TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS

IMPLEMENTATION OF DIGITAL TWIN TECHNOLOGY FOR REAL-TIME MONITORING OF AN AIRCRAFT WING MODEL

By:
Sangam Bahadur Adhikari (077BAS038)
Saphal Gandhari (077BAS039)
Suyog Basnet (077BAS045)
Kushal Neupane (077BAS048)

A FINAL REPORT SUBMITTED TO THE DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE BACHELOR'S DEGREE IN AEROSPACE ENGINEERING

DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING
LALITPUR, NEPAL

March, 2025

COPYRIGHT

TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS
DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

LETTER OF APPROVAL

The undersigned certify that they have read, and recommended to the Institute of Engineering for acceptance, a project report entitled **'IMPLEMENTATION OF DIGITAL TWIN TECHNOLOGY FOR REAL-TIME MONITORING OF AN AIRCRAFT WING MODEL'**, submitted by Sangam Bahadur Adhikari, Saphal Gandhari, Suyog Basnet and Kushal Neupane in partial fulfillment of the requirements for the Bachelor's Degree in Mechanical and Aerospace Engineering.

Supervisor: Prof. Rajendra Shrestha, PhD
Department of Mechanical and Aerospace Engineering
Institute of Engineering, Pulchowk Campus

Supervisor: Arun Bikram Thapa, Assistant Professor
Department of Mechanical and Aerospace Engineering
Institute of Engineering, Pulchowk Campus

External Examiner: Ashish Manandhar
MPC Engineer, Himalaya Airlines

Head of Department: Dr. Sudip Bhattrai, PhD, Assistant Professor
Department of Mechanical and Aerospace
Institute of Engineering, Pulchowk Campus

DATE OF APPROVAL: April , 2025

ACKNOWLEDGEMENT

This project is prepared in partial fulfillment of the requirement for the bachelor's degree in Mechanical and Aerospace Engineering. First and foremost, we express our sincere gratitude towards Prof. Dr. Rajendra Shrestha and Asst. Prof. Arun Bikram Thapa, our project supervisor, for their constant guidance, inspiring lectures, and precious encouragement. We would also like to express our gratitude to Er. Ashish Manandhar for his valuable insights and guidance. Without their invaluable supervision and suggestions, it would have been a difficult journey for us. Their useful suggestions for this whole work and cooperative behavior are sincerely acknowledged. We also thank Mr. Abhishek Bhandari for facilitating us for the project.

We would like to thank Head of Department, Dr. Sudip Bhattarai, Deputy Head of Department, Asst. Prof. Kamal Darlami, Asst. Prof. and Project Coordinator Biman Rimal and the entire Department of Mechanical and Aerospace Engineering, Institute of Engineering, Pulchowk Campus, to provide us the opportunity for a collaborative undertaking which has helped us to implement the knowledge gained over these years as a major project for the fourth year, that has greatly enhanced our knowledge and provided us with a new experience of teamwork.

We are grateful to Incubation, Innovation and Entrepreneurship Center (IIEC) for providing the resources. Our gratitude extends to our friends, seniors, and juniors who have been part of this journey and contributed to the achievement of our project goals.

Any kind of suggestion or criticism will be highly appreciated and acknowledged.

Authors:
Sangam Bahadur Adhikari (077BAS038)
Saphal Gandhari (077BAS039)
Suyog Basnet (077BAS045)
Kushal Neupane (077BAS048)

ABSTRACT

Digital twinning is one of the active fields aimed at real-time structural health monitoring of the aircraft wings. The aviation industry's drive for improved safety, greater efficiency, and reduced maintenance costs highlights the need for advanced monitoring systems. The lack of research and work on real-time monitoring using the digital twin seems persistent. The project aims to reduce the gap and provide a gateway to achieving structural health monitoring using digital twins. Our work proposes cutting-edge technology of a digital twin framework that combines physical models, sensors, cloud server, and real-time data connection. With the help of data from the strain gauge sensors placed on the wing, we conducted real-time monitoring of the structural behavior (deflection) of the wing under different loading conditions. The strain gauges were properly calibrated using controlled point loads to ensure the data they provide is accurate and reliable for further visualization. The project will open reliable and proper gateway for data transmission using cloud, identifying changes in strain patterns through deflection, and real-time visualization of the wing; achieving digital twin. This approach taken for digital twin, provides a real-time monitoring of the wing which can be further enhanced for continuous health assessment improving the accuracy and efficiency of aircraft wing maintenance. The project introduces a novel technique that seems to be lacking in the field of real-time monitoring, which will help revolutionize aircraft maintenance, ensuring higher standards of safety and operational efficiency.

Keywords: Cloud API Server, Digital Twinning, Structural Health Monitoring, Data Integration, Internet of Things(IoT), Real-Time Analysis

TABLE OF CONTENTS

List of Figures

List of Tables

LIST OF ABBREVIATIONS

ACF	Auto-correlation Function
ADC	Analog to Digital Converter
AI	Artificial Intelligence
AoA	Angle of Attack
API	Application Programming Interface
AWS	Amazon Web Service
CAD	Computer Aided Design
CATIA	Computer Aided Three-Dimensional Interactive Application
DBNs	Dynamic Bayesian Networks
DDT	Deep Digital Twin
DT	Digital Twin
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
LoRa	Long Range
MATLAB	MATrix LABoratory
MQTT	Message Queuing Telemetry Transport
M2M	Machine-To-Machine
NASA	National Aeronautics and Space Administration
PCM	Product Cycle Management
PF	Particle Filters
PLA	Polylatic Acid
PLM	Product Life Management
REST API	Representational State Transfer Application Programming Interface
SHM	Structural Health Monitoring
SSS	Stress- Strain State
UDL	Uniformally Distributed Load
Wi-Fi	Wireless Fidelity
WSN	Wireless Sensor Network
3D	Three-dimensional
5G	5th Generation

LIST OF SYMBOLS

ρ	Air density (kg/m^3)
v	Velocity of air relative to the wing (m/s)
A	Reference area (m^2)
C_l	Lift coefficient
E	Young's modulus (Pa)
b	Wing chord length (m)
h	Thickness of the wing (m)
I	Moment of inertia (m^4)
$\delta_{\max}$	Maximum deflection of the wing (mm)
q	Uniformly distributed load (N/m)
l	Wing span (m)
L	Lift force (N)

1. INTRODUCTION

1.1. Background

Aircraft structures, especially wings, face a variety of dynamic loads and environmental stresses throughout their operational life. Maintaining the structural integrity of these components is essential for ensuring flight safety, achieving operational efficiency, and managing costs effectively. Traditionally, Structural Health Monitoring (SHM) of aircraft has depended on manual inspections and scheduled maintenance routines. However, these conventional methods often struggle to detect issues as they occur, sometimes resulting in unexpected failures, higher maintenance expenses, and increased downtime [1].

Digital Twin (DT) technology has emerged as a promising solution to overcome these limitations. Simply put, a Digital Twin is a virtual model that mirrors a physical system by integrating real-time sensor data, computational simulations, and advanced analytics. This digital representation provides engineers with the ability to continuously monitor, diagnose, and predict the behavior of the actual system, thereby enabling proactive maintenance and improved safety. Through real-time diagnostics and predictive capabilities, Digital Twins offer a comprehensive approach to SHM that can reduce costs and prevent failures [2].

The concept of the Digital Twin was first introduced by Michael Grieves in 2002 as part of Product Lifecycle Management (PLM). Since then, rapid advancements in technologies such as the Internet of Things (IoT), cloud computing, and Artificial Intelligence (AI) have expanded the scope of Digital Twin applications across various sectors, including aerospace [3]. A notable historical example is NASA's Apollo 13 mission in 1970, where a physical replica of the spacecraft was used to simulate and resolve a critical issue—a practice that laid the groundwork for the Digital Twin concept [2].

In aerospace, the use of Digital Twin technology is steadily increasing because it offers numerous advantages. By developing a virtual model that is an exact replica of an aircraft wing, engineers can monitor its structural behavior in real time. This continuous observation allows for the early detection of anomalies and the prediction of potential failures before they occur. As a result, the proactive maintenance approach enabled by Digital Twins minimizes operational downtime, enhances overall safety, and extends the service life of aircraft components [4].

A Digital Twin is essentially a digital copy of a physical product whether one that currently exists or one that is planned for production. NASA defines a Digital Twin as "an integrated multi-physics, multi-scale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin" [2]. Additionally, Liu et al. describe Digital Twins as "living models of physical assets that continuously adapt to operational changes based on collected online data and can forecast the future state of the corresponding physical counterpart" [4]. Although concepts similar to Digital Twins have been in use since the 1960s, it was Grieves' formal introduction in 2002 that spurred its development in modern engineering.

Digital Twin technology stands out because it combines three essential components: the real-world physical product, its digital counterpart, and the data that connects them. This integrated approach enables not only real-time monitoring and diagnostic capabilities but also supports virtual simulations that help identify potential issues before they escalate. Unlike traditional SHM methods, which rely on periodic data and manual inspections, Digital Twins continuously update based on live data, thereby offering a more accurate and timely picture of a system's condition [3].

Moreover, traditional SHM techniques, which depend on fixed maintenance schedules, often fail to account for actual wear and tear, leading to inefficient resource use and significant losses. In contrast, Digital Twin technology harnesses live data to create a dynamic visual model of the system. This model reflects real-time changes and conditions, making it possible to spot abnormalities promptly and accurately. Consequently, this proactive method not only improves safety but also reduces unnecessary maintenance costs by addressing issues before they develop into serious problem[3].

In today's increasingly digitized world, there is a growing demand for more advanced methods to ensure safety and reliability. The complexity of modern systems further underscores the need for innovative solutions like Digital Twin technology. In aerospace engineering, the goal is to expand the use of Digital Twins for effective structural health monitoring of aircraft wings. This is achieved by fabricating a physical model of the wing and analyzing it using various software tools to validate and enhance the digital representation [5]. By integrating real-time sensor data with sophisticated computational models, Digital Twin technology offers a transformative approach to monitoring and maintaining aircraft, ensuring that they operate safely and efficiently over their entire lifespan.

1.2. Problem Statement

Aircraft wings are vital for safe flight, and understanding their real-time behavior is crucial for both design optimization and proactive maintenance. Traditional visualization methods are static and resource-intensive, often failing to integrate live sensor data that could provide a dynamic view of the wing's performance.

This project addresses the gap by developing a Digital Twin system that provides real-time visualization of an aircraft wing while opening gates to integrate advanced SHM techniques and analysis. The project will create a virtual model that mirrors the physical wing, integrating live sensor data to simulate deflection behavior. This approach not only enhances the design and simulation process but also lays the groundwork for early fault detection and maintenance planning, thereby supporting both design innovation and proactive SHM strategies.

1.3. Objectives

1.3.1. Main Objective

To implement and validate the Digital Twin Technology for an aircraft wing model.

1.3.2. Specific Objectives

1. To integrate the sensors on the wing model and calibrate the sensor data.
2. To integrate Cloud API by sending sensor data through microcontroller for real-time data communication.
3. To develop a digital twin of the physical wing model.
4. To validate and monitor the real-time data from the physical model.

1.4. Applications

The implementation of Digital Twin technology in aircraft wing real-time monitoring offers several applications, including:

1. Real-time data monitoring to detect structural anomalies.

2. Predictive maintenance to prevent unexpected failures.
3. Enhanced safety and reliability through continuous monitoring.
4. Improved lifecycle management of aircraft components.
5. Virtual characterization and testing of structural performance.
6. Interactive training environments for engineers and maintenance personnel.

1.5. Feasibility Analysis

1.5.1. Economic Feasibility

The project is economically feasible because the creation of the Digital Twin provides a cost-effective way to virtually monitor the wing's behavior under different conditions. By simulating and visualizing real-time wing deformation, future design improvements can be made without expensive physical testing. This reduces material waste and speeds up the development process, saving both time and money.

1.5.2. Technical Feasibility

The availability of the sensors and microcontroller for the data collection, the integration of the digital twin, and the availability of the CATIA and 3D print machine for the fabrication of the wing model makes it feasible. Also, the availability of the software like Unity along with Cloud API platforms makes the development of digital twin technically feasible.

1.5.3. Operational Feasibility

The operational feasibility of this project is high, as the Digital Twin system can effectively capture and visualize real-time wing deformation using affordable sensors and microcontrollers. The system integrates well with existing tools like MATLAB, Arduino IDE, and Unity, ensuring smooth data processing and visualization.

1.6. System Requirements

1.6.1. Hardware Requirements

1. ESP32 8266 Wroom microcontroller
2. Strain gauge BF350388
3. Arduino UNO
4. Rechargeable battery
5. Strain gauge Y3 module amplifier
6. HX711 amplifier
7. Jumper wires
8. High speed camera
9. Beam load machine

1.6.2. Software Requirements

1. CAD software: CATIA V5
2. Digital Twin development: Unity 3D
3. Cloud-based data acquisition: AWS Cloud
4. Programming environment: Arduino IDE
5. Image processing and data filtering: MATLAB

2. LITERATURE REVIEW

2.1. Introduction

The vision of the Digital Twin itself refers to a comprehensive physical and functional description of a component, product or system together with all available operational data[6]. It is based on the idea that a digital informational construct about a physical system could be created as an entity on its own. This digital information would be a "twin" of the information that was embedded within the physical system itself and would be linked with that physical system through the entire lifecycle of the system[7]. Data is at the heart of Digital Twin. The physical operation process is judged, analyzed, predicted and optimized in virtual means. After, the simulation and optimization of product design, manufacturing and maintenance process, it guides the physical process to perform the optimized solution[8]. Digital Twin in aerospace has been driven by four key technologies; IoT and Big Data, Advanced Analytics, Computing Power, and Accessibility[3]. DT should not be seen as a single technology but as a set of devices, communication tools, and software working together, and most importantly, autonomously[9]. Such autonomy implies Machine-to-Machine (M2M) communication without any direct human intervention, effectively achieved by the IoT. In this context, the physical world is perceived through a sensor network spatially distributed around the monitored structure, which means that a large amount of data is collected. At the same time, the devices can operate on the physical world through actuators when instructed[5].

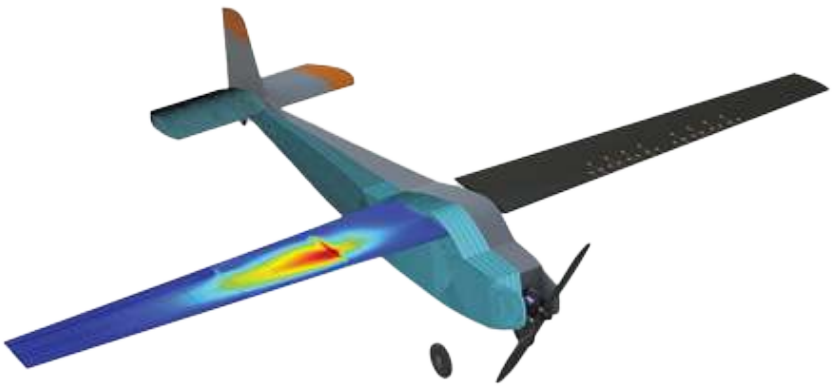

Figure 2.1: *Digital Twin of Aircraft Model*

2.2. History

The digital twin was first introduced by Grieves in 2002 as a concept within Product Life Cycle Management (PCM); similar approaches have been used since the 1960s[3]. The digital twin approach was used in missions like Apollo 13. During the mission in 1970, the oxygen tanks exploded after the launch. The problem was extremely difficult to solve as it occurred 200,000 miles away. However, NASA had the Twin of Apollo 13 on the ground, almost a physical mockup. The intervention to rescue the crew had been tested first on the physically copied model. The mission was accomplished, the crew rescued and that was the first known big success of the Twin concept[10]. With the advent of Industry 4.0 and the advances made in Big Data analytics and Machine and Deep Learning, the Digital Twin has been spreading to several industrial fields, and in 2019 75% of industrial world organizations are planning to use its innovative technologies in their business processes[11]. Nowadays, the DT continues to show an increasing trend and its market adoption is expected to grow heavily in several years. It is estimated that by 2027 almost all the IoT platforms will contain digital twinning capabilities[3].

2.3. Digital Twin for Aircraft

The development of validated multidisciplinary Structural Health Management (SHM) system tools, technologies, and techniques to enable detection, diagnosis, and prognosis in the presence of adverse conditions during flight will provide effective solutions to deal with safety-related challenges facing next-generation aircraft[1]. DT will have a huge impact on achieving the designated safety standards and solutions for the future. This groundbreaking technology, when applied to aircraft structures, empowers engineers and decision-makers to comprehensively and continuously monitor and analyze structural behavior throughout the entire life cycle.[12]

2.4. Concept of Digital Twin for Aviation

NASA and the US Air Force have been working to explore the concept of the digital twin to realize condition monitoring, fault diagnosis, life prediction, and design optimization of various types of air vehicles. Li et al. developed a versatile probabilistic model for diagnosis and prognosis, using a dynamic Bayesian network to realize the digital twin[13].In a Bayesian network, random variables are denoted by nodes(vertices) and their dependence relationships are denoted by directed edges; thus, the Bayesian network constitutes a directed

acyclic graph to represent the joint distribution of a set of random variables[10].

Booyse et al. highlighted that current deep learning-based implementations of prognostics and health monitoring are limited to specific equipment or applications[14]. Therefore, they proposed a generic framework in the form of a deep digital twin (DDT), which utilizes deep generative models to generate near-real sensor data under various operational conditions[10]. The DDT is constructed from deep generative models which learn the distribution of healthy data directly from operational data at the beginning of an asset's life cycle[13]. Ritto et al. presented a digital twin framework combining physics-based models and machine learning classifiers for dynamic structural damage detection, but it has limitations in identifying damage outside the pre-trained scope [14]. Dang et al. proposed a cloud computing and deep learning-based digital twin framework for structural health monitoring, which can perform real-time monitoring and proactive maintenance efficiently.[11] DTs technology provides new ideas for vehicle development, intelligent operation and maintenance of infrastructure, virtual test of unmanned driving, and live traffic analysis.[15]

2.5. Major Components of Digital Twin

Since the 2000s, the available components of DTs have been continuously expanded along with the innovation of technologies. Some novel technologies enable DTs to be applied in new domains, such as biochips for human body DTs, 5G, and edge computing in aircraft DTs. The major components of DTs can be divided into three categories based on the function of components:

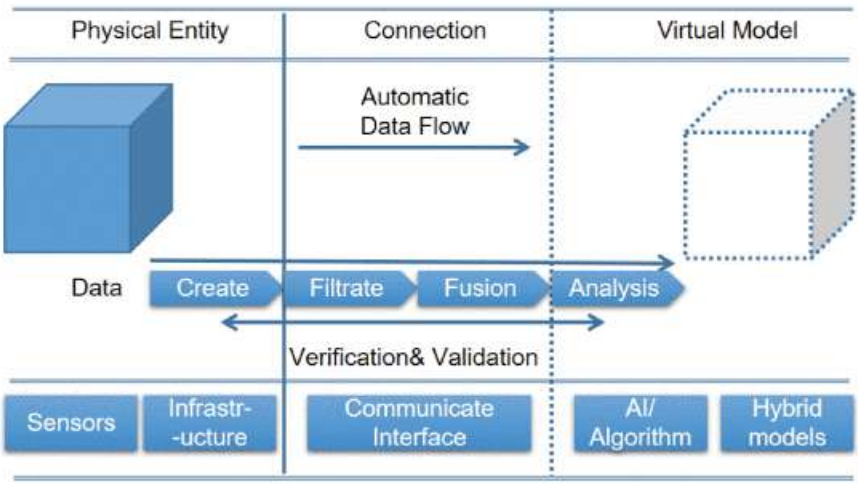

Figure 2.2: *Major Components of Digital Twin*

2.5.1. Physical Side: Sensors and Functional Infrastructure

Physical resources are devices producing information or acting on the environment (sensors and actuators). [9] The state information that defines the physical devices is extracted with the help of sensors embedded into or around them. The data collected from these sensors are aggregated and routed through a gateway that makes the data available for the digital counterpart.[16] Coupled with data transmission technologies, they guarantee that DTs can realize real-time data collection and synchronization.[4] The integration of Wireless Sensor Networks (WSNs) with cloud services forms a foundational aspect of the Internet of Things (IoT), enabling scalable and flexible data collection and sharing across various applications.[17] Additionally, the physical side should have the functional infrastructure to enable and perform specific functions depending on the requirements of DT customers. Components on the physical side are mainly to support data collection and computing. [4]

2.5.1.1 Strain gauges

For reliable operation of a structure, it is important not only to perform a qualitative calculation of its Stress Strain State (SSS) but also to study the SSS of the structure using experimental methods. To measure strains on the surface of a deformable body, various methods are used: mechanical, acoustic, optical, holographic interference, piezoelectric, etc. Currently, the vast majority of measurements are made using strain gauge resistors. [18] A strain gauge is a small sensor that measures how much a material stretches or compresses when a force is applied to it. It works by changing its electrical resistance when the material it is attached to changes shape. By measuring this small change in resistance, we can calculate how much strain the material is experiencing. [19]

The BF3503AA is a type of foil strain gauge that is widely used for basic strain measurement. It is small, easy to attach to different surfaces, and works well with materials like metal, plastic, and composite materials. This gauge is sensitive enough to detect small changes in strain, and it gives reliable results in laboratory and experimental testing setups. One of the useful features of the BF3503AA is that it is temperature compensated, meaning it can handle some changes in temperature without giving incorrect readings. The BF3503AA strain gauge is commonly used for testing wings, beams, and mechanical parts because it is simple, affordable, and effective for most basic strain measurements. [19]

For composite materials, distributed strain sensing techniques, where multiple sensors are positioned along a surface or structural component, have also been found to offer more reliable strain measurements than single-point sensors. This advantage becomes particularly important in anisotropic materials, where the mechanical response can vary significantly even

over small distances. The ability to monitor strain along a continuous line helps to capture variations more accurately, reducing the chances of missing localized stress concentrations or strain irregularities [20]

2.5.1.2 Calibration of strain gauges

Accurate measurement of structural loads on aircraft wings, especially cantilevered wings, relies heavily on the correct calibration of strain gauges. Calibration ensures that the raw electrical signals recorded by strain gauges can be reliably converted into actual physical forces, bending moments, and shear forces acting on the structure. One effective approach to calibrating strain gauges on composite cantilever wings involves applying known external loads and recording the corresponding strain data. In a study on an unmanned aircraft wing, *Święch* applied carefully controlled point loads along the span of the composite wing to create bending and twisting effects. By comparing the strain gauge outputs at different locations on the wing with these known applied loads, calibration factors were developed[21]

A similar approach was used for the Active Aero elastic Wing (F/A-18), where a large-scale calibration process was carried out using hydraulic jacks to apply known loads directly onto the aircraft wing. This setup involved both point loads and distributed loads to simulate different types of forces the wing might encounter during flight. During the calibration, strain gauge responses and wing deflections were recorded to build a comprehensive database of strain-load relationships. This database was later used to derive load equations, which allow researchers and engineers to convert strain data into physical quantities such as bending moments, shear forces, and torque[22]

Both studies highlight the importance of calibration through controlled load application in ensuring accurate structural health monitoring and aerodynamic performance evaluation. By establishing clear relationships between applied loads, strain gauge outputs, and structural deflection, these calibration processes provide a reliable foundation for further analysis and design improvements in wing structures.

2.5.2. Virtual Side: Analytical Models and AI

The reference “digital twin” is formed when data flows between an existing physical object and a digital object, and they are fully integrated in both directions. A change to the physical object triggers a change in the digital object, and vice versa. [23] The main function of the virtual side is to gather, process, and analyze the data. [4] Usually, a CAD file of the physical object is used to represent the digital twin, and the changes are reflected on the same. The

complexity and number of data and control points available to an operator are to be modeled into the digital twin as per requirements. [16] The application is to be applied in such a way that the virtual replica acts identically. Unity (or Unity 3D) is a game engine built for making games. Unity projects themselves are essentially composed of content (assets), scripts (the programming), and the engine itself which smashes everything together and makes it work on your particular platform[24]. With the provision of real-time cloud data it can integrate and provide for real time-visualization; formulating a Digital Twin.

2.5.3. Connection

The essence of connections lies in data communication technologies. In this era of Big Data and IoT, communication technologies are becoming abundant, such as 5G and LoRa for large- scale networks, Wi-Fi for small and medium workshops, and SatCom for aircraft data transmission, etc. Users can choose according to the specific connection requirement[17]. Since the 2010s, with the continuous improvement of computing power, developing data-driven models has become more efficient and in line with the trend of Big Data and IoT[4]. The IoT device interacts with the cloud server to upload the data, and the cloud server will store and host the data[25].The integrity and security represented by the use of cloud computing are considered a priority to manage access to services. Different microservices and protocols must be used to allow data transmission for connection purposes. The development of MQTT protocols to receive data from physical sensors to the development of APIs for communication purposes is used using AWS Cloud platform[26].

Herrera et al describes the backend services of AWS including the use of MQTT protocol to receive sensor data, AWS Lambda to create serverless function, API Gateway using HTTP protocol to manage gateway between physical and virtual connection. The API Gateway integrate orders to the server and serverless functions, managing data that come from the Gateway to target microservices. Unity software and cloud are used to allow users to view data from digital sensors and take control of actuators connected to IoT devices[27]. DTs are growing more adaptable these days. The choices for components and enabling technologies will increase as the “must-have” components continue to advance in sophistication and as prices continue to drop [28]. It will get harder and harder to precisely describe the standard components of DTs as more and more combinations of DT techniques are regularly studied for various domains.

3. METHODOLOGY

3.1. Introduction

We have divided our project methodology into four major sections: Wing Modeling and Sensor Installation, Sensor Calibration, DT development, and validation and image processing, each having different subsections of its own.

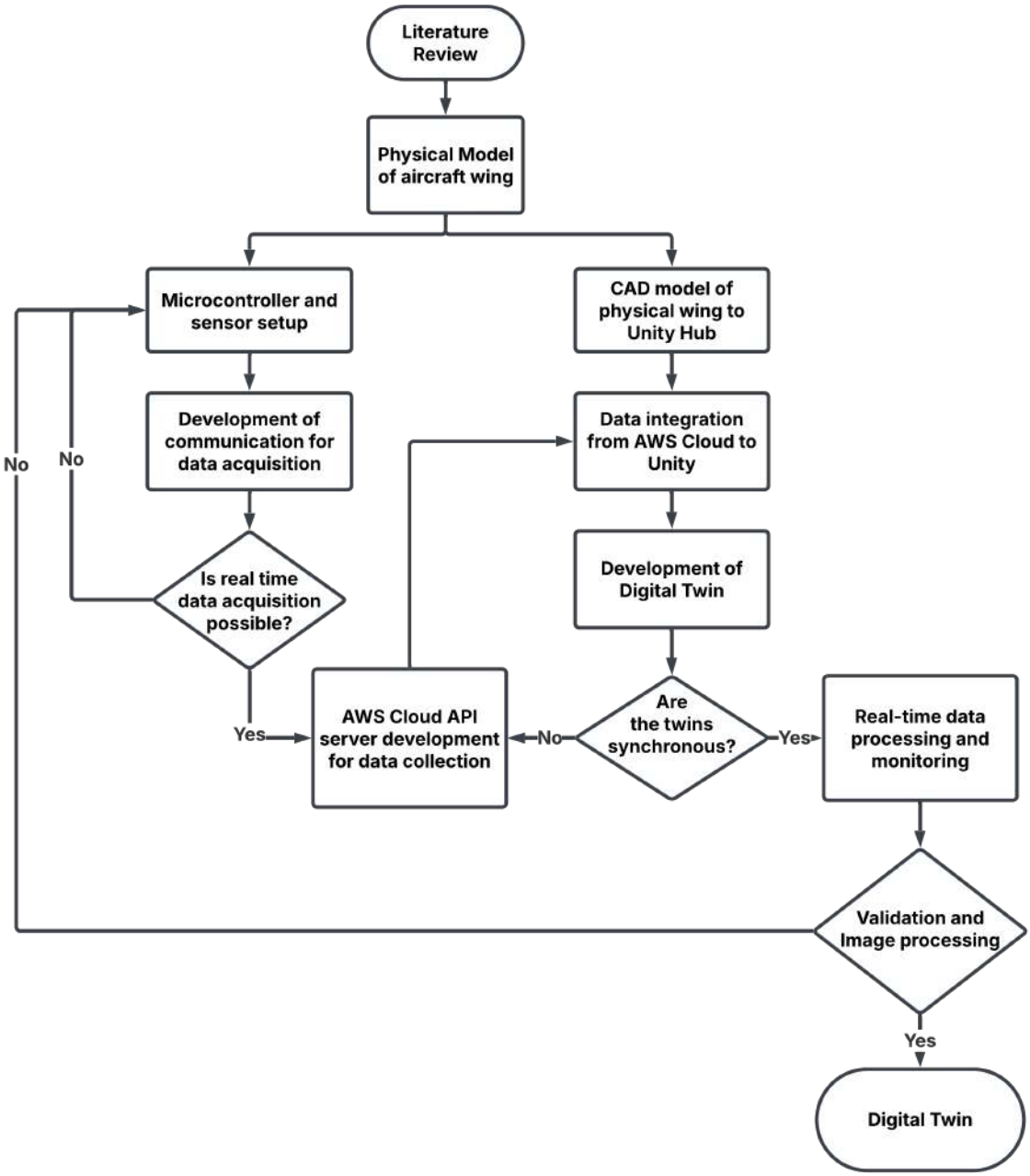

Figure 3.1: *Methodology Flowchart*

3.2. Wing Modeling and Sensor Installation

3.2.1. Wing Modeling

To understand the initial steps involved in developing a Digital Twin, the process began with creating a digital design of the aircraft wing structure using CATIA V5 software. A detailed 3D model of the wing was developed, covering all important aspects such as its geometry, internal structure, and certain functional elements.

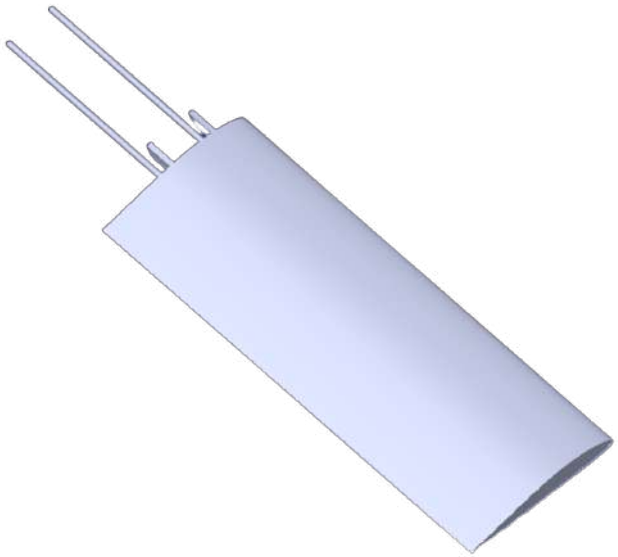

Figure 3.2: *CAD Model of Wing*

A composite wing with a half span of 56.4 cm and a root chord length of 17.25 cm was selected. To achieve a good balance between lift and drag, the Eppler 205 airfoil was chosen because it provides high lift and low drag. The internal structure of the wing was designed with balsa ribs, which were placed evenly along the span to provide internal support and maintain the shape of the wing. The outer skin of the wing was made from fiberglass composite sheets, placed on both the upper and lower surfaces, providing strength, durability, and protection to the internal components. For load-bearing and structural support, two rectangular balsa spars were positioned at 25% and 55% of the chord length, which is close to the aerodynamic center. This positioning helped distribute aerodynamic loads evenly across the wing and reduced twisting forces that could affect stability. To further improve the structural integrity, two cylindrical carbon fiber spars with a radius of 6 x 4 cm were added. These spars acted as additional reinforcements to enhance the overall stiffness and strength of the

wing[29].

After completing the digital modeling, the design was converted into a physical prototype, ensuring the real-world wing closely matched the virtual model. This combined process of digital design and physical fabrication is a key step in building a Digital Twin, where the physical and digital versions of the wing are continuously linked. By following this practical approach, valuable insights into the development process were gained, which also highlighted the potential advantages of using Digital Twin technology for real-time monitoring and analysis.

3.2.2. Sensor Installation

To enable real-time data collection and processing, several electronic components, microcontrollers, and sensors were installed on the physical wing prototype. The ESP32 microcontroller served as the main processing unit, responsible for receiving, managing, and transmitting data gathered from strain gauge sensors placed at different points on the wing.

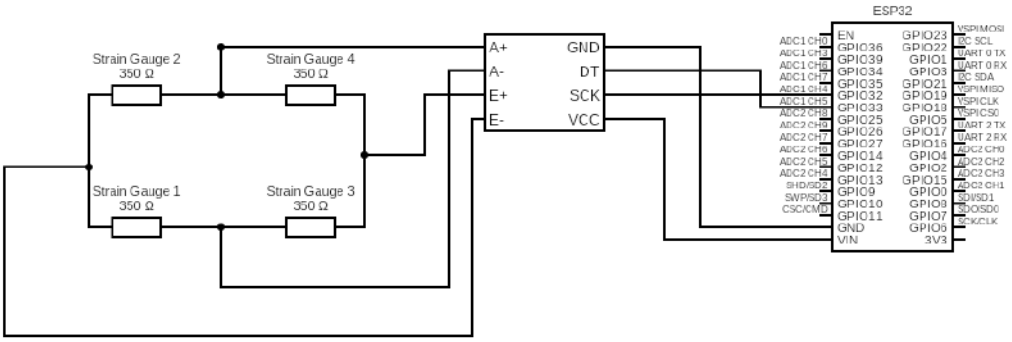

Figure 3.3: *Circuit Design*

The sensors we used were BF3503AA strain gauges, which were placed to capture strain data from both the root and tip sections of the wings. A total of eight strain gauges were installed; four placed around 44 cm from the root (closer to the tip) and four placed around 9 cm from the root (closer to the wing base). This distributed sensor placement ensured more accurate and reliable strain measurement compared to using only one sensor, as stated by A. Freiddi [24].

In order to place the strain gauges, we first cleaned the surface of the wing using iso-propyl alcohol. Once cleaned, the strain gauges were carefully unpacked and placed on the desired position. A small amount of cyanoacrylate was used to attach the strain gauges to the desired location.

Once attached, the strain gauges were connected, and their signals were amplified using the HX711 amplifier module, which is commonly used to read the data from strain gauges. The amplified datas were sent to the ESP32 microcontroller, which acted as the central hub for data collection, processing, and transmission. This sensor setup played a crucial role in enabling real-time monitoring, which is essential for capturing wing deformation and fluttering, forming a key part of the Digital Twin system.

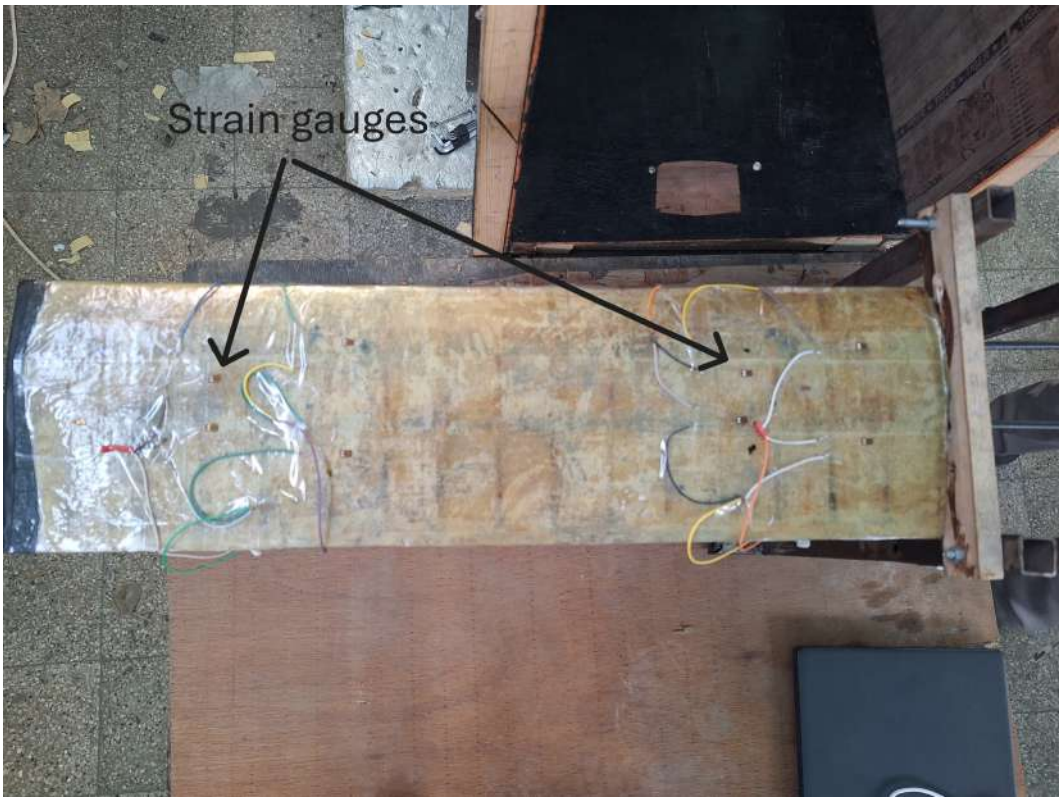

Figure 3.4: *Sensor Installation*

3.3. Sensor Calibration

In order to calibrate the strain gauge, the beam apparatus was used. The manufactured wing had to be cantilevered in order to apply load through the beam apparatus. Thus, in response, we designed a 44 cm tall stand with wooden blocks to fix the spar of the wing into the drilled holes in the wood. A sqaure base was manufactured with welded metal joints and four square metal rods were placed in the edges of the base and welded together. The two wooden blocks were cut and holes were made in the wooden blocks and rods with the help of driller. The wooden blocks were attached to the metal stand with the help of bolts and nuts.

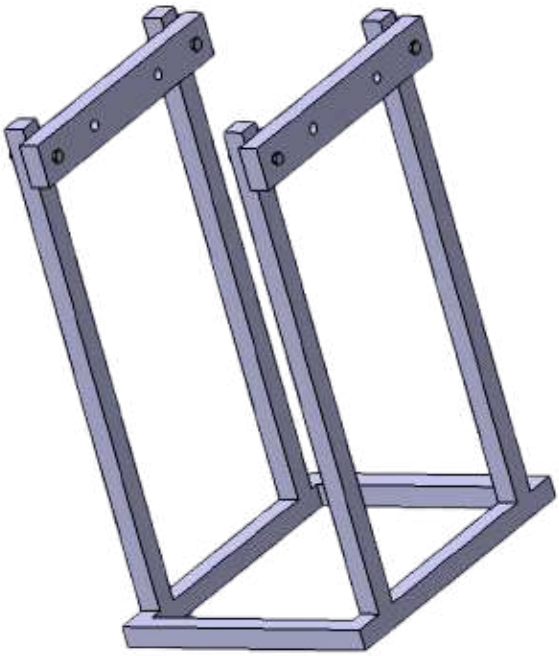

Figure 3.5: *Wing Stand Model*

The wing was then fitted into the stand through the holes in the wooden block in order to cantilever the wing, which is modeled as:

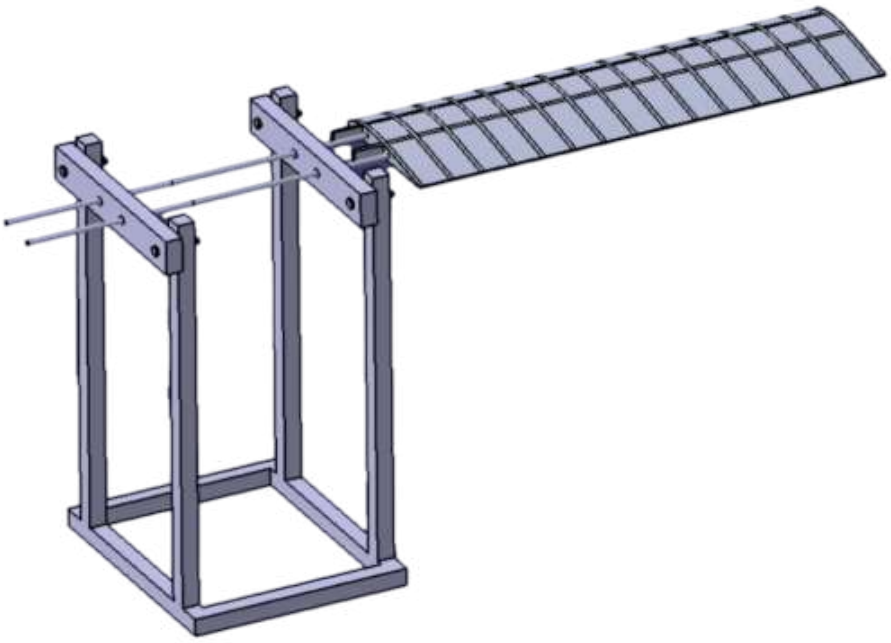

Figure 3.6: *Cantilevered wing model*

After the ESP32 was coded with the help of Arduino IDE, the wing was placed in the beam apparatus and known loads were applied which deflected the wing. The raw values of the strain gauges were collected along with the actual deflection given by the gauge in the point load apparatus for tip and root sensors. A graph was plotted to obtain the calibration factor.

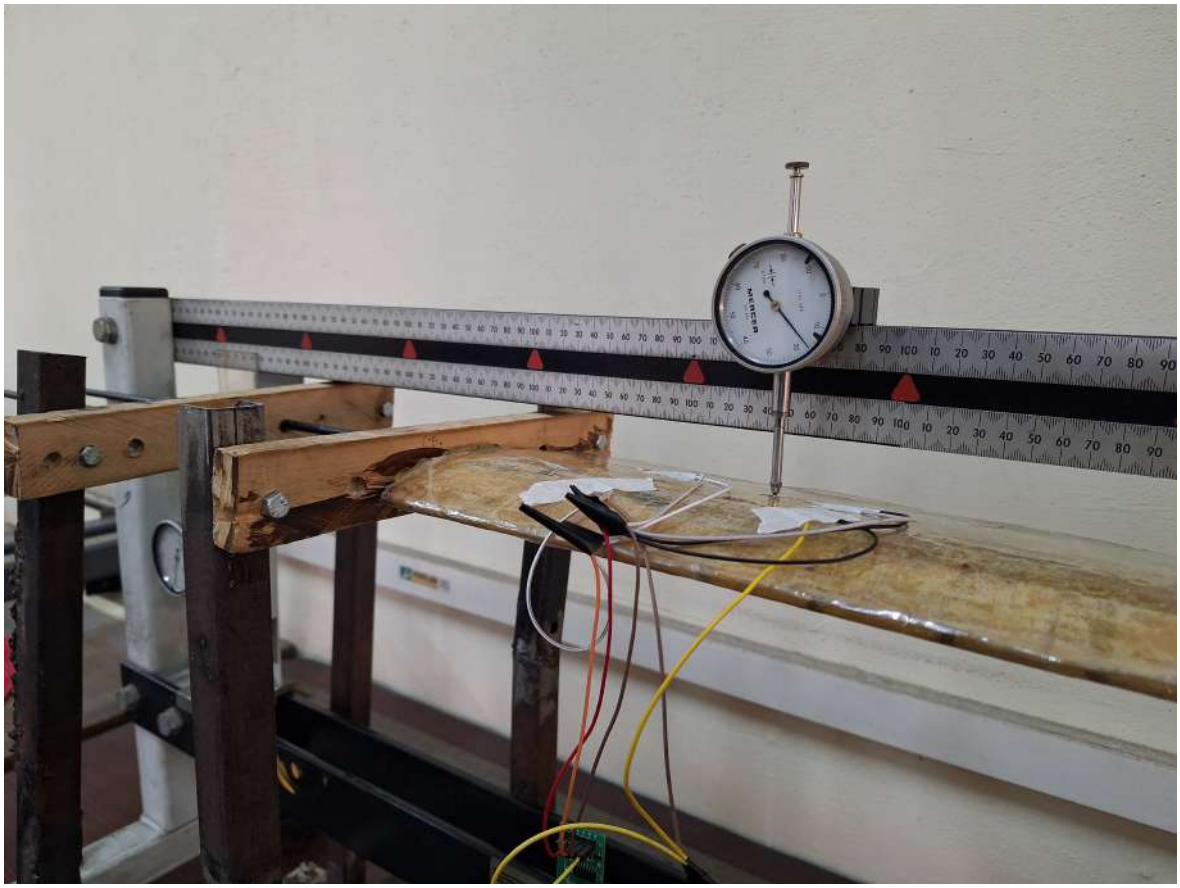

Figure 3.7: *Beam apparatus used for calibration*

The next step was to reduce the noise. For this, the ESP32 was recoded along with the constant calibration factor and moving average filter in-built in Arduino IDE was used to further calibrate and reduce noise. With the help of this, the noise and self-weight deflection of the wing structure were tarred and used for DT development.

3.4. Digital Twin Development

The development of the Digital Twin for the aircraft wing was carried out through a step-by-step process, starting from digital modeling, followed by sensor integration, data transmission, real-time visualization, and data analysis. This process focused to ensure the virtual model accurately reflected the real-world behavior of the physical wing, creating a strong link between the physical model and its digital counterpart.

3.4.1. Data Transmission and Cloud Integration

To allow real-time monitoring, various cloud platforms were explored.

1. Blynk Cloud
   Blynk Cloud was the first option for synchronous data transfer because it was simple and user friendly. It provided real-time monitoring of data and correct device connectivity making it an ideal server for our project. But regional server problems kept us from logging into our account, disrupting our entire process. As real-time synchronization was required, dependence on a platform with access problems was not a possibility.

2. ThingSpeak Cloud
   We then chose ThingSpeak as our cloud server successfully developing our own cloud setup on it. However, upon using our own cloud server and testing, there was a delay of 15 seconds which defeated our main goal to monitor in real time. Furthermore, while using ThingSpeak, we were familiar with MQTT and APIs, which enhanced our understanding of cloud server integration and real-time communication. However, this didn't solve out problem of time delay forcing us to switch to another cloud service.

3. Arduino IoT Cloud
   Next, we attempted Arduino IoT Cloud because we were coding our microcontroller with Arduino IDE and using the same server should be easy. However, we had an issue of not even being able to select our device or access the editor. So, we decided to switch to AWS cloud server which eventually became our desired cloud server.

The ESP32 was connected to AWS IoT Core, an IoT cloud platform. The AWS IoT library was installed in the Arduino IDE, and custom code, along with AWS IoT Core policies, rules, things and certificates were created to send data to the cloud platform and to identify the received data, over a Wi-Fi connection. The data from the strain gauges are collected through the MQTT client. The MQTT Test Client helped simulate how IoT devices send data using the lightweight MQTT protocol, which makes it easier to quickly test the device communications. AWS IoT Core is also responsible for securely receiving and verifying messages from connected devices, while also allowing smooth integration with other AWS tools. The IoT Rules Engine processes incoming data, applies transformations, and sends it to different AWS services as needed. To allow external apps to access this data, the API Gateway offers secure, standardized web-based endpoints. Finally, the Data Publishing Endpoint ensures that processed data can be shared with subscribers in real time, enabling event-driven

processing and allowing the data to be further analyzed or stored. These wireless communication setup enabled the system to continuously upload deflection data to the cloud helping to access remotely as well. Furthermore, a custom API was developed for secure retrieving of data on the cloud platform. This API ensured that data was organized efficiently, and it also allowed remote users to access live data from anywhere. This combination of sensor data transmission and cloud storage formed the backbone of the Digital Twin system, allowing the physical wing's condition to be continuously updated.

In order to send the data to Unity, an API Server was developed to form a gateway using lambda function through which the data received by AWS IoT Core would be sent to Unity Hub.

3.4.2. Real-Time Visualization

The detailed digital model of the wing using CATIA V5 served as the foundation for the Digital Twin development which was imported in Unity and meshed for better visualization of the deflection and fluttering of the wing. Since, the physical wing was cantilevered, a custom C# script was developed to cantilever the digital wing structure to further synchronize with its physical counterpart. To ensure accurate wing deflection animation in UnityHub, we initially tested the virtual setup prior to sensor, cloud storage, and software integration. We created the physical prototype to replicate real-world conditions. We used a meshed CAD model of the wing, which was cantilevered at the root and provided it's structural properties like density, Young's modulus, shear modulus, and Poisson's ratio. Deflection was applied at the points where sensors were integrated in the physical model. We then interpolated the of deflection data imported from a CSV file containing strain gauge measurements into the Unity. When the scene was played, the wing deflection accurately mirrored the physical model's behavior while the data was being taken.

After this, for real-time monitoring the physical model was fed with the combined data received from the AWS IoT Core gateway allowed to view and interact with the live Digital Twin. To achieve this, a custom C# script was developed in Unity to fetch the data from AWS using REST API. The C# script in unity opened the gateway created in the cloud platform and the data received by cloud would transmit to Unity in real-time. Further the data transmitted to Unity had to be connected with the wing model imported. Thus, another custom C# script was developed to connect the data transmitted to Unity from the API gateway to the wing model. This C# script functioned as an instructor or connector to instruct the wing for the needed deformation that has been received through the cloud or to connect the received API cloud data with the prototype wing model. This real-time link allowed the digital wing

model to deflect and respond based on live data, giving a real-time visual representation of the wing's physical condition.

3.4.3. Validation of Synchronization

To ensure the Digital Twin accurately represented the physical wing, a validation process was carried out. The real-time sensor data from the physical prototype was compared with the simulated behavior of the digital model. If any differences or mismatches were detected, the codes were reverified and the sensor was recalibrated to align the physical and digital behaviors. This continuous comparison and recalibration process throughout the test period ensured the synchronization of the twins. As a back-up to the AWS Cloud platform, Things speak and Blynk cloud were taken in the trial and test periods subject to comparison and fulfilment of the needed objectives.

3.4.4. Overall System Refinement and Optimization

Throughout the development process, the Digital Twin system underwent several stages of refinement through various color schemes and calibration, processing and addressing technical issues related to network, error in codes, noise issues and software limitations to improve data accuracy, and visualization capabilities. By combining real-time data collection, cloud-based storage and color gradient, the final Digital Twin system provided a comprehensive platform for monitoring the aircraft wing. This practical application of Digital Twin technology demonstrated its potential benefits in the field of aerospace engineering, particularly for real-time visualization of unmanned air vehicles providing huge asset for future enhancements.

3.5. Validation and Image Processing

Although the synchronization between the Digital Twin and the physical wing was validated visually, it was also important to confirm the accuracy of the data collected from the strain gauges. To do this, theoretical calculations were carried out to estimate the expected range of deformation the wing could experience. These calculated values were then compared with actual data collected during testing. In addition, wind tunnel test was performed, where the wing was left to experience the airflow, and its deformation was recorded using a high-speed camera. The captured images were processed to measure the actual deformation, which was then compared with both the strain gauge readings and the theoretical values. This

combination of theoretical, visual, and sensor data validation helped to ensure the accuracy and reliability of the collected data. In order to obtain this flow of reliable methodology, we went through a series of test and hit. A strain gauge is a delicate sensor and needs to be handled carefully and mounted precisely. Its minute changes in resistance need to be amplified (Y3 module or HX711 amplifier). Being sensitive, strain gauge measurements are susceptible to anomalies and require a controlled experimental environment. Filters can reduce noise but cannot eliminate uncertainties. Mounting takes time, as surface material and surface finish affect performance. Wire and solder with extreme caution not to cause short circuits, which would render the gauge useless. As mentioned earlier, DT development began from modeling, all the way to visualization in Unity. From this, gaining insights into the use of strain gauges as sensors, calibration of data and the use of cloud platform were important. Thus we initiated our project by using a metal scale to understand data calibration and sensor integration.

1. Quarter-bridge strain gauge configuration: We began by familiarizing ourselves with the strain gauge installation process, surface preparation, and circuit configuration. We initially tested a strain gauge with three dummy resistances building a quarter bridge strain gauge configuration on a PLA wing model, but the inherent properties of the material resulted in no viable data. We subsequently attached the strain gauge to a metal scale with the quarter-bridge configuration with a strain gauge Y3 module amplifier, where we were able to obtain a good signal.

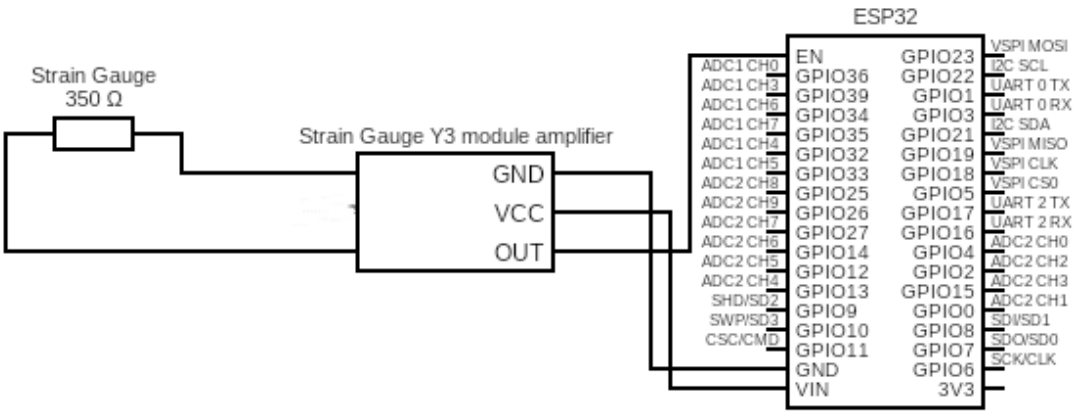

Figure 3.8: *Quarter-Bridge circuit configuration*

2. Full-bridge strain gauge configuration: We first tried to use a quarter-bridge setup, but we were hindered by the high sensitivity of the strain gauge with a small signal range. To improve amplification, we switched to a full-bridge setup on our real wing model

and used the HX711 amplifier, which provided a better gain. This change allowed us to obtain more stable data with less noise, successfully completing the strain gauge configuration.

Followingly, we attempted our DT development in PLA wing. Although we were not able to create Digital Twin of PLA wing due to the material properties, we gained a vast knowledge on the principles of DT technology.

4. RESULTS AND DISCUSSIONS

4.1. Output

4.1.1. Calibration of sensors using beam apparatus

To develop an accurate digital twin for the physical model, the first step involved proper integration and calibration of the sensors. The calibration factor is the ratio of the actual deflection shown by the deflection gauge to the raw value of the strain gauge sensors.

1. Calibration in steel scale
   Strain gauge calibration was a critical part of our project since uncalibrated raw data was useless. We used the Dead Weight Calibration method using a Beam Apparatus. The half bridge scale setup was suspended in a half-bridge mounting from two knife-edge supports, and the strain gauge was mounted centrally with a deflection gauge placed accordingly. A point load was applied near the strain gauge to minimize errors. The applied loads, deflection gauge readings, and ADC values from Arduino's console were taken. By plotting ADC value vs. deflection, we determined the calibration factor to be 0.0022.

Table 4.1: *Load (N), ADC and Deflection (mm) values*

Load (N)	**ADC**	**Deflection (mm)**
7	80	3.9
7.5	193	4
8	350	4.4
8.5	550	4.7
10	900	5.45
14	2100	7.9
16	2700	9.8

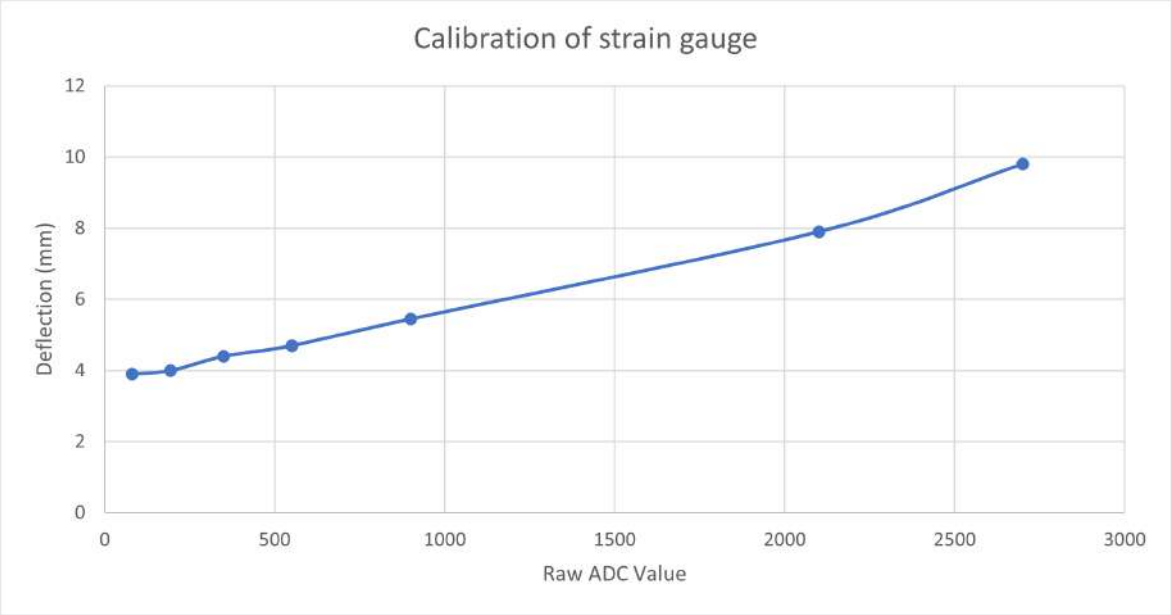

Figure 4.1: Raw ADC values vs. Deflection

2. Calibration of full-bridge configuration on wing model
   For the calibration of the full-bridge configuration, we used the Dead Weight Calibration method that was used earlier. A stand was built to cantilever the wing, simulating real-life conditions. Due to the full-bridge configuration, the load was applied in the center of the configuration to induce minimum error. Tip and root sensors were calibrated, and self-weight noise of the wing was filtered out. With calibration and filter on the test setup, sensor setup was completed.

The calibration graph can be shown below:

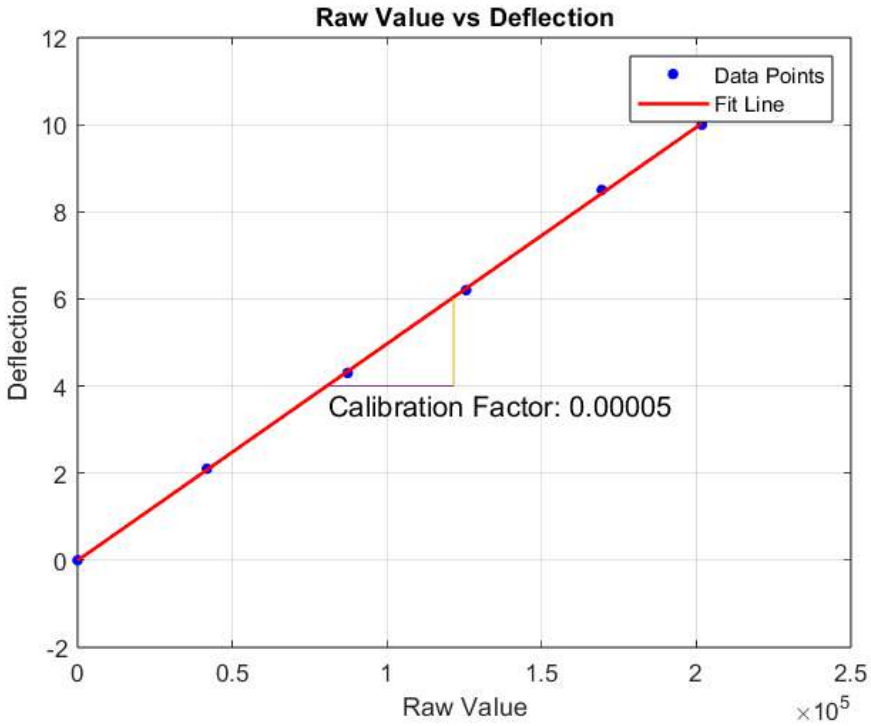

Figure 4.2: *Calibration Graph for Root Deflection*

The result of the calibration graph for the root strain gauge sensors was obtained as 0.00005.

Table 4.2: *Deflection vs. Raw Strain Gauge Value*

Deflection (mm)	**Raw Strain Gauge Value**
0.0	0
2.1	41820
4.3	87265
6.2	125655
8.5	169482
10.0	201813

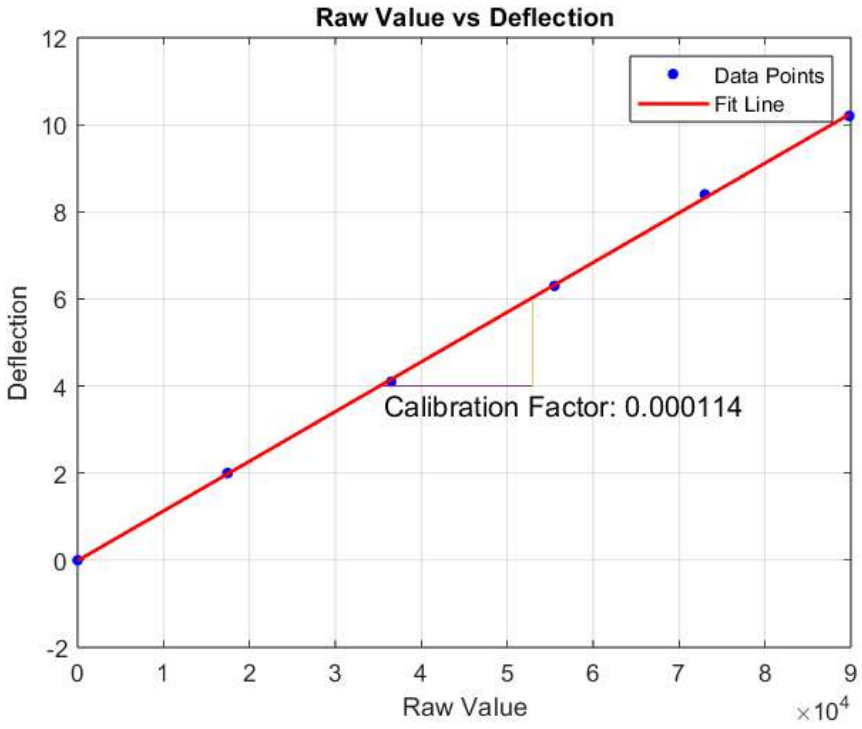

Figure 4.3: *Calibration Graph for Tip Deflection*

The result of the calibration graph for tip strain gauge sensors was obtained as 0.000114.

Table 4.3: *Deflection vs. Raw Strain Gauge Value*

Deflection (mm)	**Raw Strain Gauge Value**
0.0	0
2.0	17456
4.1	36508
6.3	55512
8.4	73140
10.2	89805

Further, for cancellation of noises, the calibrated data was sent through a moving average filter built in Arduino IDE, whose graph has been shown below:

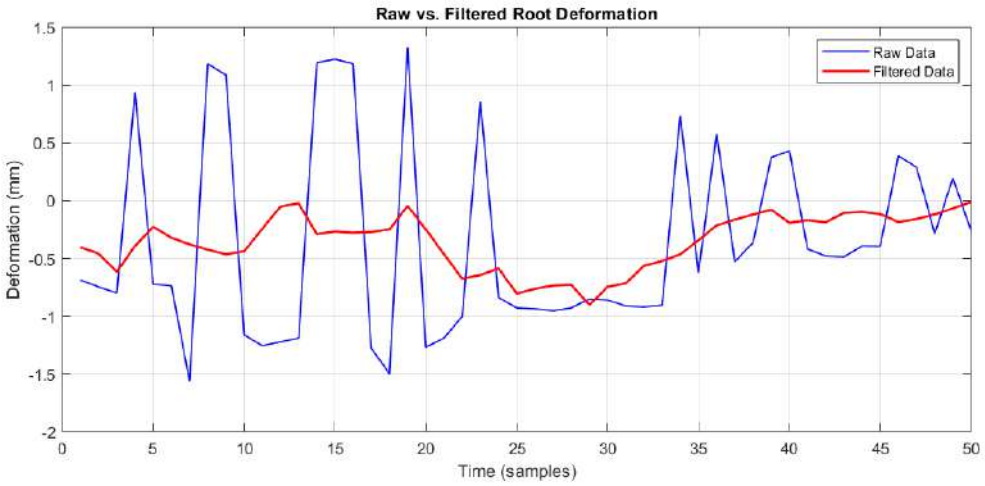

Figure 4.4: *Filtering of data received from root sensors*

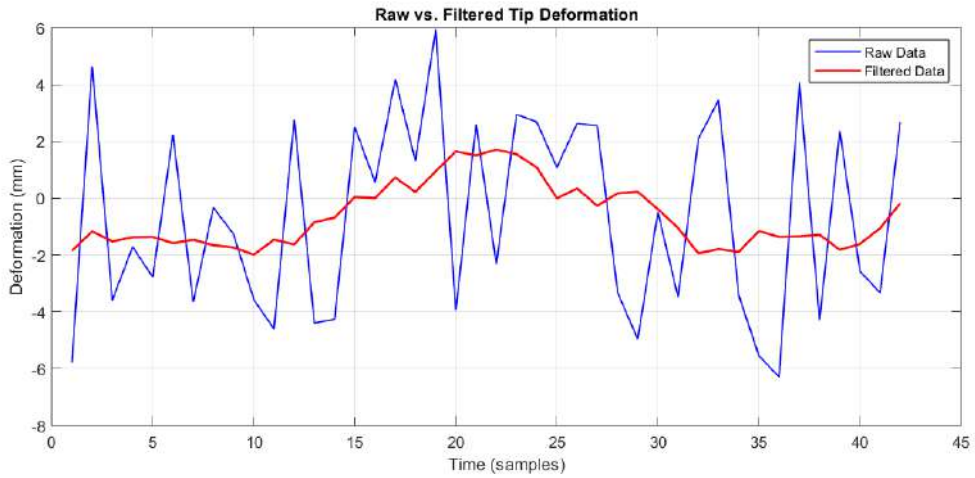

Figure 4.5: *Filtering of data received from tip sensors*

4.1.2. Real-time data communication between cloud and sensors

The calibrated and filtered data was sent to the AWS Cloud server through the ESP 32 microcontroller. As a result, we were able to observe the strain gauge value published in the AWS Cloud platform in real time from the serial monitoring window of the IDE and MQTT Test Client of AWS Cloud.

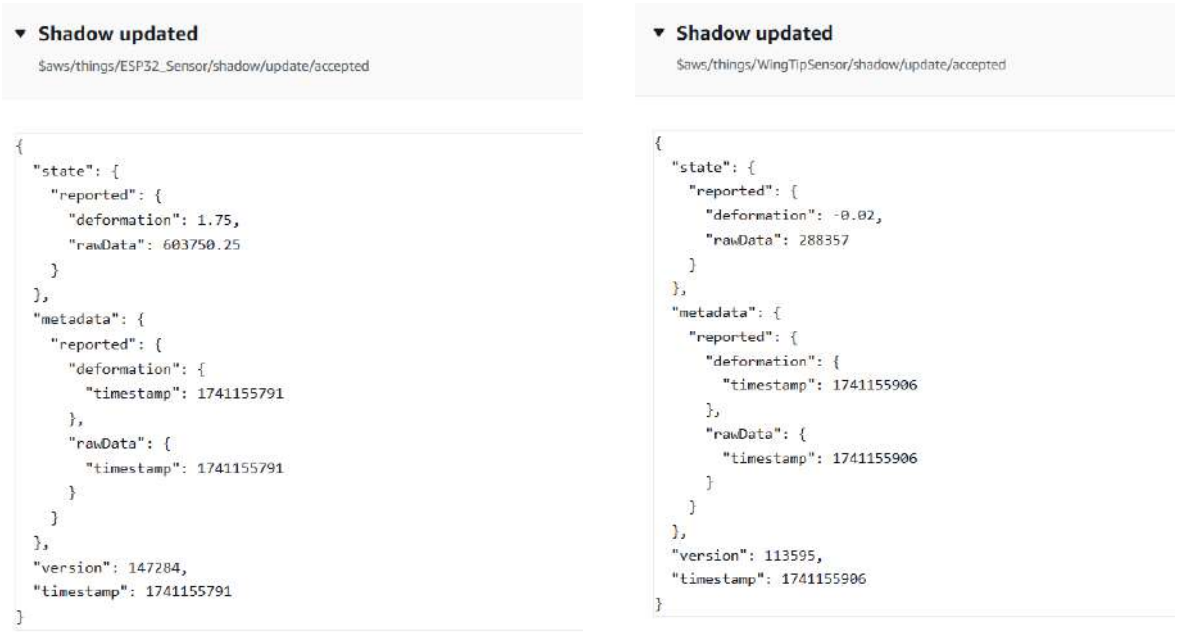

Figure 4.6: *AWS Console*

4.1.3. Development of Digital Twin

The Unity platform cannot retrieve the data directly from the strain gauge, and as a result the data had to be retrieved from the cloud platform. By feeding the data from the cloud server to the imported CAD model of the wing in Unity, we were able to visualize the behavior of the Digital Twin. However, the physical and digital twins were not synchronized. To resolve the issue of synchronization, we increased the rate of data to be published in the Cloud Server and by custom coding Unity to receive a larger number of data through the API HTTP Server Gateway and a better internet connection, we obtained a synchronized behavior of the twins.

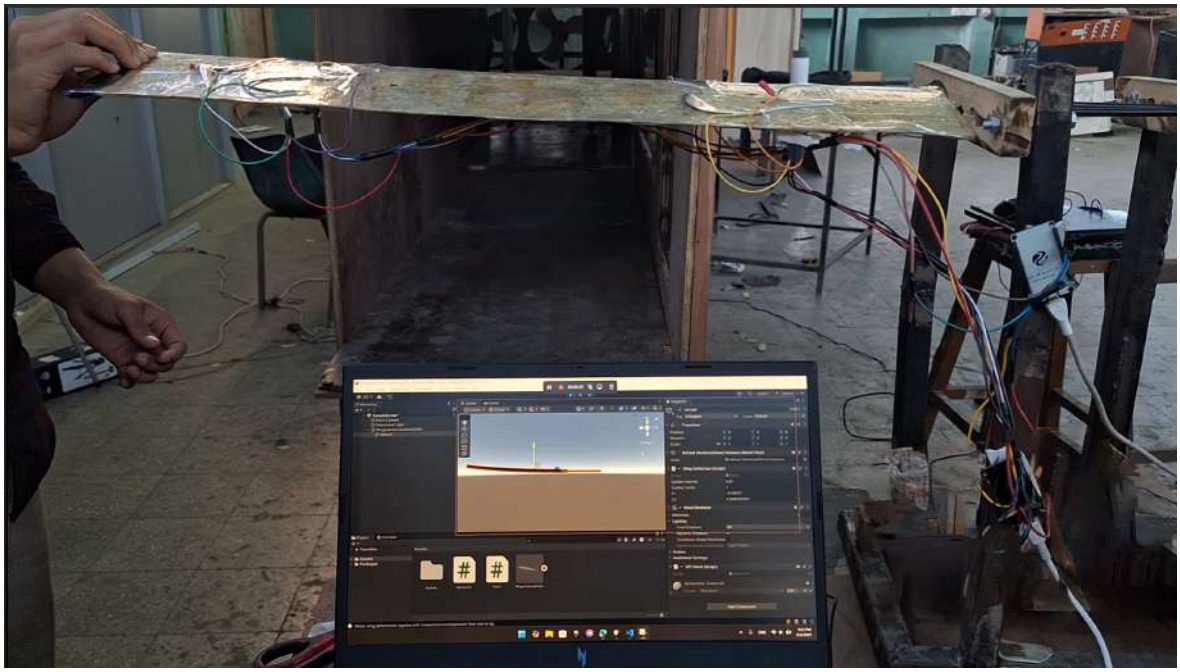

Figure 4.7: *Synchronous upward deflection*

Figure 4.8: *Synchronous downward deflection*

We also custom color-coded the digital twin for better visualization and smoother display.

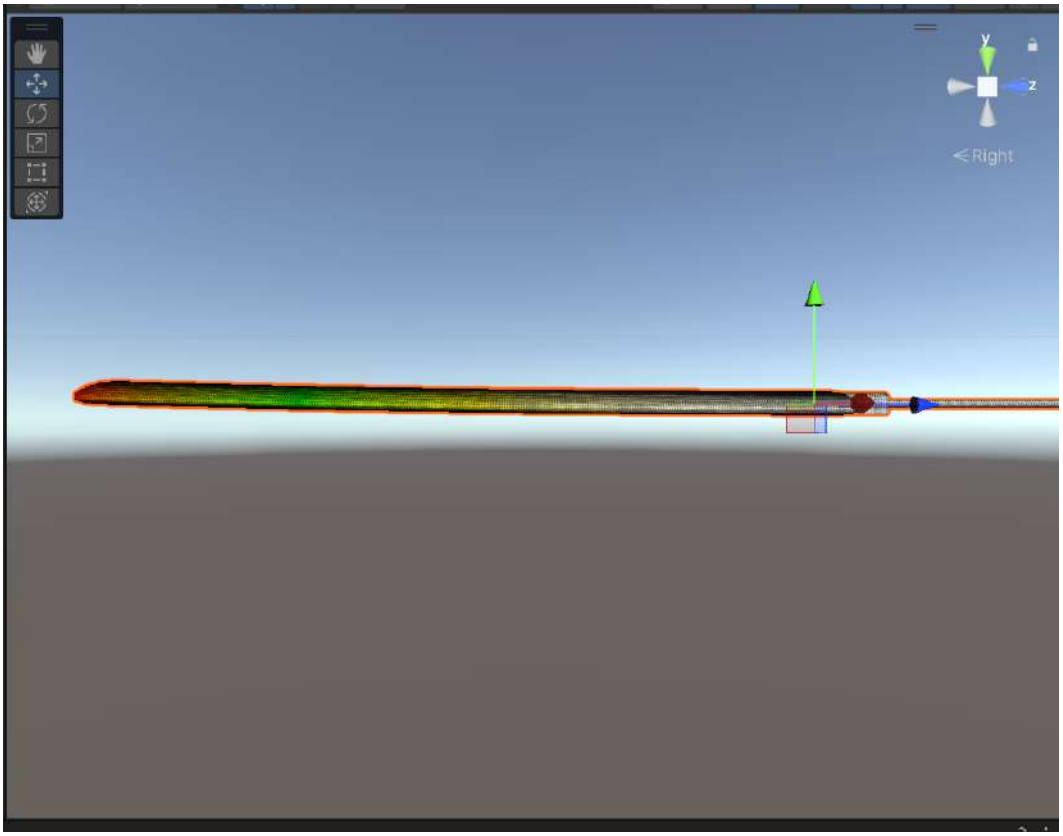

Figure 4.9: *Side view of color coded wing model*

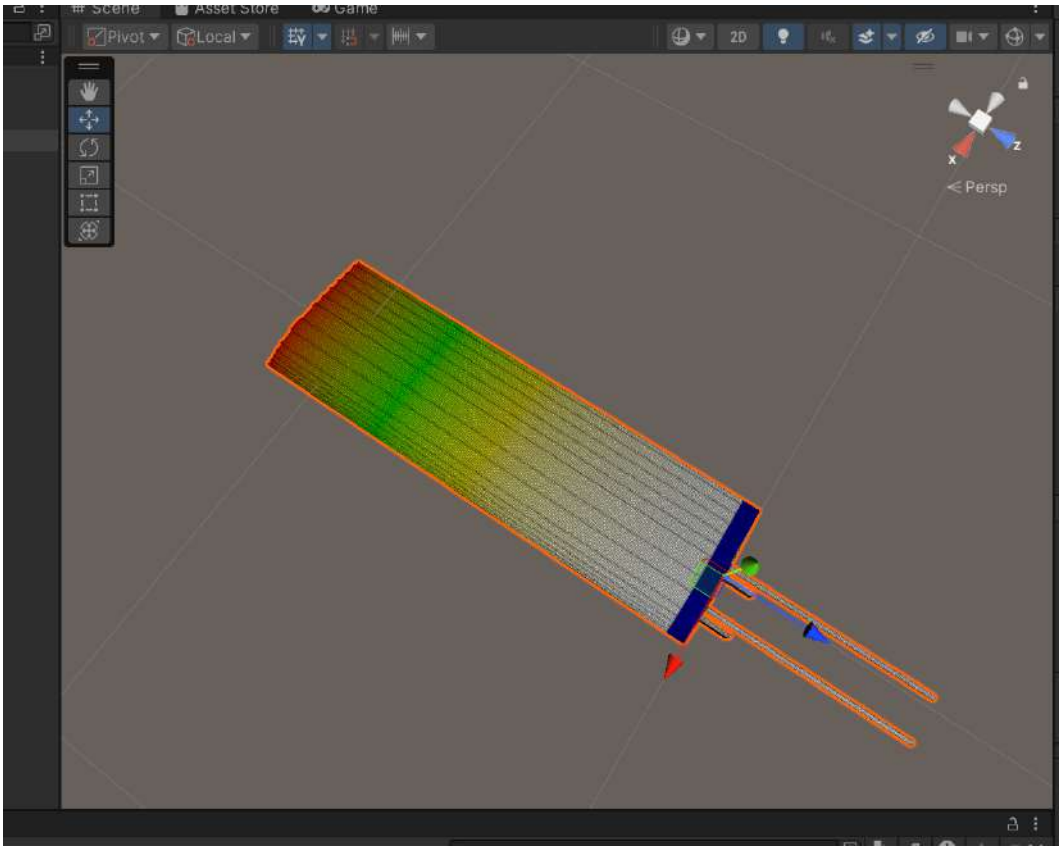

Figure 4.10: *Top View of color coded wing model*

4.1.4. Validation of the digital twin

4.1.4.1 Theoretical Validation First, the wing was subjected to the wind tunnel and the lift and drag forces acting on the wing were calculated.

The value of the lift force generated by the wing was calculated using the lift equation:

$$L = \frac{1}{2} C_l \rho A v^2$$

Where:

- L is the lift force,
- ρ (rho) is the air density,
- v is the velocity of the air relative to the wing,
- A is the reference area (typically the wing area for an aircraft),
- C_l is the coefficient of lift, which depends on the shape of the airfoil and the angle of attack (AoA).

From the above formula, the maximum lifting force (F) on the wing at 5 degrees AOA was found to be 20.274 N [29].

Since the wing structure is cantilevered and rectangular, it was assumed to behave like a cantilever beam with the following material properties:

- Young's modulus (E) = 3 GPa
- Wing chord length (b) = 17.25 cm
- Thickness (h) = 1.8 cm

The moment of inertia (I) was calculated using:

$$I = \frac{1}{12} b h^3 = 8.3835 \times 10^{-8}\,\mathrm{m}^4$$

Since the airflow in the wind tunnel was symmetric and laminar, the flow was considered to be a uniformly distributed load (UDL).

The maximum deflection of the wing was calculated as:

$$\delta_{\max} = \frac{q\,l^4}{8\,E\,I} \quad \text{where}\, q = \frac{F}{L}$$

$$\delta_{\max} = 1.8077\,\text{cm} = 18.077\,\text{mm}$$

When the wing was subjected to airflow in the wind tunnel, the wind speed measured by the anemometer was approximately 8.5 m/s.

The lift force on the wing was recalculated using:

$$L = \frac{1}{2} C_l\,\rho\,A\,v^2 = 4.6498\,\text{N}$$

The corresponding deflection of the wing was:

$$\delta = \frac{q\,l^4}{8\,E\,I} = 4.416\,\text{mm}$$

4.1.4.2 Image processing

The fluttering wing subjected to the wind tunnel was captured using a high-speed camera.

The deflection of the wing when subjected to constant wind of velocity 8.5 m/s is obtained to be around 4 mm. From previous calculations, we observed that the maximum deflection the wing can withstand is around 18 mm. With the graph below, we can visualize the data obtained from the stain gauges along with the actual data shown through image processing. Since the wing was placed slightly behind the test section, the wing will experience a light variation in wind, which led to fluctuations in the obtained data and flutter of the wing. By comparing these results with the actual results from our wing sensors, we can validate our digital twin.

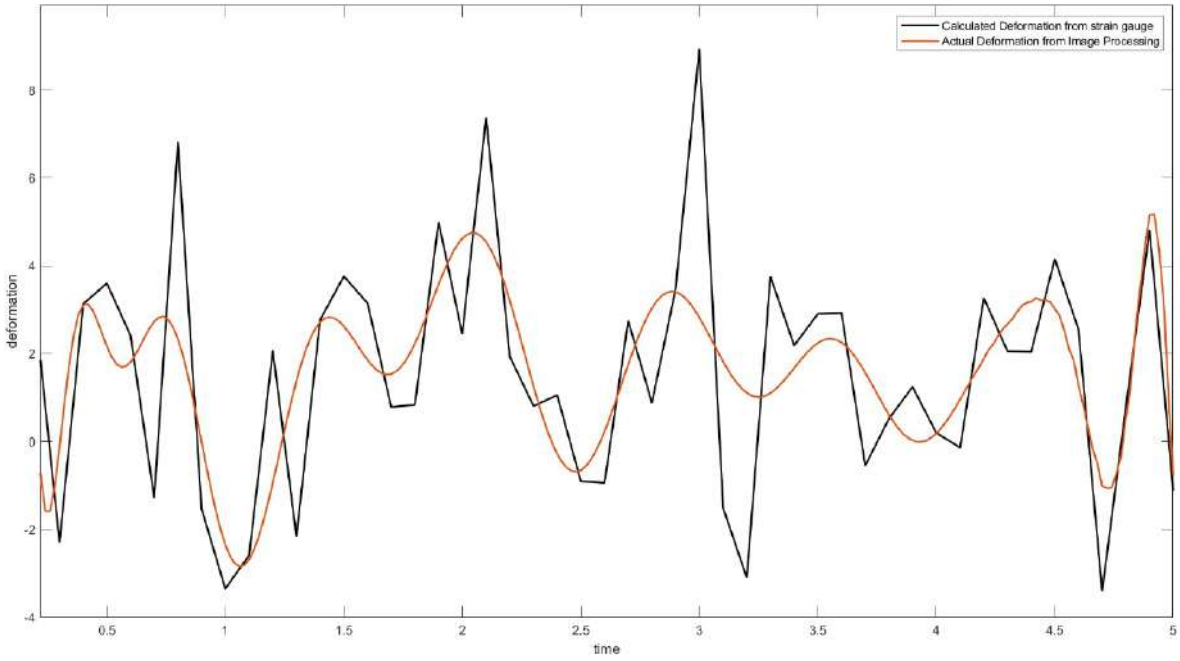

Figure 4.11: *Image processing vs obtained data from strain gauges*

4.2. Limitation

1. Integrating complex systems can introduce additional challenges, disrupting data flow and compatibility.
2. High rate data couldn't be fed in Unity Hub.
3. Using discrete variables for proactive health monitoring.

4.3. Problems Faced

We encountered several challenges during the integration of sensor data with a cloud API to ensure continuous evaluation and minimal delay. One of the issue was managing real-time data collection from the BF3503AA strain gauge through the amplifiers, where noise and scaling inaccuracies required precise calibration and filtering. Connectivity posed another challenge, as configuring the ESP32 for seamless Wi-Fi communication with the cloud involved troubleshooting authentication issues and optimizing protocols for reliability. Further, the unstable internet connection posed additional challenges to reduce lag time between the physical and digital twins. Minimizing time delay further demanded payload optimization, batching, and compression strategies, alongside ensuring that cloud backends like AWS IoT could process data with low latency.

4.4. Implications and Significance

1. The digital twin technology is a real-time monitoring process that analyzes the data in real time.
2. The data acquired in real-time can be used for structural health monitoring.
3. Safety and reliability can be increased through the application of digital twin technology.
4. The digital twin technology is used to identify structural anomalies to detect failures.
5. It allows the researchers to study the different contours of different physical properties like deformation when applying various loads, enabling a better understanding of the structure of the component.

4.5. Budget Analysis

Table 4.4: *Material Cost Table*

S.No.	**Material**	**Quantity**	**Rate (Rs)**	**Cost (Rs)**
1	ESP32 Dev Module	3	900	2700
2	Arduino UNO	2	1500	3000
3	HX711 Amplifier	3	180	540
4	BF3503AA Strain Gauge	10	440	4400
5	BF3505AA Strain Gauge	5	520	2600
6	Strain Gauge Y3 Module Amplifier	9	630	5670
7	AWS Subscription	1	2980	2980
8	Jumper Wire (Male/Female)	80	4	320
9	Miscellaneous	–	–	3500
			Total (Rs)	**23,210**

4.5.1. Gantt Chart

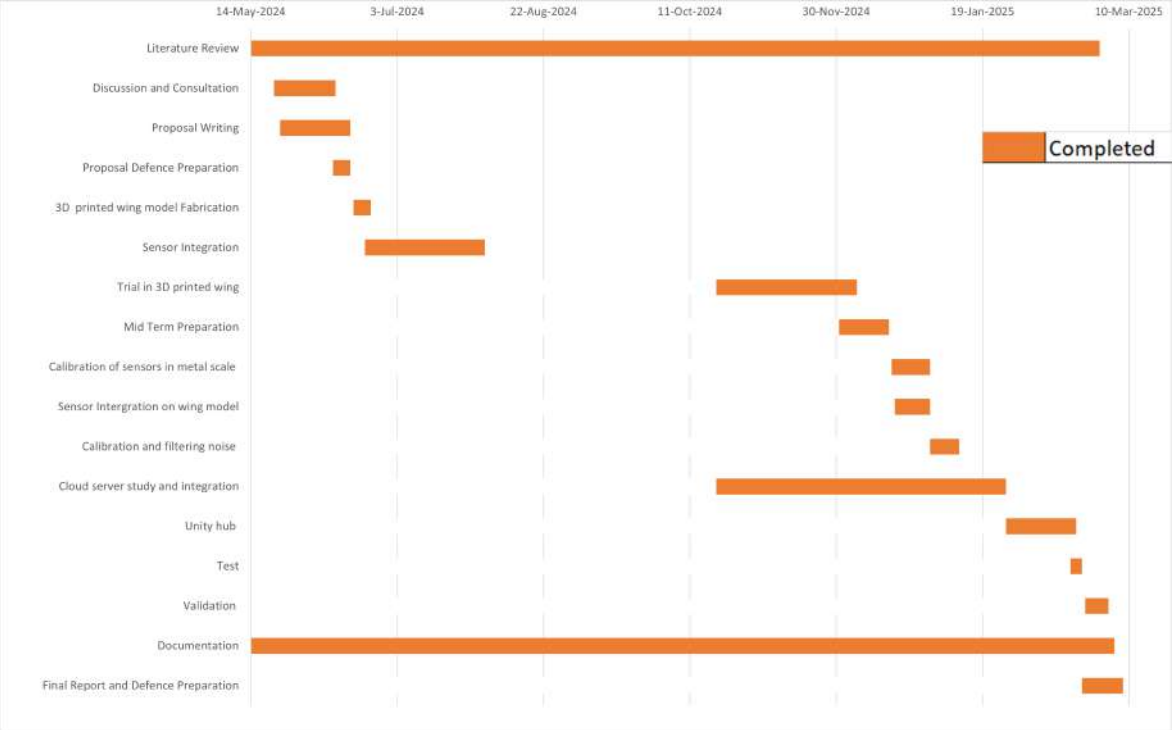

Figure 4.12: Gantt Chart

5. CONCLUSION

5.1. Conclusion

The development of a digital twin for the wing model marks a significant step forward in the use of advanced proactive technologies in the field of aviation. By creating a virtual replica of the physical wing, we were able to visualize and monitor the behavior of the wing, improving our understanding of structural performance. This step in the use of the latest technology serves as a solid foundation for the future integration of SHM systems and feedback systems, enabling proactive monitoring and predictive maintenance in the upcoming stages of the project.

This digital twin technology opens new possibilities for simulating real-world scenarios, optimizing design processes, and evaluating the structural response to external forces, all in a virtual environment. Challenges related to data synchronization, real-time updates, network issues, and system optimization remain, providing additional opportunities for enhancement. In general, this project successfully demonstrates the potential of digital twin technology in the aviation sector, laying the foundation for future advances in safety, operational efficiency, and life-cycle management through the eventual incorporation of real-time structural health monitoring.

5.2. Scope of Future Enhancements

5.2.1. Use of Bayesian Network

In the future, the digital twin can be improved by adding Dynamic Bayesian Networks (DBNs). These are probability-based models that can handle uncertainties like changes in applied loads, material properties, and crack growth. DBNs can combine data from different sources and continuously update the condition of the wing, making predictions more reliable.

Using Particle Filters (PF), the system can also track changes in the health of the wing over time, helping to predict failures and estimate the remaining life of the wing. Adding these methods would make the digital twin even more useful for monitoring and predicting structural issues in the future.

5.2.2. Development of feedback loop or two-way communication between physical and digital system for real-time monitoring and control

A key improvement in our project can the development of a two-way communication system between the physical wing structure and its digital counterpart. Currently, we have been able to use the digital twin for monitoring deformation and fluttering collected from the physical wing. However, by developing a feedback loop, the system can go beyond just observation and enable interaction between the two systems. This means that real-time data from the physical wing will not only be sent to the digital twin but also analyzed instantly. Based on the analysis, responses or corrective actions can be sent back to the physical system if needed. For example, in a long-range drone, if excessive deformation or fluttering of propeller/wings are detected which can be due to turbulences or weather impacts or faults which can be identified by increasing the types of sensors integrated, the system can trigger adjustments, such as aborting the mission if needed or increasing/decreasing heights or changing speeds involuntarily. This would create a closed-loop system with continuous monitoring and automatic adjustments improve reliability and efficiency.

5.2.3. Integration of real-time feedback mechanisms for dynamic control of physical entities through digital twins

Another significant enhancement can be integration of real-time feedback mechanisms to dynamically control the physical wing based on digital twin simulations. Instead of just displaying data on a dashboard, the user will be able to dynamically instruct the physical entity with its digital twin empowering the user with Structural Health Monitoring (SHM) of the aircraft.

For instance, if the digital twin detects excessive strain on a specific part of the wing during wind tunnel testing, it could suggest modification like redistributing load. Also, in a aircraft wing, features like anti-icing, de-icing could be controlled from a ground based pilot as well, though needs a well-established and strong cyber security, this enhancement in larger scale can completely facilitate an already developed air transportation to new heights. The system could also alert engineers in real-time about potential structural weaknesses before they become critical. This integration would make the digital twin more than just a passive tool it would act as an assistant for engineers and researchers, improving safety and efficiency.

5.2.4. Implementation of AI-driven decision-making to activate or adjust systems based on data from digital twins

Furthermore, the capabilities of the system, artificial intelligence (AI) can be integrated to assist in decision-making and proactive analysis. AI algorithms will analyze real-time data from the sensors in the wing and compare it with simulations to predict potential issues before they occur taking DT technology from a passive tool to a proactive front line driver.

For example, if the AI detects an unusual strain pattern that could lead to structural failure, it could automatically suggest adjustments or even activate corrective measures way before it is to happen. Over time, AI can learn from past data to improve its decision-making accuracy, making the system more intelligent and efficient.

By implementing these improvements, the digital twin will evolve into a smarter and more interactive system. It will not only monitor structural changes but also assist in optimizing performance, reducing risks, and improving aircraft wing design through real-time analysis and automated control.

References

[1] B. R. Seshadri and T. Krishnamurthy, "Structural health management of damaged aircraft structures using digital twin concept," in *25th AIAA/AHS Adaptive Structures Conference*, January 2017.

[2] E. H. Glaessgen and D. Stargel, "The digital twin paradigm for future nasa and u.s. air force vehicles," 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:110572580

[3] U. Z. H. Aydemir and U. Durak, "The digital twin paradigm for aircraft review and outlook," in *AIAA Scitech*, January 2020.

[4] A. W. L. Li, S. Aslam and S. Perinpanayagam, "Digital twin in aerospace industry: A gentle introduction," *IEEE Access*, 2022.

[5] J. C. J. F. M. Chiachío, M. Megía and M. L. Jalón, "Structural digital twin framework: Formulation and technology integration," *Automation in Construction*, 2022.

[6] R. R. S. Boschert, C. Heinrich, "Next generation digital twin," in *TMCE*, 2018.

[7] M. Grieves and J. Vickers, "Origins of the digital twin concept," 2016.

[8] Q. Qi and F. Tao, "Digital twin and big data towards smart manufacturing and industry 4.0: 360 degree comparison," *IEEE Access*, vol. 6, 2018.

[9] R. Minerva and N. Crespi, "Digital twins: Properties, software frameworks, and application scenarios," *IT Professional*, Jan.-Feb. 2021.

[10] S. M. C. Li and L. Wang, "Dynamic bayesian network for aircraft wing health monitoring digital twin," in *American Institute of Aeronautics and Astronautics*, January 2017.

[11] S. B. V. Kandasamy, E. Manohar and G. Perumal, "Digital twin in aerospace industry and aerospace transformation through industry 4.0 technologies," in *Digital Twins in Industrial Production and Smart Manufacturing*. John Wiley Sons, Ltd, 2023, ch. 16, p. 381–403.

[12] B. A. R. S. M. O. Tavares, J. A. Ribeiro and P. M. S. T. de Castro, "Aircraft structural design and life-cycle assessment through digital twins," *Designs*, 2024.

[13] D. N. W. W. Booyse and S. Heyns, "Deep digital twins for detection, diagnostics and prognostics," *Mechanical Systems and Signal Processing*, 2020.

[14] X. H. Y. P. X. S. X. Lai, L. Yang and W. Sun, "Digital twin-based structural health monitoring by combining measurement and computational data: An aircraft wing example," *Journal of Manufacturing Systems*, vol. 69, pp. 76–90, 2023.

[15] Y. D. J. Wu, X. Wang and Z. Lv, "Digital twins and artificial intelligence in transportation infrastructure: Classification, application, and future research directions," *Computers and Electrical Engineering*, 2022.

[16] S. Paripooranan and R. Abishek, "An implementation of ar-enabled digital twins for 3d printing," in *2020 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2021.

[17] R. Piyare, "Towards internet of things (iots): Integration of wireless sensor network to cloud services for data collection and sharing," *International Journal of Computer Networks and Communications*, 2013.

[18] V. S. N. Rogacheva and Y. Zheglova, "Piezoelectric gauge of small dynamic bending strains," 2020.

[19] G. O. A. Freddi and L. Cristofolini, *Introduction to the Application of Strain Gages*, 2015.

[20] M. M. A. Baldassarre, J. Ocampo and C. Rans, "Accuracy of strain measurement systems on a non-isotropic material and its uncertainty on finite element analysis," June 2020.

[21] Święch, "Calibration of a load measurement system for an unmanned aircraft composite wing based on fibre bragg gratings and electrical strain gauges," March 2020.

[22] T. C. N. C. R. S. E. Y. R. A. Lokos, C. D. Olney and D. Bessette, "Strain gage loads calibration testing of the active aeroelastic wing f/a-18 aircraft," August 2013.

[23] E. K. M. Balla, O. Haffner and J. Ciga´nek, "Educational case studies: Creating a digital twin of the production line in tia portal, unity, and game4automation framework," *Sensors*, May 2023.

[24] M. B. (PSU.edu), "Beginning data visualization in unity: Scatterplot creation," 2024, available at: https://psu.edu.

[25] N. A. Sabarudin and A. S. Sadun, "Development of a real-time iot data logger for api index monitoring," *Progress in Engineering Application and Technology*, vol. 3, no. 1, pp. 632–646, June 2022.

[26] P. C. Herrera[1], J. Reátegui[1], M. Hurtado[1], and M. Loyola, "Integration of hci-enhancing devices," *Human-Computer Interaction and Beyond: Advances Towards Smart and Interconnected Environments (Part I)*, p. 126, 2021.

[27] R. G. Yauri and Mallqui, "Iot control and visualization system with digital twins and augmented reality in a digital transformation space," *International Journal of Online Biomedical Engineering*, vol. 20, no. 4, p. 18, 2024.

[28] E. K. O. H. R. Leskovský and D. Rosinová, "Proposal of digital twin platform based on 3d rendering and iiot principles using virtual / augmented reality," in *2020 Cybernetics Informatics (KI)*, 2020.

[29] A. Bhandari, "Experimental characterization and aeroelastic analysis of a composite wing," November 2023.

A. TRIAL PHASE WORK

A.1. Trial of Calibration and sensor integration in metal scale

Figure A.1: Calibration trial in metal scale

A.2. Use of 3d printed aircraft wing model

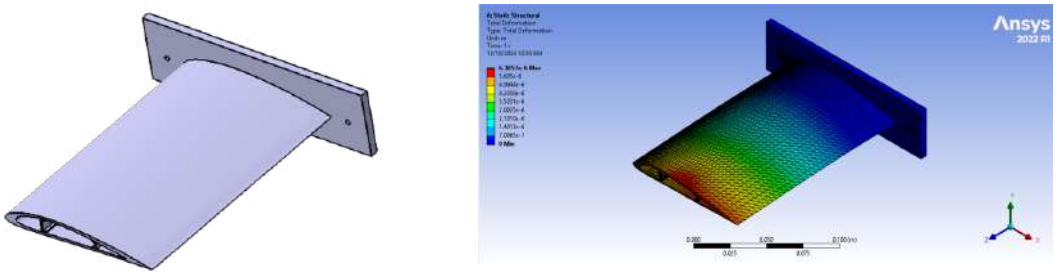

Figure A.2: CATIA Model and Ansys Analysis of 3d wing

Figure A.3: Set-up to test DT in 3d wing

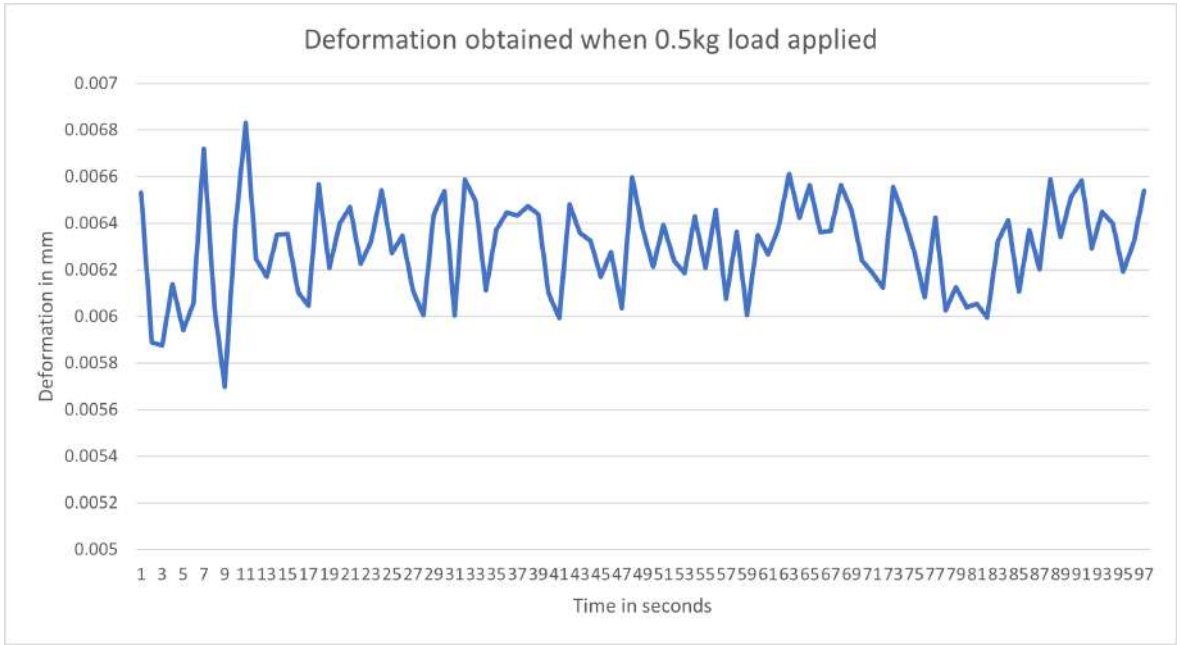

Figure A.4: Deformation data when 0.5 Kg load applied

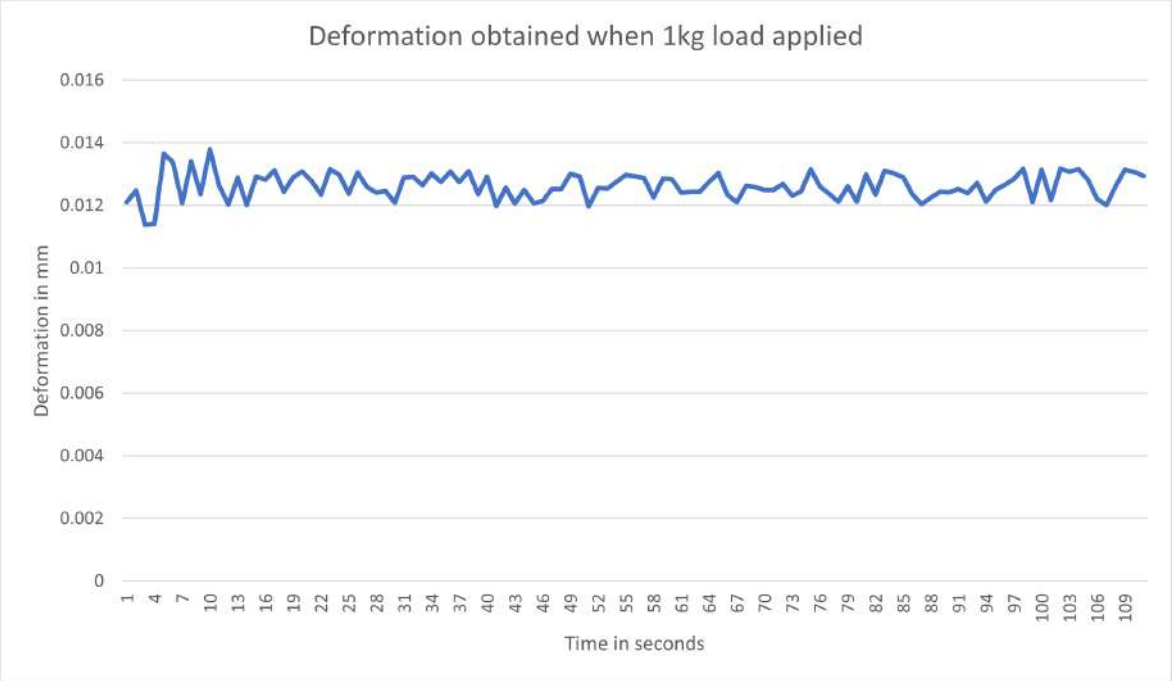

Figure A.5: Deformation data when 1 Kg load applied

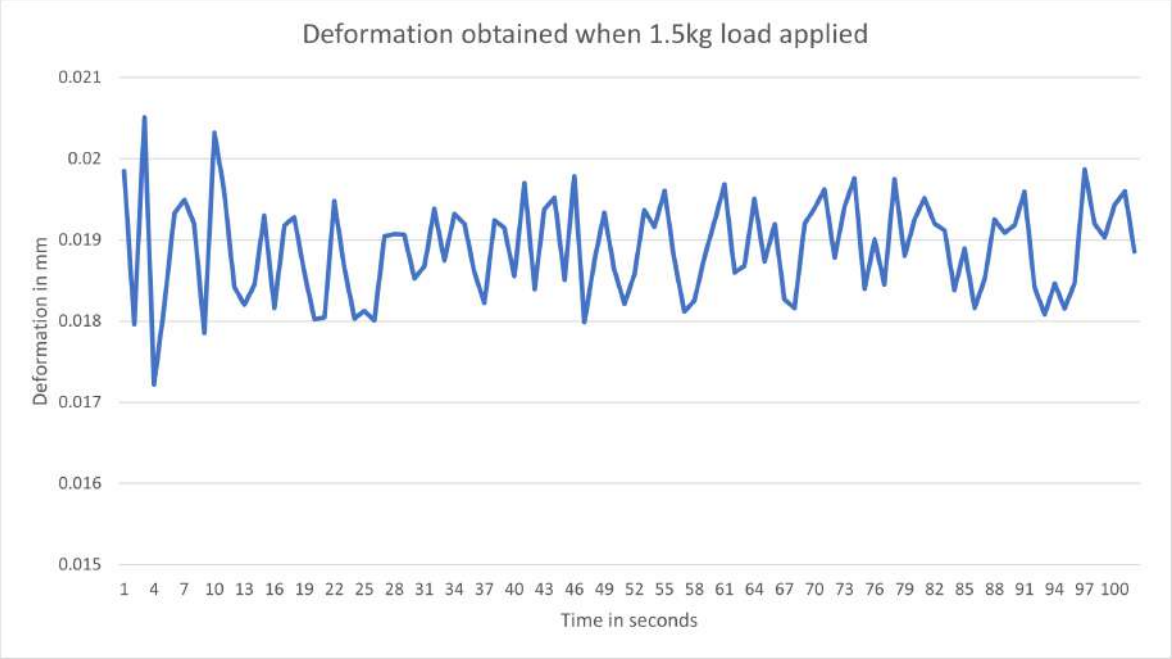

Figure A.6: Deformation data when 1.5 Kg load applied

B. MISCELLANEOUS

Figure B.1: During test for image processing

Figure B.2: 3d printed wing model

Figure B.3: Fiber fabricated wing model

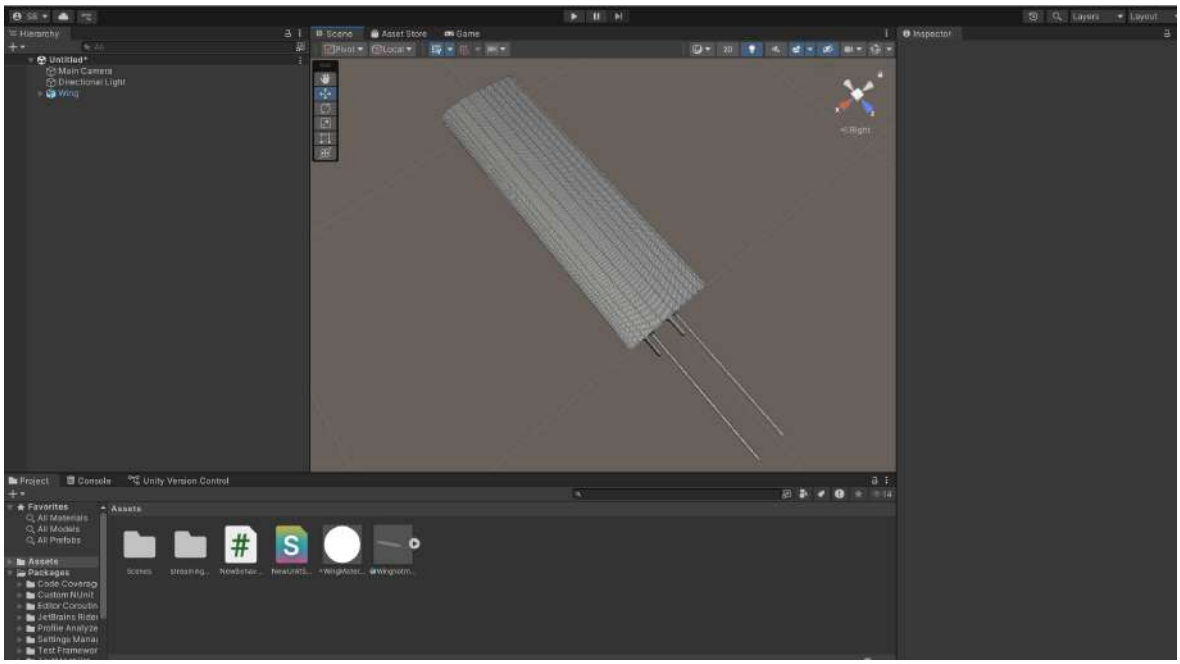

Figure B.4: Shaded wire frame in final Unity wing

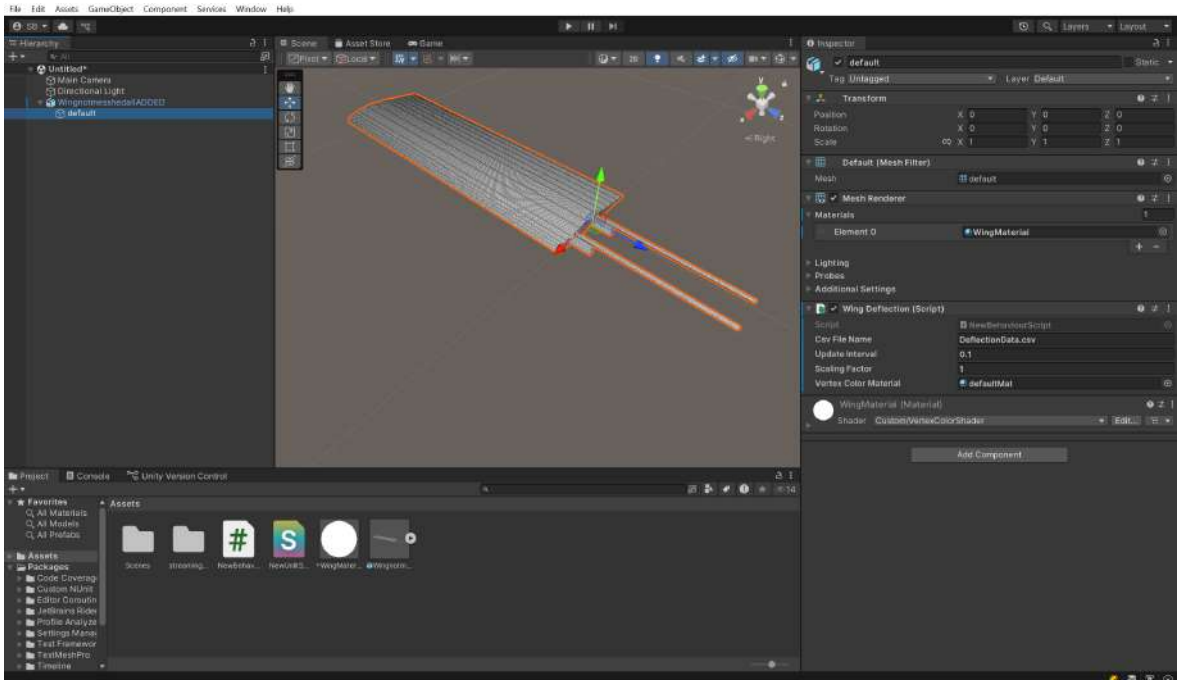

Figure B.5: Unity Interface

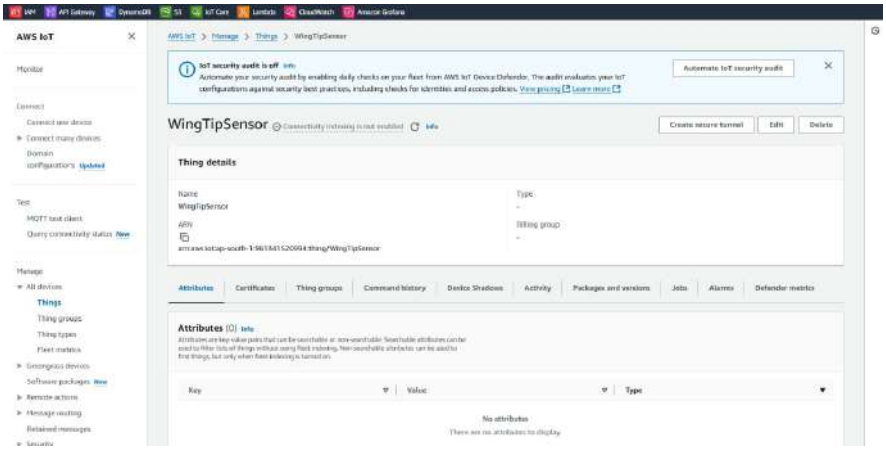

Figure B.6: IoT core interface

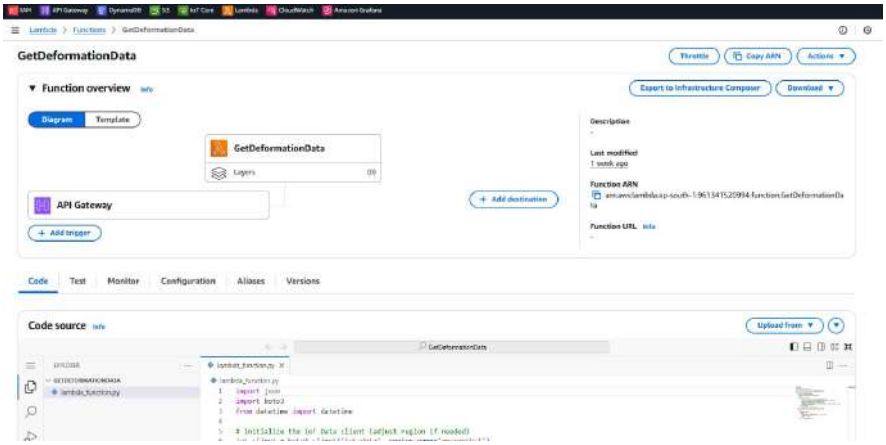

Figure B.7: Lambda Function Interface

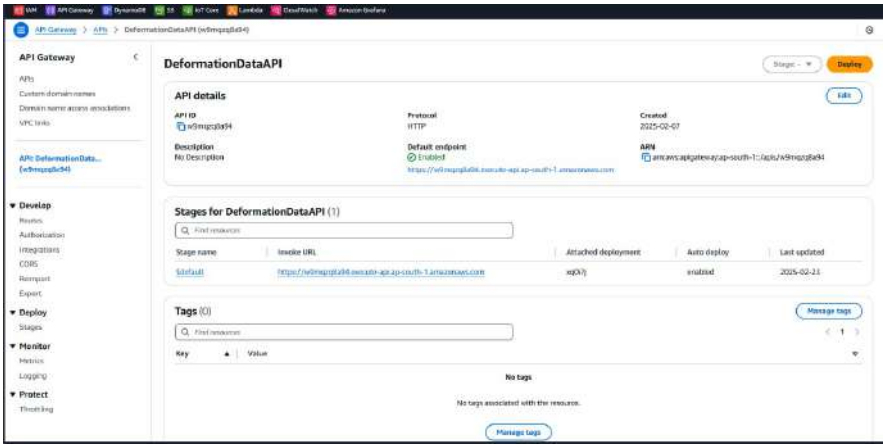

Figure B.8: API Gateway Interface

Figure B.9: Wind Tunnel Set-up

Figure B.10: Image processing y-position track

C. CODES

C.1. Arduino IDE Codes

```
//Arduino code to send root sensor data to AWS Cloud Platform//
#include "secrets.h"
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <HX711.h>
"$aws/things/ESP32_Sensor/shadow/update"

WiFiClientSecure net = WiFiClientSecure();
PubSubClient client(net);

#define DOUT  33
#define SCK   32

HX711 strainGauge;

const float calibrationFactor = 0.00005;
const int NUM_SAMPLES = 10;
float sampleBuffer[NUM_SAMPLES];
int sampleIndex = 0;
bool bufferFilled = false;

void connectAWS();
void sendDataToAWS(float rawData, float deformation);
float applyMovingAverage(float newSample);

void setup() {
  Serial.begin(115200);
  connectAWS();

  Serial.println("Initializing HX711...");
  strainGauge.begin(DOUT, SCK);
  Serial.println("Taring sensor...");
  strainGauge.tare();
  Serial.println("Tare complete. Starting measurements...");
```

```
36
37    for (int i = 0; i < NUM_SAMPLES; i++) {
38      sampleBuffer[i] = 0.0;
39    }
40  }
41
42  void loop() {
43    if (!client.connected()) {
44      connectAWS();
45    }
46    client.loop();
47
48    float rawData = strainGauge.get_units(10);
49    float filteredData = applyMovingAverage(rawData);
50    float deformation = filteredData * calibrationFactor;
51
52    Serial.print("Raw␣Data:␣"); Serial.print(rawData);
53    Serial.print("␣␣|␣␣Filtered␣Data:␣");
        Serial.print(filteredData);
54    Serial.print("␣␣|␣␣Deformation:␣"); Serial.print(deformation,
        6);
55    Serial.println("␣mm");
56
57    sendDataToAWS(filteredData, deformation);
58    delay(1000);
59  }
60
61  void connectAWS() {
62    WiFi.mode(WIFI_STA);
63    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
64    Serial.println("Connecting␣to␣Wi-Fi...");
65    while (WiFi.status() != WL_CONNECTED) {
66      delay(500);
67      Serial.print(".");
68    }
69    Serial.println("\nWi-Fi␣connected!");
70
71    net.setCACert(AWS_CERT_CA);
72    net.setCertificate(AWS_CERT_CRT);
73    net.setPrivateKey(AWS_CERT_PRIVATE);
74
```

```
  client.setServer(AWS_IOT_ENDPOINT, 8883);
  Serial.print("Connecting to AWS IoT...");
  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }
  Serial.println("\nAWS IoT connected!");
}

void sendDataToAWS(float rawData, float deformation) {
  String payload = "{\"state\":{\"reported\":{";
  payload += "\"rawData\":" + String(rawData) + ",";
  payload += "\"deformation\":" + String(deformation);
  payload += "}}}";

  client.publish(AWS_IOT_SHADOW_UPDATE_TOPIC, payload.c_str());
}

float applyMovingAverage(float newSample) {
  sampleBuffer[sampleIndex] = newSample;
  sampleIndex++;

  if (sampleIndex >= NUM_SAMPLES) {
    sampleIndex = 0;
    bufferFilled = true;
  }

  float sum = 0.0;
  int count = bufferFilled ? NUM_SAMPLES : sampleIndex;

  for (int i = 0; i < count; i++) {
    sum += sampleBuffer[i];
  }

  return sum / count;
}
```

```
//Arduino code to send tip sensor data to AWS Cloud Platform//
#include "secrets.h"
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include <PubSubClient.h>
#include <HX711.h>
"$aws/things/ESP32_Sensor/shadow/update"

WiFiClientSecure net = WiFiClientSecure();
PubSubClient client(net);

#define DOUT  33
#define SCK   32

HX711 strainGauge;

const float calibrationFactor = 0.000114;

const int NUM_SAMPLES = 10;
float sampleBuffer[NUM_SAMPLES];
int sampleIndex = 0;
bool bufferFilled = false;

void connectAWS();
void sendDataToAWS(float rawData, float deformation);
float applyMovingAverage(float newSample);

void setup() {
  Serial.begin(115200);
  connectAWS();

  Serial.println("Initializing HX711...");
  strainGauge.begin(DOUT, SCK);

  Serial.println("Taring sensor...");
  strainGauge.tare();
  Serial.println("Tare complete. Starting measurements...");

  for (int i = 0; i < NUM_SAMPLES; i++) {
    sampleBuffer[i] = 0.0;
  }
```

```
157 }
158
159 void loop() {
160   if (!client.connected()) {
161     connectAWS();
162   }
163   client.loop();
164
165   float rawData = strainGauge.get_units(10);
166
167   float filteredData = applyMovingAverage(rawData);
168
169   float deformation = filteredData * calibrationFactor;
170
171   Serial.print("Raw␣Data:␣"); Serial.print(rawData);
172   Serial.print("␣␣|␣␣Filtered␣Data:␣");
        Serial.print(filteredData);
173   Serial.print("␣␣|␣␣Deformation:␣"); Serial.print(deformation,
        6);
174   Serial.println("␣mm");
175
176   sendDataToAWS(filteredData, deformation);
177
178   delay(1000);  // Sampling every second
179 }
180
181 void connectAWS() {
182   WiFi.mode(WIFI_STA);
183   WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
184   Serial.println("Connecting␣to␣Wi-Fi...");
185   while (WiFi.status() != WL_CONNECTED) {
186     delay(500);
187     Serial.print(".");
188   }
189   Serial.println("\nWi-Fi␣connected!");
190
191   net.setCACert(AWS_CERT_CA);
192   net.setCertificate(AWS_CERT_CRT);
193   net.setPrivateKey(AWS_CERT_PRIVATE);
194
195   client.setServer(AWS_IOT_ENDPOINT, 8883);
```

```
  Serial.print("Connecting to AWS IoT...");
  while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
  }
  Serial.println("\nAWS IoT connected!");
}

void sendDataToAWS(float rawData, float deformation) {
  String payload = "{\"state\":{\"reported\":{";
  payload += "\"rawData\":" + String(rawData) + ",";  // Filtered raw data
  payload += "\"deformation\":" + String(deformation);
  payload += "}}}";

  client.publish(AWS_IOT_SHADOW_UPDATE_TOPIC, payload.c_str());
}

float applyMovingAverage(float newSample) {
  sampleBuffer[sampleIndex] = newSample;
  sampleIndex++;

  if (sampleIndex >= NUM_SAMPLES) {
    sampleIndex = 0;  // Wrap around
    bufferFilled = true;
  }

  // Calculate average
  float sum = 0.0;
  int count = bufferFilled ? NUM_SAMPLES : sampleIndex;  // Use fewer samples initially

  for (int i = 0; i < count; i++) {
    sum += sampleBuffer[i];
  }

  return sum / count;
}
------------------------------------------------------------------
------------------------------------------------------------------
//Code to connect the sensors with AWS Cloud//
```

```
#include <pgmspace.h>

#define SECRET
#define THINGNAME "WingTipSensor"

const char WIFI_SSID[] = "Kushal..";
const char WIFI_PASSWORD[] = "#Digitaltwin";
const char AWS_IOT_ENDPOINT[] =
   "a3g5wv43urcd3e-ats.iot.ap-south-1.amazonaws.com";


static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN␣CERTIFICATE-----
MIIDQTCCAimgAwIBAgITBmyfz5m/jAo54vB4ikPmljZbyjANBgkqhkiG9w0BAQsF
ADA5MQswCQYDVQQGEwJVUzEPMA0GA1UEChMGQW1hem9uMRkwFwYDVQQDExBBbWF6
b24gUm9vdCBDQSAxMB4XDTE1MDUyNjAwMDAwMFoXDTM4MDExNzAwMDAwMFowOTEL
MAkGA1UEBhMCVVMxDzANBgNVBAoTBkFtYXpvbjEZMBcGA1UEAxMQQW1hem9uIFJv
b3QgQ0EgMTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALJ4gHHKeNXj
ca9HgFB0fW7Y14h29Jlo91ghYPl0hAEvrAIthtOgQ3pOsqTQNroBvo3bSMgHFzZM
9O6II8c+6zf1tRn4SWiw3te5djgdYZ6k/oI2peVKVuRF4fn9tBb6dNqcmzU5L/qw
IFAGbHrQgLKm+a/sRxmPUDgH3KKHOVj4utWp+UhnMJbulHheb4mjUcAwhmahRWa6
VOujw5H5SNz/0egwLX0tdHA114gk957EWW67c4cX8jJGKLhD+rcdqsq08p8kDi1L
93FcXmn/6pUCyziKrlA4b9v7LWIbxcceVOF34GfID5yHI9Y/QCB/IIDEgEw+OyQm
jgSubJrIqg0CAwEAAaNCMEAwDwYDVR0TAQH/BAUwAwEB/zAOBgNVHQ8BAf8EBAMC
AYYwHQYDVR0OBBYEFIQYzIU07LwMlJQuCFmcx7IQTgoIMA0GCSqGSIb3DQEBCwUA
A4IBAQCY8jdaQZChGsV2USggNiMOruYou6r4lK5IpDB/G/wkjUu0yKGX9rbxenDI
U5PMCCjjmCXPI6T53iHTfIUJrU6adTrCC2qJeHZERxhlbI1Bjjt/msv0tadQ1wUs
N+gDS63pYaACbvXy8MWy7Vu33PqUXHeeE6V/Uq2V8viTO96LXFvKWlJbYK8U90vv
o/ufQJVtMVT8QtPHRh8jrdkPSHCa2XV4cdFyQzR1bldZwgJcJmApzyMZFo6IQ6XU
5MsI+yMRQ+hDKXJioaldXgjUkK642M4UwtBV8ob2xJNDd2ZhwLnoQdeXeGADbkpy
rqXRfboQnoZsG4q5WTP468SQvvG5
-----END␣CERTIFICATE-----
)EOF";

// Device Certificate
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN␣CERTIFICATE-----
MIIDWTCCAkGgAwIBAgIUe8pypyR92vlGiS38Hu6pJyJQT8YwDQYJKoZIhvcNAQEL
BQAwTTFLMEkGA1UECwxCQW1hem9uIFdlYiBTZXJ2aWNlcyBPPUFtYXpvbi5jb20g
SW5jLiBMPVNlYXR0bGUgU1Q9V2FzaGluZ3RvbiBDPVVTMB4XDTI1MDIxODA2NTYw
MloXDTQ5MTIzMTIzNTk1OVowHjEcMBoGA1UEAwwTQVdTIElvVCBDZXJ0aWZpY2F0
```

```
ZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAN4oShY9ryLXmc5rN3tY
k/vPvnvvLYURSDYWukyHFTkMaHPM7AiySf/rm/Vt5Rzc1mPJ5PX69QnFVS5a2LMP
jHH1HGV7TPoAXUMCJTMTFNTSWjN1k+rUc1ZA8fMd0zzbo89EO6vewLMrcDH98GQ1
3RFpgLAd5tLQjteLJmJhajxcpZGMQnZ/Y84RblglqSg3Mr851rTj240xS5qc29CJ
HPxFu2xICbSksIskjcRmJyjcrgDfw27OiYVEN3Fa7E3OmBGpR6XpvxF7x/84M8nP
vTittNVPddKtL1+sIF0ARe4LGGamilMc2EdEBNjuZE/A42b8+CUYo1xQp6Z7jPWn
X5MCAwEAAaNgMF4wHwYDVR0jBBgwFoAU7viTW8pHKrhZYaAZve3kkRmzWFIwHQYD
VR0OBBYEFAD/dwuJyhdlCHspEdLnCMRvFRLlMAwGA1UdEwEB/wQCMAAwDgYDVR0P
AQH/BAQDAgeAMA0GCSqGSIb3DQEBCwUAA4IBAQAZYA58OcdiiQ16dQEKqsHAGTow
tiSDxEG/f2WBywChvAZpjb2aXdB6CvfxwcLqlSJuGYc2DTLcliO4ezyrpLXSydts
AFI3LQ+sTzPg4ziuA3dUUV735lu2BR5I7YqdXscQWbCTrvkjEOjlwRukguE7TesK
NqydO6f89os4D3yDzKP+sPvZS+wxU60jpX7lu4/eZ2eUplyuPHjKtH8JFtQEKNuR
OPtKLXxIqXCHTp6bW4j/TKHs1+BuVgyMANodG2vxIEEAWji54e1zseaKwQgp1s7r
eGUe02FK5Afg9MnTw6mObH9ipcTPFK9XIswmM3CBPv36xK1yO2LUPyqX0+oX
-----END␣CERTIFICATE-----
)KEY";

// Device Private Key
static const char AWS_CERT_PRIVATE[] PROGMEM = R"KEY(
-----BEGIN␣RSA␣PRIVATE␣KEY-----
MIIEpgIBAAKCAQEA3ihKFj2vIteZzms3e1iT+8++e+8thRFINha6TIcVOQxoc8zs
CLJJ/+ub9W3lHNzWY8nk9fr1CcVVLlrYsw+McfUcZXtM+gBdQwIlMxMU1NJaM3WT
6tRzVkDx8x3TPNujz0Q7q97AsytwMf3wZDXdEWmAsB3m0tCO14smYmFqPFylkYxC
dn9jzhFuWCWpKDcyvznWtOPbjTFLmpzb0Ikc/EW7bEgJtKSwiySNxGYnKNyuAN/D
bs6JhUQ3cVrsTc6YEalHpem/EXvH/zgzyc+9OK201U910q0vX6wgXQBF7gsYZqaK
UxzYR0QE2O5kT8DjZvz4JRijXFCnpnuM9adfkwIDAQABAoIBAQC5alL7YRxNLjj2
qVGzJujL4EWRVrxdNWFZk3qF6LBz3sFG9R+C2pscd97A5qm0fzSIMLdh/cSKFSLU
/4mhWTbU3ciHgnNqrOcpyB6YFDLstWwHSvjbo9lngejuClSFlKORl9ddyRp376dC
cjkyyinqIG4Bj9NY3Cx8BW1R6xAps3MAp3O4DHcIzEyToP4003ukk/HkApj0EEkh
uNZxp6WEEuvpCLLu9s54fZhdt95JqE5ZUCMnZ+YWbAKUL0Vm/UUa+2++DsAULLqD
zoNkuZ0azlddm8BamgCrxiUWDGxzwgMEppJajBC/ghiusuU3x+O9w9tbtsbKh313
xF3O+pShAoGBAPqw/KrTZN08zb8Vn2qY8zvgkMBxRK+KbZRiP1wKla2nYsMSKUCl
gCXytHg/sIX2UH74Eq3A/vb1K3zQQ/427h4j6Sux0Uxci2+kdVrWevZH2fEWu3Qz
0amIysS9DhUYIC6nM/qw6RK22skH32qQr2mqmfFv1PFibBuyPKdiFFGpAoGBAOLc
njYv+mU4LwnN4EFvJElHey5SINQybB1aOOYZjIxcGftWlqfssoYf2w+iRNEprjC6
/C850fFuD7g0N+sXwOvwM0lfego2GbcbtqLBwNhBc6Gw9BK9PW/aRjtbuEGgorPg
CIVQJ9S/skECCz26efgJ3/SULKqCfTKTAnx2E+TbAoGBANcGeiK4IZrWLZAKJuvS
90bTrrUyMnzzGlydDMWQ2qPRoBe7+yBVFJ74F63Qy0OMa60McVER6l+/GSb8qGVj
jkjR+o0c/24lLwB6qqJYt62JIbn2CqF4VQy6pVms42hagjgh+Ka9mZ0QtIacNxj7
s0nKKDL7/kVxa1MyRW445T7JAoGBAN0XYWEx9Pe8f6swZ0OeC2EXNud3xuLGmTMq
AJXPhHguMmwOgRj6xymL1jQwUpz93/NB6BAma4K9XriIu450QeIu3+y8DfXxr7IS
```

```
316 YjXQWh7n+scz60pv/3YI5bVnt/TNloNy7RrveUIP6KsOUOJB4Chph0eXOMicD2PA
317 fgXKFa1NAoGBAK1zxTz1RMnFEGR/bCdFGKeayqK5xsQZ4aQF69VoBQKoBOi+Zunc
318 aAUN68QiEiq9ikkVqPG/3jCUFvw6AzH4vX4HdJfdzNDGvQOtQXHd2BlDSvcu5DEt
319 4ffxE2uKY9/TPa/JLVHGICcIFDM6wWZsDnCsmq7LLZ+zWC9SV4bxoCLB
320 -----END␣RSA␣PRIVATE␣KEY-----
321 )KEY";
```

C.2. MATLAB Code for Image Processing

```
%Time_VS_Dy_PLOT (HIGHSPEEDCAMERA)
clc;
clear;
close all;

dataDir =
    'C:\Users\kusha\Desktop\dt_chronus\deformationframe_1';
fps = 500;
n_im = 2500;
ppmm = 3.8;

time = (0:n_im-1) / fps;

h1 = 300; h2 = 1000; xstart = 450; xend = 1400;
n_xp = 3;  % Number of tracking points

data_maps = struct();
for n = 1:n_im
    data_maps.(sprintf('im_%d', n)) = read_tif(h1, h2, xstart,
        xend, dataDir, n);
end

def_profiles = struct();
for i_im = 1:n_im
    ylocs = y_locs_tif(data_maps.(sprintf('im_%d', i_im)),
        h2-h1, linspace(100, xend-xstart-100, n_xp));
    def_profiles.(sprintf('im_%d', i_im)) = ylocs(:,1);
end

def_ydat = struct2array(def_profiles) / ppmm;
def_ydat = def_ydat';

time_shifted_def_y = def_ydat(1,1) - def_ydat(:,1);

figure;
plot(time, time_shifted_def_y, 'k-', 'LineWidth', 2);
xlabel('Time [s]');
ylabel('\DeltaY [mm]');
set(gca, 'FontSize', 14, 'FontName', 'Times');
```

```
grid on;

function [sch_map] = read_tif(h1,h2,xstart,xend,wd,i_im)
    filename=[wd '/frame_' num2str(i_im,'%06.0f') '.tiff'];
    [Im] = imread(filename);
    sch_map = double(Im(h1:h2,xstart:xend));
end
function [ylocs] = y_locs_tif(iIm,y,xlocs)
    ylocs = nan(length(xlocs),2); % The first column is
       populated with dy data
    for jj = 1:length(xlocs) % jj is x , or columns
        j = xlocs(jj);
        for i = y-20:-1:20  % i is y , or rows
            if abs(iIm(i-10,j)-iIm(i,j)) > 80
                ylocs(jj,1) = i-4;
                break;
            end
        end
    end
end
```

C.3. Unity Codes

```
//Data from AWS to Unity Code//
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;

public class APIClient : MonoBehaviour
{
private string apiUrl =
"https://w9mqzq8a94.execute-api.ap-south-1.amazonaws
.com/deformation";

    private WingDeflection wingDeflectionScript;

    [System.Serializable]
    public class DeformationData
    {
        public float ESP32_Sensor;
        public float WingTipSensor;
        public string timestamp;
    }
    void Start()
    {
        wingDeflectionScript =
           FindObjectOfType<WingDeflection>();
        StartCoroutine(FetchDataLoop());
    }

    IEnumerator FetchDataLoop()
    {
        while (true)
        {
            yield return StartCoroutine(GetAPIData());
            yield return new WaitForSeconds(0.1f); // Fetch
               data every 0.1 seconds (1/10th of a second)
        }
    }
    IEnumerator GetAPIData()
    {
```

```
        using (UnityWebRequest request =
            UnityWebRequest.Get(apiUrl))
        {
            yield return request.SendWebRequest();

if (request.result == UnityWebRequest.Result.ConnectionError ||
request.result == UnityWebRequest.Result.ProtocolError)
{
Debug.LogError("API Error: " + request.error);
}
else
{
string jsonResponse = request.downloadHandler.text;
DeformationData data =
    JsonUtility.FromJson<DeformationData>(jsonResponse);

                float esp32Deflection_m = data.ESP32_Sensor /
                    1000f;
                float wingTipDeflection_m = data.WingTipSensor
                    / 1000f;
 {esp32Deflection_m}"); {wingTipDeflection_m}");
                Debug.Log($"Timestamp: {data.timestamp}");
                wingDeflectionScript.SetDeflectionValues
(esp32Deflection_m, wingTipDeflection_m);
            }
        }
    }
}
-----------------------------------------------------------------
-----------------------------------------------------------------
//Code to see the real time deformation in Unity 3D//
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEditor;

public class WingDeflection : MonoBehaviour
{
    public float updateInterval = 1f;
    public float scalingFactor = 1.0f; // Adjust to match
       simulation/ANSYS scale
```

```
73
74      private MeshFilter meshFilter;
75      private Vector3[] originalVertices;
76      private Vector3[] modifiedVertices;
77      private Color[] vertexColors;
78
79      private float rootZ;
80      private float tipZ;
81      private float wingSpan;
82
83      public float D1 = 1f;
84      public float D2 = 0f;
85
86      private const float sensorDistance = 0.5f;
87
88
89      public float pointLoad = 10f;
90      public float uniformLoad = 5f;
91      public float appliedMoment = 20f;
92
93      void Start()
94      {
95          meshFilter = GetComponent<MeshFilter>();
96
97          if (meshFilter == null || meshFilter.mesh == null)
98          {
99              Debug.LogError("   MeshFilter or Mesh not found!");
100             return;
101         }
102
103         Mesh mesh = Instantiate(meshFilter.mesh);
104         mesh.MarkDynamic();
105         meshFilter.mesh = mesh;
106
107         originalVertices = mesh.vertices;
108         modifiedVertices = new Vector3[originalVertices.Length];
109         vertexColors = new Color[originalVertices.Length];
110
111         FindRootAndTipZ();
112         ApplyDefaultColors();
113         StartCoroutine(UpdateDeflection());
```

```
114     }
115
116     void FindRootAndTipZ()
117     {
118         rootZ = float.MinValue;
119         tipZ = float.MaxValue;
120
121         foreach (Vector3 v in originalVertices)
122         {
123             if (v.z > rootZ) rootZ = v.z;
124             if (v.z < tipZ) tipZ = v.z;
125         }
126
127         wingSpan = Mathf.Abs(rootZ - tipZ);
128         Debug.Log($"  Wing Root at Z={rootZ}, Tip at
                Z={tipZ}, Span={wingSpan} meters");
129     }
130
131     IEnumerator UpdateDeflection()
132     {
133         while (true)
134         {
135             ApplyDeflection(D2);
136             yield return new WaitForSeconds(updateInterval);
137         }
138     }
139
140     void ApplyDeflection(float esp32Deflection)
141     {
142         float t_sensor = sensorDistance / wingSpan;
143
144
145         for (int i = 0; i < originalVertices.Length; i++)
146         {
147             Vector3 vertex = originalVertices[i];
148             float distanceFromRoot = rootZ - vertex.z;
149             float t_total = distanceFromRoot / wingSpan;
150
151
152             float loadDeflection = 0f;
153
```

```
154
155             if (pointLoad != 0)
156             {
157                 loadDeflection = (pointLoad /
                        Mathf.Pow(wingSpan, 3)) *
                        Mathf.Pow(distanceFromRoot, 3) *
                        scalingFactor;
158             }
159             else if (uniformLoad != 0)
160             {
161                 loadDeflection = (uniformLoad / (2 *
                        Mathf.Pow(wingSpan, 3))) *
                        Mathf.Pow(distanceFromRoot, 2) * (3 *
                        wingSpan - distanceFromRoot) * scalingFactor;
162             }
163             else if (appliedMoment != 0)
164             {
165                 loadDeflection = (appliedMoment / (2 *
                        Mathf.Pow(wingSpan, 2))) * (wingSpan -
                        distanceFromRoot) * scalingFactor;
166             }
167             float interpolatedDeflection = (esp32Deflection /
                    Mathf.Pow(t_sensor, 3)) * Mathf.Pow(t_total, 3) *
                    scalingFactor;
168             float finalDeflection = loadDeflection +
                    interpolatedDeflection;
169
170             modifiedVertices[i] = new Vector3(vertex.x,
                    vertex.y + finalDeflection, vertex.z);
171
172             float normalizedDeflection =
                    Mathf.Clamp(finalDeflection /
                    Mathf.Abs(finalDeflection), -1f, 1f);
173             vertexColors[i] = Color.Lerp(Color.red, Color.blue,
                    (normalizedDeflection + 1f) / 2f);
174         }
175
176         meshFilter.mesh.vertices = modifiedVertices;
177         meshFilter.mesh.colors = vertexColors;
178         meshFilter.mesh.RecalculateNormals();
179         meshFilter.mesh.RecalculateBounds();
```

```
180
181         Debug.Log("   Whole wing deformation applied with
                interpolation and load distribution.");
182     }
183
184     void ApplyDefaultColors()
185     {
186         for (int i = 0; i < vertexColors.Length; i++)
187         {
188             vertexColors[i] = Color.white;
189         }
190         meshFilter.mesh.colors = vertexColors;
191     }
192
193     public void SetDeflectionValues(float esp32Deflection,
            float wingTipDeflection)
194     {
195         D2 = esp32Deflection;
196         D1 = wingTipDeflection;
197     }
198 }
199 ------------------------------------------------------------------
200 ------------------------------------------------------------------
201 \\Code to cantilever the wing\\
202 using UnityEngine;
203 using System.Collections;
204
205 [RequireComponent(typeof(MeshFilter))]
206 public class CantileverWingDeflection : MonoBehaviour
207 {
208     public float updateInterval = 1f;
209     public float scalingFactor = 1.0f;
210
211     private MeshFilter meshFilter;
212     private Vector3[] originalVertices;
213     private Vector3[] modifiedVertices;
214     private Color[] vertexColors;
215
216     private float rootZ;
217     private float tipZ;
218     private float wingSpan;
```

```
219
220     public float D1 = 1f;
221     public float D2 = 0f;
222
223     private const float sensorDistance = 0.5f;
224
225     void Start()
226     {
227         meshFilter = GetComponent<MeshFilter>();
228
229         if (meshFilter == null || meshFilter.mesh == null)
230         {
231             Debug.LogError("  MeshFilter or Mesh not found!");
232             return;
233         }
234
235         Mesh mesh = Instantiate(meshFilter.mesh);
236         mesh.MarkDynamic();
237         meshFilter.mesh = mesh;
238
239         originalVertices = mesh.vertices;
240         modifiedVertices = new Vector3[originalVertices.Length];
241         vertexColors = new Color[originalVertices.Length];
242
243         FindRootAndTipZ();
244         ApplyDefaultColors();
245         StartCoroutine(UpdateDeflection());
246     }
247
248     void FindRootAndTipZ()
249     {
250         rootZ = float.MaxValue;
251         tipZ = float.MinValue;
252
253         foreach (Vector3 v in originalVertices)
254         {
255             if (v.z < rootZ) rootZ = v.z;
256             if (v.z > tipZ) tipZ = v.z;
257         }
258
259         wingSpan = Mathf.Abs(tipZ - rootZ);
```

```
260         Debug.Log($"    Wing Root at Z={rootZ}, Tip at
                Z={tipZ}, Span={wingSpan} meters");
261     }
262
263     IEnumerator UpdateDeflection()
264     {
265         while (true)
266         {
267             ApplyCantileverDeflection(D2);
268             yield return new WaitForSeconds(updateInterval);
269         }
270     }
271
272     void ApplyCantileverDeflection(float esp32Deflection)
273     {
274         float t_sensor = sensorDistance / wingSpan;
275         float sensorDeflection = esp32Deflection *
                scalingFactor;
276
277
278         for (int i = 0; i < originalVertices.Length; i++)
279         {
280             Vector3 vertex = originalVertices[i];
281             float distanceFromRoot = vertex.z - rootZ;
282             float t = distanceFromRoot / wingSpan;  //
                    normalized span position (0 at root, 1 at tip)
283
284
285             float deflection = 0f;
286
287             if (t <= t_sensor && t_sensor > 0)
288             {
289
290                 deflection = sensorDeflection * (t / t_sensor);
291             }
292             else
293             {
294
295                 deflection = sensorDeflection * Mathf.Pow(t /
                        t_sensor, 3);
296             }
```

```
297
298             modifiedVertices[i] = new Vector3(vertex.x,
                    vertex.y + deflection, vertex.z);
299
300
301             float normalizedDeflection =
                    Mathf.Clamp01(Mathf.Abs(deflection /
                    sensorDeflection));
302             vertexColors[i] = Color.Lerp(Color.red, Color.blue,
                    normalizedDeflection);
303         }
304
305
306         meshFilter.mesh.vertices = modifiedVertices;
307         meshFilter.mesh.colors = vertexColors;
308         meshFilter.mesh.RecalculateNormals();
309         meshFilter.mesh.RecalculateBounds();
310
311         Debug.Log("    Cantilever wing deformation applied.");
312     }
313
314     void ApplyDefaultColors()
315     {
316         for (int i = 0; i < vertexColors.Length; i++)
317         {
318             vertexColors[i] = Color.white;
319         }
320         meshFilter.mesh.colors = vertexColors;
321     }
322
323     public void SetDeflectionValues(float esp32Deflection,
        float wingTipDeflection)
324     {
325         D2 = esp32Deflection;
326         D1 = wingTipDeflection;  // This could be used for
                validation against predicted tip deflection
327     }
328 }
```