediction horizon (50 time steps), optimizing a sequence of control inputs (such as control surface deflections), and enforcing system constraints. As mentioned earlier, MPC applies only the first control input in the optimized sequence, then re-solves the optimization at the next time step using updated measurements—a process known as receding horizon control.

When MPC is integrated with a LSTM network, the controller gains a data-driven model capable of capturing nonlinear and time-dependent dynamics. Figure 4.22 to 4.24 show the performance of LSTM processing a 200 time steps of past system states and predicts the future system evolution within the prediction horizon. Notet that some critical moments in the test datasets were chosen to evaluate LSTM prediction. As we expect from the time domain error analysis, LSTM remarkabley understimates the actual displacement, shown by blue line, around t= 11s, shown in Figure 4.23. This error is the maximum within selected horizons shown in black color. As mentioned earlier, these forecasts are then used by MPC to determine the optimal control actions in a rolling framework, continuously updating decisions as new data becomes available. I expect this synergy

makes LSTM-MPC particularly effective for controlling complex dynamical systems such as those encountered in aeroelastic applications in future research.

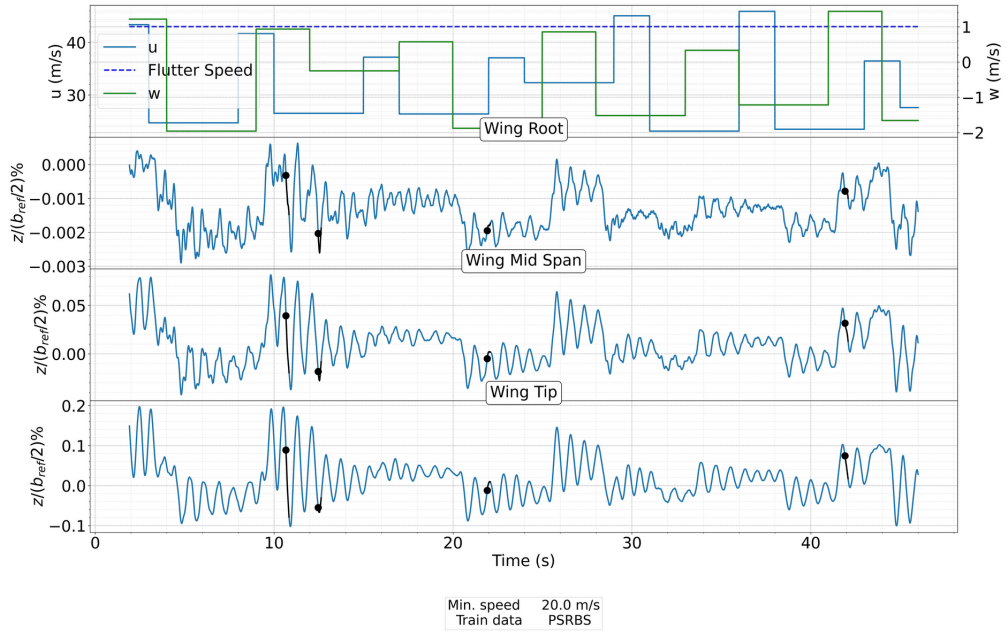


Figure 4.22: LSTM prediction of wingtip vertical displacement at different times within the horizon compared with SHARPy for the test dataset #1

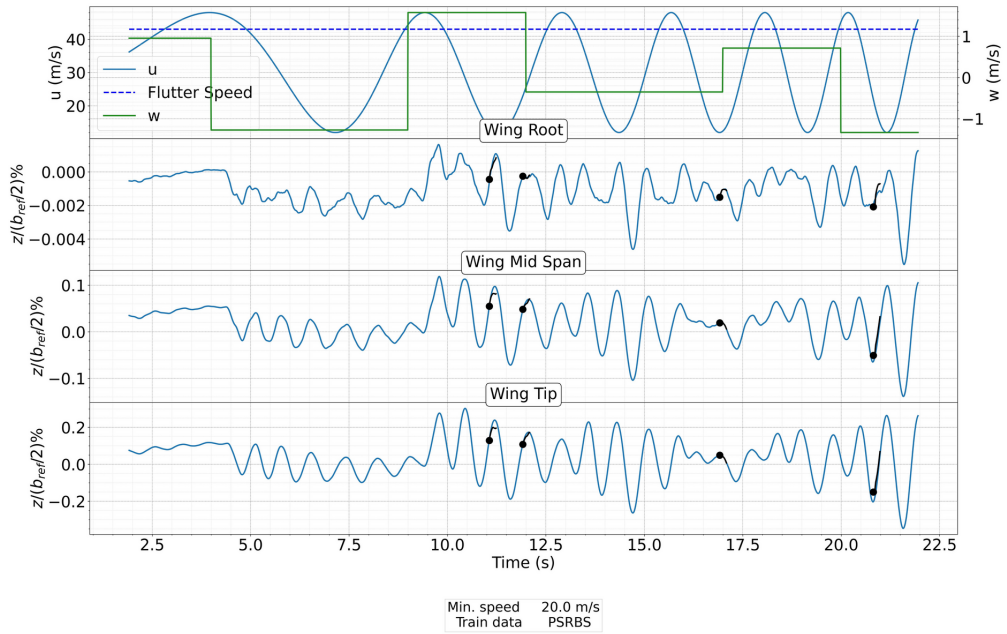


Figure 4.23: LSTM prediction of wingtip vertical displacement at different times within the horizon compared with SHARPy for the test dataset #2

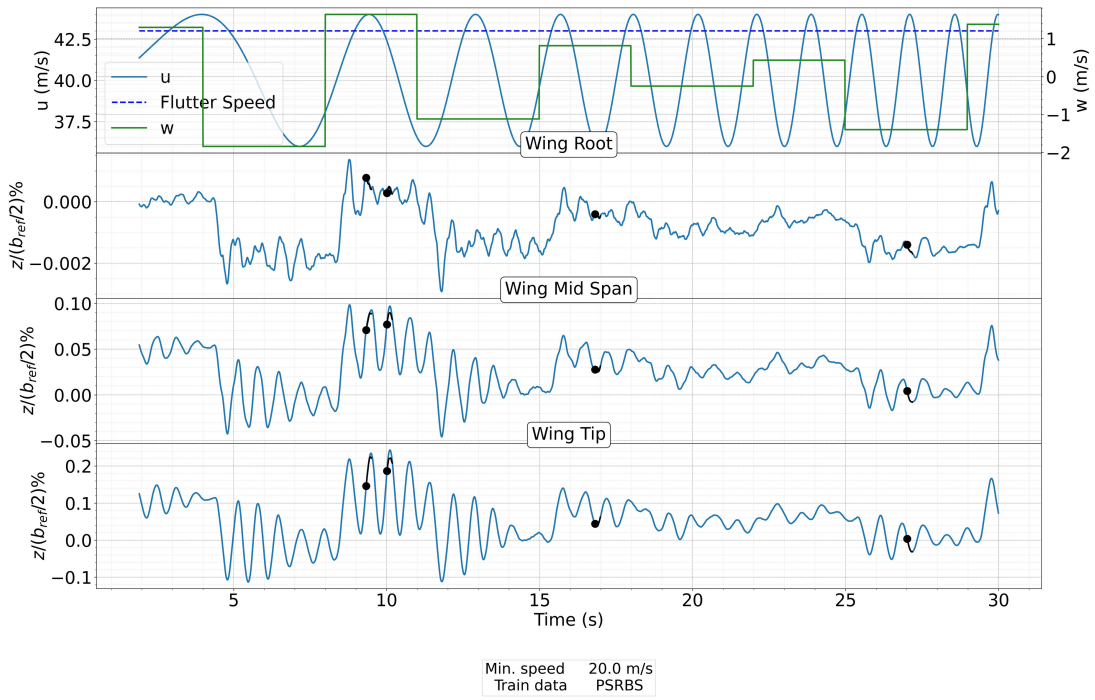


Figure 4.24: LSTM prediction of wingtip vertical displacement at different times within the horizon compared with SHARPy for the test dataset #3

4.3 Conclusion

In this study, we explored advanced data-driven modelling techniques to predict the aeroelastic behavior of a flexible BWB aircraft wing near the flutter onset point using DMDc and an LSTM neural network. Using extensive simulations in SHARPy, we developed an aeroelastic framework that integrates control inputds with transient aeroelastic simulation, generating four distinct datasets for training and testing the data-driven methods. Our findings suggest that LSTM networks are well-suited for real-time prediction and control of aeroelastic instabilities, providing a more efficient and accurate approach compared to DMDc, especially in scenarios where nonlinear aeroelastic behavior plays a critical role.

Although the LSTM model demonstrated good agreement with the SHARPy simulation during validation, when the system was excited with velocities beyond the range of those seen during training, it showed unacceptable prediction error over time as the frequency of the input sine sweep increased. However, when the velocity remained within the range of the training dataset, the LSTM model showed acceptable performance even as the frequency of the sine-sweep function increased. A more accurate data-driven model would require a richer database which, in turn, would increase computational time during the training phase. Furthermore, a longer time sequence might cause the

LSTM to forget old memories. A key recommendation is that implementing LSTM in MPC may require periodic retraining of the model to prevent the loss of correct aerostructural information of the system over long sequences during a real flight.

This study demonstrated that the temporal evolution of structural deformation close to flutter speeds can be accurately modelled using LSTM neural networks, suggesting that this method could be a good candidate to be paired with an MPC controller and used to effectively control an aeroelastic system close the flutter speeds. The integration of machine learning techniques, particularly LSTMs, with aeroelastic solvers like SHARPy is a promising direction for future research regarding the use of these models with MPC controllers. These models can contribute to the design and operation of next-generation flexible aircraft with enhanced performance and stability. Future work should focus on integrating an LSTM model with an MPC controller to maintain aircraft stability beyond the flutter onset point and expand the flight envelope.

Chapter 5

Conclusions

This chapter concludes the thesis by summarizing the key contributions, findings, and implications of the research. A nonlinear aeroelastic simulation framework was developed using SHARPy and validated against experimental data. Two reduced-order modeling techniques—DMDc and LSTM—were trained to predict aeroelastic behavior near flutter conditions. The LSTM model significantly outperformed DMDc in both accuracy and computational efficiency, making it a strong candidate for real-time control applications. The chapter also outlines potential directions for future work, including real-time MPC integration, uncertainty quantification, online learning, and experimental validation.

This thesis presents a novel, data-driven framework for modeling and predicting the nonlinear aeroelastic behavior of flexible aircraft wings near flutter conditions. Motivated by the growing need for efficient real-time prediction and real time control in next-generation aircraft, this study integrates physics-based simulations using SHARPy with data-driven reduced-order models, namely Dynamic Mode Decomposition with control (DMDc) and Long Short-Term Memory (LSTM) neural networks.

The study began by reviewing the limitations of classical reduced-order models, such as linearity assumptions and inability to generalize beyond training regimes. LSTM networks were identified as a promising solution, capable of learning complex temporal dynamics from simulation data. To support this, a comprehensive methodology was developed: (1) SHARPy was used to simulate the aeroelastic response of both a BWB-inspired flexible wing and the benchmark rectangular Pazy wing, capturing essential nonlinear phenomena such as transient oscillations and Limit Cycle Oscillations (LCOs); (2) A data pipeline was established for generating and preprocessing training data using pseudo-random binary sequences and sinusoidal gust inputs; and (3) DMDc and LSTM models were trained on the resulting state histories for forecasting.

5.1 Key Findings

- **Validation of SHARPy**: The nonlinear SHARPy model could correctly follow the trend of key features observed in experimental data from the Pazy wing. It also is capable of predicting flutter onset and LCO amplitude growth, thus establishing SHARPy as a reliable simulator for generating training data for data-driven aeroelastic modeling.
- **Modeling Comparison**: While DMDc was able to capture basic system trends during low-amplitude or linear regimes, its performance deteriorated significantly near flutter where nonlinear effects dominate. In contrast, the LSTM model achieved excellent agreement with the SHARPy "ground truth" predictions, even under strong nonlinear conditions.
- **Performance Metrics**: Across multiple testing datasets, LSTM showed superior predictive accuracy with less than 2% normalized error within the training range, while also generalizing better to previously unseen excitations compared to DMDc. Moreover, LSTM achieved this at a fraction of the computational cost, operating over 6,000 times faster than SHARPy in prediction mode.
- **Suitability for Control Integration**: LSTM's sequential architecture and computational efficiency make it well-suited for integration into MPC frameworks, allowing real-time flutter

suppression strategies that were previously infeasible using full-order or linearized models.

5.2 Future Work

Building on the outcomes of this thesis, the following directions are proposed for further research:

- **Closed-loop control**: Incorporate the trained LSTM model into a real-time MPC loop to actively suppress flutter using feedback from state sensors.
- **Online adaptation**: Develop online learning strategies allowing the LSTM model to adapt in-flight based on new sensor data.
- **Autoencoder-enhanced ROM**: Equip the LSTM model with an autoencoder structure to serve as a nonlinear reduced-order model, enabling more accurate state compression and improved MPC prediction and optimization performance.
- **Experimental validation**: Deploy the trained models on hardware-in-the-loop or wind tunnel test setups to evaluate real-world applicability.

Bibliography

[1] J. R. Wright and J. E. Cooper, *Introduction to Aircraft Aeroelasticity and Loads*. John Wiley & Sons, Ltd., 2014. [Online]. Available: https://doi.org/10.1002/9781118700440

[2] K. L. Roger, G. E. Hodges, and L. Felt, "Active flutter suppression-a flight test demonstration," *Journal of Aircraft*, vol. 12, no. 6, pp. 551–556, 1975. [Online]. Available: https://doi.org/10.2514/3.59833

[3] S. Suzuki, "Simultaneous structure/control design synthesis for aero-servo-elastic system," *Finite Elements in Analysis and Design*, vol. 14, no. 2, pp. 197–208, 1993, special Issue Optimum Design in Japan. [Online]. Available: https://doi.org/10.1016/0168-874X(93)90020-Q

[4] M. Brown and R. Vos, *Conceptual Design and Evaluation of Blended-Wing Body Aircraft*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2018-0522

[5] G. Singh, V. Toropov, and J. Eves, *Topology Optimization of a Blended-Wing-Body Aircraft Structure*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2016-3364

[6] L. Bertsch, F. Wienke, J. Blinstrub, M. Iwanizki, P. Balack, and J. Häßy, "System noise of tube-and-wing and blended-wing–body concept aircraft," *Journal of Aircraft*, vol. 0, no. 0, pp. 1–14, 0. [Online]. Available: https://doi.org/10.2514/1.C037818

[7] E. Livne, "Aircraft active flutter suppression: State of the art and technology maturation needs," *Journal of Aircraft*, vol. 55, no. 1, pp. 410–452, 2018. [Online]. Available: https://doi.org/10.2514/1.C034442

[8] D. H. Hodges and G. A. Pierce, *Introduction to Structural Dynamics and Aeroelasticity*, 2nd ed. Cambridge University Press, 2011.

[9] M. Drela, *Integrated simulation model for preliminary aerodynamic, structural, and control-law design of aircraft*. [Online]. Available: https://doi.org/10.2514/6.1999-1394

[10] M. Karpel, M. Idan, and D. Cohen, *Aeroservoelastic interaction between aircraft structural and control design schemes*. [Online]. Available: https://doi.org/10.2514/6.1998-1864

[11] M. Idan, M. Karpel, and B. Moulin, "Aeroservoelastic interaction between aircraft structural and control design schemes," *Journal of Guidance, Control, and Dynamics*, vol. 22, no. 4, pp. 513–519, 1999. [Online]. Available: https://doi.org/10.2514/2.4427

[12] M. K. Hickner, U. Fasel, A. G. Nair, B. W. Brunton, and S. L. Brunton, "Data-driven unsteady aeroelastic modeling for control," *AIAA Journal*, vol. 61, no. 2, pp. 780–792, 2023. [Online]. Available: https://doi.org/10.2514/1.J061518

[13] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, 1st ed. USA: Cambridge University Press, 2019.

[14] T. D. Economon, F. Palacios, S. R. Copeland, T. W. Lukaczyk, and J. J. Alonso, "Su2: An open-source suite for multiphysics simulation and design," *AIAA Journal*, vol. 54, no. 3, pp. 828–846, 2016. [Online]. Available: https://doi.org/10.2514/1.J053813

[15] N. Fonzi, S. L. Brunton, and U. Fasel, "Data-driven nonlinear aeroelastic models of morphing wings for control," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 476, no. 2239, p. 20200079, 2020. [Online]. Available: https://doi.org/10.1098/rspa.2020.0079

[16] N. Fonzi and S. L. Brunton, "Data-driven modeling for transonic aeroelastic analysis," *Journal of Aircraft*, vol. 61, no. 2, pp. 625–637, 2024. [Online]. Available: https://doi.org/10.2514/1.C037409

[17] M. Ripepi, M. J. Verveld, N. W. Karcher, T. Franz, M. Abu-Zurayk, S. Görtz, and T. M. Kier, "Reduced-order models for aerodynamic applications, loads and mdo," *CEAS Aeronautical Journal*, vol. 9, no. 1, pp. 171–193, Mar. 2018. [Online]. Available: https://doi.org/10.1007/s13272-018-0283-6

[18] J. Lelkes, D. A. Horváth, B. Lendvai, B. Farkas, B. D. Bak, and T. Kalmár-Nagy, "Data-driven aerodynamic models for aeroelastic simulations," *Journal of Sound and Vibration*, vol. 564, p. 117847, 2023. [Online]. Available: https://doi.org/10.1016/j.jsv.2023.117847

[19] K. P. Champion, P. Zheng, A. Y. Aravkin, S. L. Brunton, and J. N. Kutz, "A unified sparse optimization framework to learn parsimonious physics-informed models from data," *CoRR*, vol. abs/1906.10612, 2019. [Online]. Available: http://arxiv.org/abs/1906.10612

[20] B. Zhang, J. Han, H. Yun, and X. Chen, "Nonlinear aeroelastic system identification based on neural network," *Applied Sciences*, vol. 8, no. 10, 2018. [Online]. Available: https://www.mdpi.com/2076-3417/8/10/1916

[21] G. Borrelli, L. Guastoni, H. Eivazi, P. Schlatter, and R. Vinuesa, "Predicting the temporal dynamics of turbulent channels through deep learning," *International Journal of Heat and Fluid Flow*, vol. 96, p. 109010, 2022. [Online]. Available: https://doi.org/10.1016/j.ijheatfluidflow.2022.109010

[22] H. Eivazi, L. Guastoni, P. Schlatter, H. Azizpour, and R. Vinuesa, "Recurrent neural networks and koopman-based frameworks for temporal predictions in a low-order model of turbulence," *International Journal of Heat and Fluid Flow*, vol. 90, p. 108816, 2021. [Online]. Available: https://doi.org/10.1016/j.ijheatfluidflow.2021.108816

[23] J. Bakarji, K. Champion, J. Nathan Kutz, and S. L. Brunton, "Discovering governing equations from partial measurements with deep delay autoencoders," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 479, no. 2276, p. 20230422, 2023. [Online]. Available: https://royalsocietypublishing.org/doi/abs/10.1098/rspa.2023.0422

[24] Q. Guo, X. Li, Z. Zhou, D. Ma, and Y. Wang, "Neural network-based aeroelastic system identification for predicting flutter of high flexibility wings," *Scientific Reports*, vol. 15, no. 1, p. 623, 2025. [Online]. Available: https://doi.org/10.1038/s41598-024-82573-7

[25] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735

[26] Y. Lee, C. Park, N. Kim, J. Ahn, and J. Jeong, "Lstm-autoencoder based anomaly detection using vibration data of wind turbines," *Sensors*, vol. 24, no. 9, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/9/2833

[27] R. Zahn, M. Winter, M. Zieher, and C. Breitsamter, "Application of a long short-term memory neural network for modeling transonic buffet aerodynamics," *Aerospace*

Science and Technology, vol. 113, p. 106652, 2021. [Online]. Available: https://doi.org/10.1016/j.ast.2021.106652

[28] W. Li, X. Gao, and H. Liu, "Efficient prediction of transonic flutter boundaries for varying mach number and angle of attack via lstm network," *Aerospace Science and Technology*, vol. 110, p. 106451, 2021. [Online]. Available: https://doi.org/10.1016/j.ast.2020.106451

[29] A. del Carre, A. Muñoz-Simón, N. Goizueta, and R. Palacios, "SHARPy: A dynamic aeroelastic simulation toolbox for very flexible aircraft and wind turbines," *Journal of Open Source Software*, vol. 4, no. 44, p. 1885, Dec. 2019. [Online]. Available: https://doi.org/10.21105/joss.01885

[30] A. D. Carre and R. Palacios, *Efficient Time-Domain Simulations in Nonlinear Aeroelasticity*. [Online]. Available: https://doi.org/10.2514/6.2019-2038

[31] O. Avin, D. E. Raveh, A. Drachinsky, Y. Ben-Shmuel, and M. Tur, "Experimental aeroelastic benchmark of a very flexible wing," *AIAA Journal*, vol. 60, no. 3, pp. 1745–1768, 2022. [Online]. Available: https://doi.org/10.2514/1.J060621

[32] A. Drachinsky, O. Avin, D. E. Raveh, Y. Ben-Shmuel, and M. Tur, "Flutter tests of the pazy wing," *AIAA Journal*, vol. 60, no. 9, pp. 5414–5421, 2022. [Online]. Available: https://doi.org/10.2514/1.J061717

[33] M. Ritter, J. Hilger, A. Ribeiro, A. E. Öngüt, M. Righi, D. E. Raveh, A. Drachinsky, C. Riso, C. E. Cesnik, B. Stanford, P. Chwalowski, R. K. Kovvali, S. Duessler, K. C. Cheng, R. Palacios, J. P. dos Santos, F. D. Marques, G. R. R. Begnini, A. A. Verri, J. J. Lima, F. B. de Melo, and F. L. Bussamra, *Collaborative Pazy Wing Analyses for the Third Aeroelastic Prediction Workshop*. [Online]. Available: https://doi.org/10.2514/6.2024-0419

[34] G. Pagliuca and M. Forster, *Aeroelastic Solutions of the AGARD 445.6 Wing using Linear- and Nonlinear-CFD*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2022-1326

[35] B. M. Lowe and D. W. Zingg, "Flutter prediction using reduced-order modeling with error estimation," *AIAA Journal*, vol. 60, no. 7, pp. 4240–4255, 2022. [Online]. Available: https://doi.org/10.2514/1.J061389

[36] E. Kaiser, J. N. Kutz, and S. L. Brunton, "Sparse identification of nonlinear dynamics for model predictive control in the low-data limit," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 474, no. 2219, p. 20180335, 2018. [Online]. Available: https://doi.org/10.1098/rspa.2018.0335

[37] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, and B. Açıkmeşe, "Model predictive control in aerospace systems: Current state and opportunities," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 7, pp. 1541–1566, 2017. [Online]. Available: https://doi.org/10.2514/1.G002507

[38] A. I. da Silveira, F. França, T. S. Morais, and P. Assis, *Dual Mode Predictive Control for Active Flutter Suppression on a Two-Dimensional Aerofoil in the Presence of Input Constraints*. [Online]. Available: https://doi.org/10.2514/6.2024-4502

[39] S. Maraniello and R. Palacios, "State-space realizations and internal balancing in potential-flow aerodynamics with arbitrary kinematics," *AIAA Journal*, vol. 57, no. 6, pp. 2308–2321, 2019. [Online]. Available: https://doi.org/10.2514/1.J058153

[40] D. Li, S. Zhao, A. Da Ronch, J. Xiang, J. Drofelnik, Y. Li, L. Zhang, Y. Wu, M. Kintscher, H. P. Monner, A. Rudenko, S. Guo, W. Yin, J. Kirn, S. Storm, and R. D. Breuker, "A review of modelling and analysis of morphing wings," *Progress in Aerospace Sciences*, vol. 100, pp. 46–62, 2018. [Online]. Available: https://doi.org/10.1016/j.paerosci.2018.06.002

[41] N. Goizueta, A. Wynn, and R. Palacios, *Parametric Krylov-based order reduction of aircraft aeroelastic models*. [Online]. Available: https://arc.aiaa.org/doi/abs/10.2514/6.2021-1798

[42] N. Goizueta, A. Wynn, R. Palacios, A. Drachinsky, and D. E. Raveh, "Flutter predictions for very flexible wing wind tunnel test," *Journal of Aircraft*, vol. 59, no. 4, pp. 1082–1097, 2022. [Online]. Available: https://doi.org/10.2514/1.C036710

[43] H. Hesse, J. Murua, and R. Palacios, *Consistent Structural Linearization in Flexible Aircraft Dynamics with Large Rigid-Body Motion*. [Online]. Available: https://doi.org/10.2514/6.2012-1402

[44] M. Karbasi, M. Ali, S. M. Bateni, C. Jun, M. Jamei, A. A. Farooque, and Z. M. Yaseen, "Multi-step ahead forecasting of electrical conductivity in rivers by using a hybrid convolutional neural network-long short-term memory (cnn-lstm) model enhanced by boruta-xgboost feature selection algorithm," *Scientific Reports*, vol. 14, no. 1, p. 15051, 2024. [Online]. Available: https://doi.org/10.1038/s41598-024-65837-0

[45] A. Pereira, S. Warwick, A. Moutinho, and A. Suleman, "Infrared and visible camera integration for detection and tracking of small uavs: Systematic evaluation," *Drones*, vol. 8, no. 11, p. 650, 2024. [Online]. Available: https://doi.org/10.3390/drones8110650

[46] A. Zeinalzadeh and M. P. and, "Evaluation of novel-objective functions in the design optimization of a transonic rotor by using deep learning," *Engineering Applications of*

Computational Fluid Mechanics, vol. 15, no. 1, pp. 561–583, 2021. [Online]. Available: https://doi.org/10.1080/19942060.2021.1895889

[47] A. Zeinalzadeh, G. H. Kamakoli, and M. Pakatchian, "A hybrid data-driven framework for loss prediction of mca airfoils," *Engineering Analysis with Boundary Elements*, vol. 163, pp. 394–405, 2024. [Online]. Available: https://doi.org/10.1016/j.enganabound.2024.03.008

Appendix A

Additional Information

A.1 Step-by-step DMDc Implementation with Python

```
import numpy as np

# === STEP 1: Collect Snapshot Matrices ===
# StateData: (n x m) matrix of state vectors
# InputData: (q x m) matrix of input vectors

X = StateData[:, :-1]       # shape (n, m-1)
Xp = StateData[:, 1:]       # shape (n, m-1)
Ups = InputData[:, :-1]     # shape (q, m-1)

# Stack X and Ups vertically to form Omega
Omega = np.vstack((X, Ups))  # shape (n+q, m-1)

# === STEP 2: SVD of Input Matrix Omega ===
U, Sig, Vh = np.linalg.svd(Omega, full_matrices=False)
thresh = 1e-10
```

```
rtil = np.sum(Sig > thresh)

Util = U[:, :rtil]
Sigtil = np.diag(Sig[:rtil])
Vtil = Vh[:rtil, :]

# === STEP 3: SVD of Output Matrix Xp ===
U2, Sig2, Vh2 = np.linalg.svd(Xp, full_matrices=False)
r = np.sum(Sig2 > thresh)

Uhat = U2[:, :r]
Sighat = np.diag(Sig2[:r])
Vbar = Vh2[:r, :]

# === STEP 4: Compute Approximations of A and B ===
n = X.shape[0]
q = Ups.shape[0]

U_1 = Util[:n, :]       # first n rows of Util
U_2 = Util[n:n+q, :]    # next q rows of Util

# Approximate operators A and B
approxA = Uhat.T @ Xp @ Vtil.T @ np.linalg.inv(Sigtil) @ U_1.T @ Uhat
approxB = Uhat.T @ Xp @ Vtil.T @ np.linalg.inv(Sigtil) @ U_2.T

# === STEP 5: Eigenvalue Decomposition of A ===
eigvals, W = np.linalg.eig(approxA)

# === STEP 6: Compute DMD Modes ===
Phi = Xp @ Vtil.T @ np.linalg.inv(Sigtil) @ U_1.T @ Uhat @ W

# (Optional) Output eigenvalues and modes
print("Eigenvalues:\n", eigvals)
print("DMD Modes (Phi):\n", Phi)
```

Listing A.1: Python code for DMDc implementation

A.2 Pseudo code for updating aerodynamic properties

```
1. Assert prerequisites:
- If the object does not have attribute `conn_glob`, raise an error:
"Run `update_derived_params` before generating files".
```

```
2. Initialize variables:
- Retrieve parameters:
$n\_surfaces$, $num\_node\_surf$, $num\_elem\_tot$, $pct\_flap$, $control\
 _surface$.

3. Generate aerofoil profiles:
- Initialize an empty list: $Airfoils\_surf$.

Case 1: Multiple surfaces ($n\_surfaces == 2$)
- For each node $inode$ in surface 0:
- Compute $\eta = inode / num\_node\_surf$ (spanwise location).
- Interpolate NACA camber line using geometric utilities and append to
 $Airfoils\_surf$.
- Set flap controls:
- Initialize $ws\_elem = 0$.
- For each surface $i\_surf$:
- For each element $i\_elem$ in the surface:
- For each local node $i\_local\_node$:
- If $i\_elem \geq int(num\_elem\_surf \cdot (1 - pct\_flap))$:
- Set control values:
- Surface 0: Right flap ($control\_surface[ws\_elem + i\_elem, i\_local\
 _node] = 0$).
- Surface 1: Left flap ($control\_surface[ws\_elem + i\_elem, i\_local\_node
 ] = 1$).
- Increment $ws\_elem$ by $num\_elem\_surf$.
- Adjust control surfaces for symmetry:
- Combine surface connections.
- Reverse and reorder elements.

Case 2: Single surface ($n\_surfaces == 1$)
- Compute $num\_node\_half = (num\_node\_surf + 1) / 2$.
- For each node $inode$ in the half-surface:
- Compute $\eta = inode / num\_node\_half$.
- Interpolate NACA camber line and append to $Airfoils\_surf$.
- Create airfoil distribution:
- Extract half-surface connections.
- Concatenate flipped and original connections.

4. Store aerofoil profiles:
- Assign $Airfoils\_surf$ and $airfoil\_distribution$ to object attributes.

5. Update additional aerodynamic properties:
```

```
- Set $aero\_node$ to mark all nodes as aerodynamic.
- Set surface discretization properties like $surface\_m$.
- Initialize arrays for $twist$, $chord$, and $elastic\_axis$.
- Update the $control\_surface$ matrix.
```

A.3 Pseudo code for solver setup and post-processing

```
1. Configure the structural solver for `DynamicCoupled`:
- Set `structural_solver` to `NonLinearDynamicPrescribedStep`.

2. Configure settings for `NonLinearDynamicPrescribedStep`:
- Disable gravity: `gravity_on = True`.

3. Link `structural_solver_settings` of `DynamicCoupled` to settings from `
 NonLinearDynamicPrescribedStep`.

4. Configure the aerodynamic solver for `DynamicCoupled`:
- Set `aero_solver` to `StepUvlm`.
- Link `aero_solver_settings` of `DynamicCoupled` to settings from `StepUvlm
 `.

5. Define postprocessors for `DynamicCoupled`:
- Set `postprocessors` to:
- `WriteVariablesTime`
- `BeamLoads`
- `BeamPlot`
- `AerogridPlot`.

6. Configure settings for postprocessors:
- `BeamLoads`:
- `csv_output = off`
- `BeamPlot`:
- `include_rbm = on`
- `include_applied_forces = on`
- `AerogridPlot`:
- `include_rbm = on`
- `include_applied_forces = on`
- `minus_m_star = 0`
- `WriteVariablesTime`:
- `structure_variables = ['pos', 'psi', 'psi_dot']`
- `aero_panels_variables = ['gamma']`
- `aero_nodes_variables = ['dynamic_forces']`
```

```
- `structure_nodes = np.arange(0, wing.StructuralInformation.num_node)`
```

A.4 Python code for transient simulation solver setup

```
flow = [
'BeamLoader',
'AerogridLoader',
'StaticCoupled',
'AerogridPlot',
'BeamPlot',
'DynamicCoupled',
'LiftDistribution'
]
```