University of Pavia

Faculty of Engineering

Department of Electrical, Computer and Biomedical Engineering

Master Degree in Industrial Automation Engineering

Visualization tool of temporal data collected from the bridge sensors

Supervisor:
Prof. Tullio Facchinetti

Candidate:
Mina Rezaei Aderyani

Academic Year 2024/2025

Abstract

The Structural Health Monitoring (SHM) industry faces fundamental challenges, particularly in transforming large volumes of sensor data into understandable information for real-time decision-making. This thesis presents a lightweight and scalable system for real-time visualization of bridge health-monitoring data. The proposed system consists of a modular architecture that reads, synchronizes, and processes raw data, and then converts them into an intuitive visual representation. The methodology includes real-world data collection, the design of data-transmission mechanisms, and the development of an interactive user interface. The results show that the system can correctly display various sensor positions, identify their status through color rendering, and enable users to explore historical data. The system operates in two modes—real-time mode for immediate monitoring and playback mode for review and analysis—providing a high degree of flexibility.

Although this study demonstrates promising results, it also faces limitations such as dependence on the quality of input data, the absence of complete environmental information, and the lack of advanced anomaly-detection algorithms. Nevertheless, future developments—such as integrating intelligent damage-detection methods, implementing automated alert systems, and adding three-dimensional visualization capabilities—can enhance the system's performance and transform it into an effective tool for improving safety, reducing maintenance costs, and supporting sustainable infrastructure management.

Freedom means having the power to choose to be yourself, even when the world wants something else.

Long live Iran.

Acknowledgments

I would like to sincerely thank my supervisor and professor, Tullio Facchinetti, for his patience, kindness, and constant support throughout my thesis. His encouragement and belief in me motivated me to complete this work.

I am also sincerely grateful to my family, whose unconditional love and encouragement have sustained me during my studies and throughout my life. Their belief in me has been a source of strength and motivation.

My sincere appreciation goes to the brave people of Iran who are standing for freedom and a better tomorrow.

Contents

List of Figures

List of Tables

Acronyms

UNIPV University of Pavia

Robolab Robotics Laboratory

SHM Structural Health Monitoring

DT Digital Twin

BIM Building Information Modeling

OSP Optimal Sensor Placement

WSN Wireless Sensor Network

VBI Vehicle–Bridge Interaction

WIM Weigh-In-Motion

FFT Fast Fourier Transform

FDD Frequency Domain Decomposition

SVD Singular Value Decomposition

IOT Internet-of-Things

VR Virtual Reality

AR Augmented Reality

WebGL Web Graphics Library

GIS Geographic Information System

VTk Visualization Toolkit

Chapter 1

Introduction

Bridges are among the most important components of modern transportation infrastructure. They provide essential connections across natural or artificial obstacles and ensure the continuous movement of people, goods, and services. Beyond their functional role, bridges exert a profound influence on social and economic dimensions: they facilitate regional development, enable trade, and contribute to social integration (see figure 1.1).

At the same time, bridges are exposed throughout their service life to a variety of environmental and operational loads, including vehicular traffic, wind, temperature fluctuations, and seismic activity. These factors generate vibrations and stresses that can lead to material fatigue, gradual deterioration, or even sudden damage. Maintaining the structural integrity of bridges is therefore critical to ensuring safety and preventing disruptions in the transportation network, highlighting the need for continuous monitoring and preventive maintenance.

Traditional inspection methods, primarily based on scheduled visual assessments and manual measurements, while useful for identifying visible defects, often fail to detect early-stage anomalies. Moreover, these methods are labor-intensive and typically require temporary traffic restrictions, which means that vital data related to transient or dynamic events occurring between inspections may be lost (see figure 1.2).

1.1 Overview of Structural Health Monitoring (SHM)

Structural Health Monitoring (SHM) is a key area in civil engineering, aimed at continuously assessing the condition of structures and enabling early detection of damage. SHM systems employ networks of sensors installed on the structure to

Figure 1.1: View of the Kigamboni Bridge. Image by Miltonisaya, licensed under CC BY-SA 4.0.

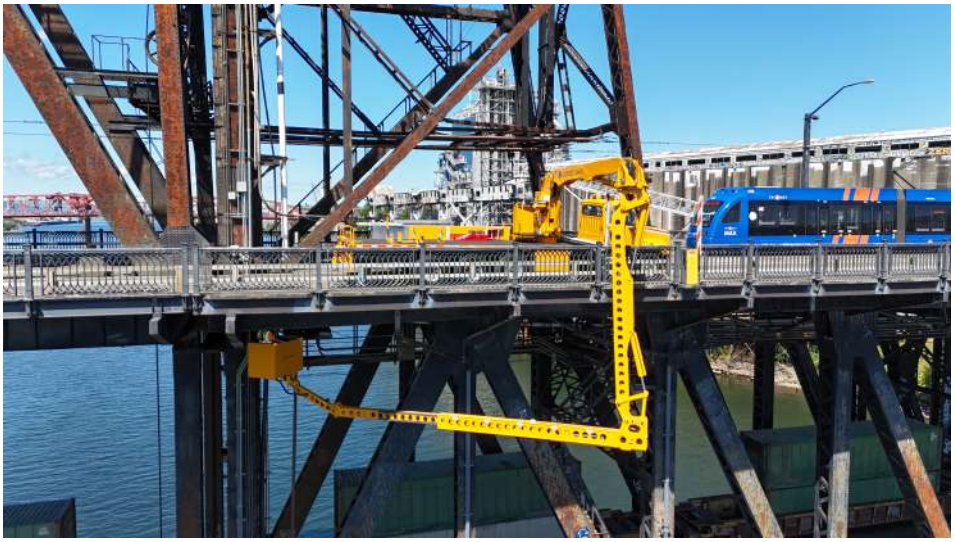

Figure 1.2: Inspection of the Steel Bridge with a MAX train crossing. Image by Oregon Department of Transportation, licensed under CC BY 4.0.

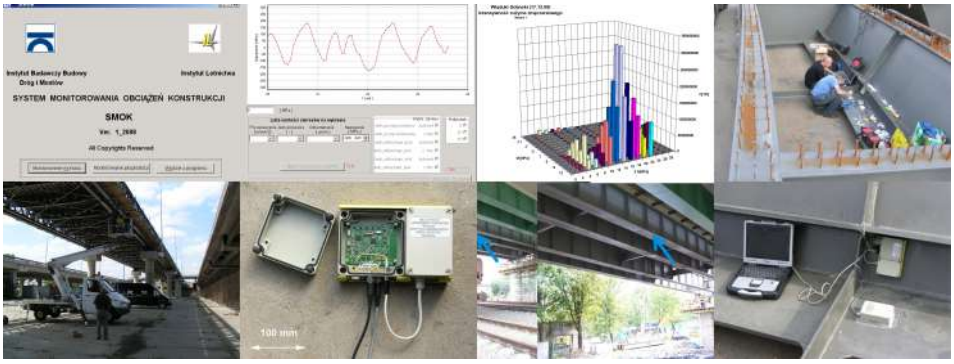

Figure 1.3: Autonomous system for continuous monitoring of structure loads (SMOK). Image by Ph.D. Eng. Marek Dębski, Jan Kaźmierski, Daniel Dębski, Edward Babiasz & Piotr Olaszek, licensed under CC BY-SA 3.0.

record real-time data on how a bridge responds to operational and environmental loads. These datasets make it possible to extract sensitive dynamic indicators such as natural frequencies, vibration mode shapes, and damping ratios. Changes in these indicators can signal the presence of damage, gradual deterioration, or shifts in structural behavior.

1.2 Problem Statement

While the sensor network on the bridges provides high-quality data, raw numerical outputs are not immediately useful for operational decision-making. Engineers require tools that can convert large datasets into meaningful visual information, allowing quick identification of unusual patterns and potential problems. Current methods for reviewing data often involve static plots generated offline, which limit the ability to monitor structural behavior in real-time (see figure 1.3).

Moreover, the absence of a flexible, user-friendly visualization platform makes it difficult to explore the dynamic responses of the bridge under different load and environmental conditions. There is therefore a need for a system that can process and visualize data in real-time, provide clear and intuitive graphical outputs, and integrate seamlessly with the existing SHM infrastructure.

1.3 Research Significance

The development of an effective visualization system for the Bridge data offers several advantages. Real-time visualization allows for quick detection of abnormal

vibration patterns, supporting preventive maintenance and early anomaly identification. Additionally, it enables engineers to monitor the bridge's dynamic behavior without waiting for lengthy post-processing, providing valuable operational decision support. It also contributes to the broader goals of digitalizing infrastructure management within Smart City initiatives by facilitating integration with smart infrastructure. Finally, the visualization and analysis of historical data help identify redundant or low-value sensors in the network, enabling future optimization of the SHM configuration, which can reduce maintenance costs and simplify system management without sacrificing data quality.

1.4 Aim and Scope

This thesis aims to design and implement a real-time visualization tool for data collected from the sensor network. The tool will be developed using the Python programming language and the Pygame library, chosen for its capability to render interactive graphics efficiently. The scope of this research is limited to the visualization process, which includes reading live or recorded vibration data from the SHM system, processing signals to improve clarity (noise filtering and normalization), and displaying interactive graphical outputs representing sensor readings over time and space. It should be noted that this research does not address the physical installation of sensors or the development of data acquisition hardware.

1.5 Methodology Overview

The methodology adopted in this research consists of several stages. First, a review of existing solutions is conducted to analyze current SHM visualization techniques and to identify existing gaps. Next, a modular visualization framework is then designed in Python, with a focus on enabling real-time rendering capabilities. This is followed by the implementation phase, in which modules for data reading, processing, and display are integrated using Pygame. Finally, the developed tool is tested and evaluated with different datasets, with particular attention to responsiveness, accuracy, and usability.

1.6 Contributions

The main contributions of this thesis can be summarized as follows. It introduces a lightweight and scalable visualization system for real-time bridge data, designed

to be applicable across a wide range of bridge infrastructure monitoring projects. The system has been developed with flexibility and adaptability in mind, allowing for extension and deployment in other bridges and similar projects.

As a case study, the proposed framework has been directly integrated with real SHM datasets from the Po Bridge, thereby demonstrating its practical applicability under real-world conditions.

1.7 Thesis Structure

The remainder of this document is organized as follows:

- Chapter 2: Reviews the state of the art in SHM systems and data visualization.
- Chapter 3: Describes the tools, software libraries, and sensor technologies used in this research.
- Chapter 4: Details the implementation of the visualization tool, including data handling and rendering techniques.
- Chapter 5: Presents the results of system testing and discusses the findings.
- chapter 6: Summarizes conclusions and suggests directions for future work.

This structure provides a coherent path for the reader, moving from theoretical foundations to the practical realization of the system, and finally to an assessment of its performance and its broader implications.

Chapter 2

State of the art

In this chapter, the theoretical and practical foundations of bridge Structural Health Monitoring (SHM) are first reviewed, including its objectives, measurable quantities, and the role of dynamic responses in damage detection.

The deployment of sensors and the operational and analytical challenges of bridge monitoring such as noise, environmental effects, data volume, and near–real-time requirements are examined. Next, the influence of traffic loads and vehicle motion on the bridge response is described, with a focus on moving-load models and vehicle–bridge interaction, as well as the distinction between offline and real-time analysis.

After that, trends and patterns in SHM data visualization – from classical diagrams to Building Information Modeling (BIM)/DT approaches and immersive environments – are introduced, and finally, the existing tools and platforms for visualizing bridge DTs are reviewed; their limitations and gaps are then identified, and based on this, the research proposes a visualization approach for representing vehicle–bridge interaction.

2.1 SHM of Bridges

SHM refers to the process of observing, measuring, and analyzing the response of a structure over time with the aim of assessing its health condition and enabling early damage detection.

In the classical definition provided by *Farrar and Worden* [1], SHM is founded on the fundamental assumption that any damage or change in the physical state of a structure leads to alterations in its dynamic characteristics or other measurable responses, and that these changes can be identified through recorded data.

Accordingly, SHM is presented as a data-driven framework for evaluating structural performance under real operating conditions.

Bridges, as one of the most critical components of transportation infrastructure, represent one of the most significant applications of SHM systems.

These structures are continuously subjected to variable traffic loads, environmental conditions, temperature fluctuations, and time-dependent deterioration.

The primary objective of SHM in bridges is to enhance public safety through early damage detection, support predictive maintenance strategies, and reduce the life-cycle costs of the structure.

Recent review studies indicate that the use of SHM enables a transition from traditional visual inspection methods to approaches based on real operational data, thereby playing an important role in optimizing engineering decision-making [2], as shown in Figure 2.1.

To achieve these objectives, bridge SHM systems rely on measuring a set of physical quantities that describe the structural behavior from different perspectives. One of the most important of these quantities is the vibrational response of the structure. Bridge vibrations, which are primarily generated by vehicle passage, wind, or environmental excitations, provide valuable information about the structure's dynamic characteristics, including natural frequencies, mode shapes, and damping ratios. Changes in these parameters are often considered indicators of damage or stiffness reduction, and for this reason, vibration-based methods are regarded as one of the most common approaches in bridge SHM [1, 3].

Strain is another key quantity in the SHM of bridges. Measuring strain enables the assessment of stress distribution and load transfer across different structural components, and is particularly important for evaluating the performance of critical elements under actual traffic loads. Long-term strain monitoring can be used to investigate fatigue behavior, the effects of overloads, and the detection of localized damage, and has been employed in many SHM systems based on fiber-optic sensors or electrical strain gauges [4, 2].

Acceleration, which is typically measured by accelerometers, is considered one of the most common types of data in bridge SHM systems. Acceleration data play a fundamental role in dynamic analyses and in identifying the modal characteristics of the structure, and they are widely used to examine the bridge response to moving loads and transient events. Due to their ease of installation and high sensitivity to dynamic variations, accelerometers constitute the core of many bridge monitoring systems [5].

Figure 2.1: Bridge with vehicular traffic, Source: (`https://pixabay.com/photos/architecture-bridge-buildings-cars-1845278/`), free license).

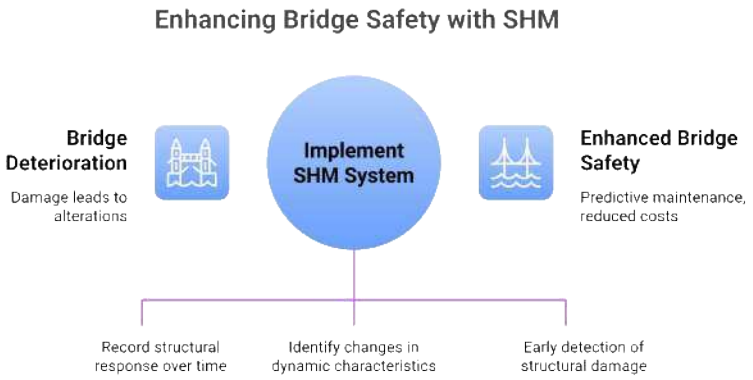

Figure 2.2: Implementation of a Structural Health Monitoring system as a core element in bridge safety management.

In addition to dynamic quantities, displacement measurement also holds particular importance for evaluating both the global and local behavior of a bridge. Displacements reflect the deformation of the structure under traffic loads, long-term effects such as creep and settlement, or reductions in structural stiffness. Although direct displacement measurement is more challenging than acceleration or strain monitoring, recent advances in sensing technologies have enabled the broader use of displacement data in bridge SHM, especially for long-span bridges and for assessing serviceability criteria [6].

In addition to mechanical responses, many bridge SHM systems also include the measurement of environmental parameters. Temperature is one of the most important of these parameters, as thermal variations can have a significant influence on the structural response and the recorded data, and in some cases, temperature-induced changes may be mistaken for damage effects. Therefore, recording and accounting for temperature data is essential for the correct interpretation of the structural response and for increasing the reliability of the analyses [1, 2]. In some systems, parameters such as humidity, wind speed, rotation angles, and crack openings are also measured to obtain a more comprehensive picture of the bridge's health condition [7], as illustrated in Figure 2.2.

2.2 Deployment of Sensors and Key Challenges in Bridge SHM

In bridge SHM systems, the arrangement of sensors and the management of operational and data-related challenges are considered key factors in the system's effectiveness and reliability. Designing the sensor layout requires balancing analytical criteria, practical constraints, and communication requirements, while simultaneously addressing challenges such as data noise, environmental effects, and the large volume of measured information. The following sections review the common principles of sensor deployment and the challenges associated with SHM.

2.2.1 Sensor Deployment on Bridges

In bridge SHM systems, the design of the sensor layout is typically framed as an information-driven problem, meaning that the objective is to obtain the most reliable information about the structural behavior using a limited number of sensors and within constrained costs. Within this framework, the specialized literature introduces Optimal Sensor Placement (OSP) as an independent topic, in which sensor locations are selected based on criteria such as improving modal identifiability, increasing the informational content of the data, and reducing redundancy among measurement channels. Review studies on OSP methods show that a family of criteria and techniques—including those based on modal independence, modal energy, information entropy, and mode-shape correlation metrics—are employed for this purpose and have been widely applied in structures such as bridges [8, 9, 10].

In practice, the design of sensor installation locations is typically carried out based on structural and dynamic analyses. Case studies on bridge monitoring show that modal analysis and numerical modeling—particularly finite element modeling—are used to understand the distribution of dynamic responses, interpret the measured data, and evaluate the sensitivity of structural behavior to environmental variations or structural changes. Therefore, sensor layouts are often selected in a manner that ensures the reliable recording of the responses required for modal analysis and for monitoring dynamic changes in the structure [11].

From the perspective of measurement type, the literature on bridge monitoring projects shows that dynamic sensors (such as accelerometers) are typically used to record the global dynamic response of the structure and to extract modal characteristics, whereas local sensors (such as strain gauges and certain displacement sensors) are primarily employed to monitor the local behavior of members and to assess the

effects of loading in critical regions. This functional distinction does not produce a *"fixed location-based rule"* (for example, a mandatory installation at mid-span), but it is reported in real-world projects and bridge-related review studies as the common rationale guiding the design of measurement systems [12, 13].

In addition to analytical criteria, practical constraints also play a decisive role. Limitations related to access and installation safety, the feasibility of cabling or the quality of wireless links, protection against moisture, corrosion, or impact, as well as the possibility of periodic maintenance and repair, all lead to the final sensor layout typically being a compromise between the *"theoretical optimal design"* and *"practical feasibility"*. Experiences with deploying wireless systems on large structures further show that sensor location must be considered simultaneously with network-related factors, including communication coverage, data transmission paths, and link stability [14, 15, 16].

2.2.2 Challenges in Bridge SHM

One of the main challenges in bridge monitoring is the noise and variability of data in real environments. Sensor data, in addition to instrumentation noise, are influenced by environmental factors – particularly temperature – and experimental studies on bridge monitoring have shown that temperature variations can cause significant changes in modal parameters such as natural frequencies. As a result, without modeling or compensating for environmental effects, distinguishing changes caused by damage from those induced by environmental conditions becomes difficult. This issue has been explicitly reported in studies on bridge monitoring under environmental variability as well as in research related to thermal effects on modal parameters [17, 18, 19].

Another important challenge is the *large volume of data*. In dynamic monitoring, high sampling rates and multiple channels generate *massive time-series datasets*; this issue becomes even more pronounced in sensor-network-based systems—particularly wireless ones—because the data must not only be stored but also transmitted and managed. Comprehensive reviews on Wireless Sensor Network (WSN)-based SHM emphasize that the *high data generation rate* and *network scalability* have made data-architecture design, data compression/reduction, and near-source processing essential requirements. More recent reviews on advances in WSNs for SHM also report this theme as a dominant trend of the past decade [20, 21].

In the end, achieving near-real-time data is confronted not only with the large

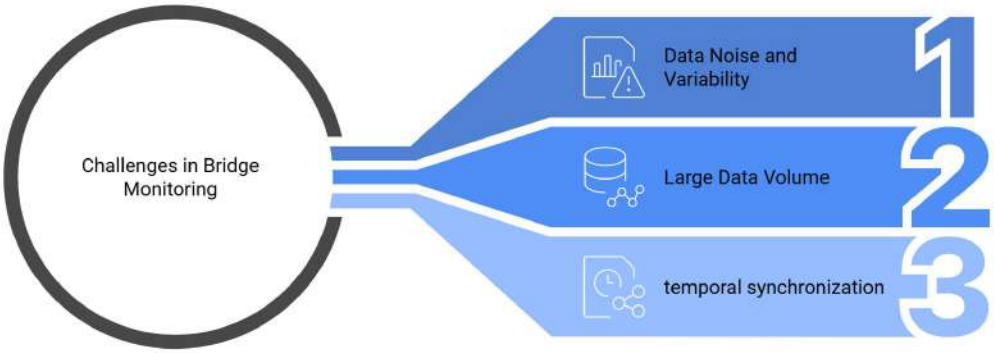

Figure 2.3: Navigating challenges in bridge monitoring.

volume of data but also with communication limitations and temporal synchronization requirements. For multi-channel dynamic analyses, precise synchronization among measurement nodes and communication reliability are critically important, and studies on wireless SHM systems identify time synchronization, reliability, scalability, and energy/bandwidth constraints as key limitations. Systematic reviews specifically focused on bridges within the WSN domain likewise consider these factors—together with the type of sensors and the type of response *(global/local, static/dynamic)*—as determining elements in the design of continuous monitoring systems [22, 16], as illustrated in fig. 2.3.

.

2.3 Traffic-Induced Loading and Its Influence on Bridge Response

Traffic loads generated by passing vehicles are among the primary sources of dynamic excitation in highway bridges and strongly affect the structural response. Due to the variable nature of these loads and their dependence on vehicle characteristics, speed, and roadway conditions, accurate modeling of traffic effects is essential for predicting the actual behavior of the bridge.

2.3.1 Modeling of Moving Loads and Vehicle–Bridge Interaction

The excitation generated by passing vehicles is one of the most significant sources of dynamic loading in highway bridges and is typically represented in the literature through two main modeling families: the moving-load model and the vehicle–bridge interaction model. In the moving-load model, the vehicle is assumed to be a set of concentrated forces (usually axle loads) that travel along the deck at a specified speed, and the structural response is obtained by solving the bridge's equations of motion under these spatially and temporally varying loads. Due to its simplicity and computational efficiency, this approach is widely used in dynamic analyses and parametric studies and is extensively applied in engineering assessments and in evaluating the sensitivity of bridge response to traffic parameters [23].

However, many studies have shown that under real conditions, the contact force applied to the bridge is neither constant nor known in advance, because vehicle motion and bridge vibration mutually influence each other. In Vehicle–Bridge Interaction (VBI) models, the vehicle is typically represented as a dynamic system (for example, a mass–spring–damper model or a multi-degree-of-freedom model), while the bridge and roadway surface are modeled using three-dimensional finite element representations. This framework becomes particularly important when road surface irregularities, vehicle suspension characteristics, and speed variations play a significant role in the dynamic response [24].

2.3.2 Effects of Vehicle Type, Speed, Axle Configuration, and Roadway Conditions

The bridge response under traffic passage does not depend solely on the *weight* of the vehicle; rather, a combination of vehicle and passage characteristics governs the intensity of the dynamic response. State-of-the-art reviews and code-oriented studies indicate that the dynamic amplification or impact factor depends on parameters such as vehicle speed, axle spacing and number of axles, roadway roughness, and the dynamic properties of the vehicle, and that it can vary significantly across different conditions [25].

Among these factors, vehicle *speed* plays a key role, as speed not only alters the duration of load application but also affects the likelihood of the excitation approaching the natural frequencies of the bridge and thereby increasing or decreasing dynamic amplification. Studies based on the evaluation of dynamic axle loads show that speed, axle spacing, and roadway roughness can modify the magnitude of the

dynamic load transmitted to the bridge and therefore must be considered when predicting the dynamic response [26].

Studies that place particular emphasis on *heavy or multi-axle vehicles* also report that the axle configuration (including the number of axles and their spacing) can generate considerable excitation even at relatively low speeds, and that *critical conditions* may arise depending on the axle arrangement and the characteristics of the bridge [27].

Vehicle type also influences the response from two perspectives. In the moving-load model, differences in vehicle type are introduced primarily through variations in axle loads and axle configurations, whereas in VBI models, differences in vehicle type also appear in the dynamic parameters (such as sprung/unsprung masses, suspension stiffness, and damping), which can alter the predicted response compared to the simplified model. Moreover, in real traffic, multiple vehicles typically pass simultaneously and speed variations (acceleration/deceleration) occur, making single-vehicle scenarios insufficient to represent all operational conditions and creating the need for broader scenario generation or the use of field data (such as Weigh-In-Motion (WIM) data or traffic datasets) [25, 27, 28].

2.4 Offline vs. Real-Time Data Analysis in Bridge Monitoring

In studies related to bridge health monitoring and the analysis of bridge response under traffic loads, data-analysis methods can generally be divided into two categories: offline analysis and real-time (or near-real-time) analysis. These two approaches differ fundamentally not only in the timing of data processing but also in system architecture, data-management strategies, and their intended application objectives.

In offline analysis, data obtained from field measurements or numerical simulations are first stored and then analyzed in a post-processing stage. In this approach, there is no strict time constraint for extracting results, and the primary focus is on analytical accuracy, comparison of different loading scenarios, identification of dynamic characteristics, and calibration of numerical models. The results are typically presented in the form of time- or frequency-domain response plots, dynamic indicators, and engineering parameters. Although this analysis method is highly suitable for parametric and research-oriented studies, its non-instantaneous nature does not allow real-time reaction to, or interpretation of, the structural behavior

during operation [29].

In contrast, in the real-time analysis approach used in structural health monitoring systems, the objective is for the measured data to be continuously received, processed, and analyzed such that data acquisition, processing, and decision-making all occur in real time. As emphasized in [30], if data analysis is not performed simultaneously with data acquisition, *there is no justification for continuous monitoring*, and the system effectively becomes indistinguishable from triggered recordings.

To achieve this capability, data processing is performed using sliding time windows (running windows). These windows allow structural dynamic features—including natural frequencies, damping, and other health indicators—to be computed continuously and updated over time. The article explains that this method ensures that the analysis results consistently reflect the instantaneous condition of the structure and enable both sudden and gradual changes to be tracked with precision. These characteristics make real-time analysis an essential option in SHM systems for continuous monitoring, rapid detection of behavioral changes, and support of operational decision-making.

In recent years, the development of intelligent monitoring platforms and bridge DTs has enabled real-time analysis to move beyond purely numerical processing and evolve into an integrated framework encompassing data acquisition, processing, analysis, and result delivery. Nevertheless, even within these systems, many in-depth dynamic analyses and detailed traffic-load evaluations are still performed offline, while real-time analysis remains primarily focused on extracting summary indicators and presenting the current condition of the structure. This separation of roles indicates that offline and real-time analysis are not considered substitutes for one another, but rather complementary layers within bridge health-monitoring systems [31].

2.5 Sensor Data Analysis Methods in Bridge Health Monitoring

After data are collected from the sensors installed on the bridge, the data-analysis stage plays a fundamental role in interpreting the structural behavior and assessing its health condition. In the structural health-monitoring literature, a wide range of analytical methods have been developed for processing sensor data, with the primary objective of extracting meaningful features from the structural response, identifying abnormal changes, and detecting damage. Before any data presentation

or visualization, these methods rely mainly on numerical and statistical analyses and can generally be categorized into time-domain analyses, frequency-domain analyses, damage-detection methods, and machine-learning-based approaches.

2.5.1 Time-domain analysis

One of the fundamental approaches in the health monitoring of civil structures is time-domain data analysis. In this approach, the measured responses from sensors are used as raw inputs to evaluate the structural behavior under environmental loads, vehicle passages, or transient events. This type of time-domain examination is typically considered the first stage in the data-analysis process and can provide preliminary information about general trends or abnormal variations in structural performance. However, accurate interpretation of these variations requires an appropriate understanding of the structure's dynamic behavior and the environmental factors influencing its response—an aspect consistently emphasized in SHM studies [32].

2.5.2 Frequency-Domain Analysis

Alongside time-domain methods, frequency-domain analysis is one of the principal approaches for structural modal identification. In this approach, the recorded signals of the structure's ambient response are processed using tools such as the Fast Fourier Transform (FFT) and more advanced modal-analysis techniques to extract the structural dynamic parameters, including natural frequencies, mode shapes, and damping ratios. Classical frequency-domain methods, such as Peak Picking, perform adequately for well-separated modes but face limitations when dealing with closely spaced modes or noisy data. The Frequency Domain Decomposition (FDD) technique, introduced in this article, enables accurate modal separation and parameter extraction by decomposing the power spectral density matrix and applying Singular Value Decomposition (SVD), even in the presence of noise and closely spaced modes. This approach is considered one of the efficient and user-friendly methods for modal identification of structures under ambient excitation [33].

2.5.3 Modal Analysis

Vibration-based damage-detection methods are developed on the basis of features such as changes in natural frequencies, mode shapes, and damping, since these parameters are directly governed by the physical properties of the structure, includ-

ing stiffness, mass, and damping. In these approaches, the current condition of the structure is compared with a healthy reference state to identify the presence of damage and, in some cases, its location. Comprehensive reviews indicate that although these methods perform well under controlled conditions, they face challenges in real structures due to the low sensitivity of certain modal parameters to damage, the significant influence of environmental and operational variations, and measurement noise. These factors make the interpretation of results dependent on engineering experience and judgment, and they render the practical application of such methods more complex than what is typically observed in laboratory environments [34].

2.5.4 Machine-Learning-Based Approaches

In recent years, data-driven approaches and machine-learning algorithms have gained significant attention as effective tools for supporting the structural health-monitoring process, including in bridge applications. These methods, grounded in the concepts of *pattern recognition* and *statistical learning*, aim to extract damage-sensitive features from the large and complex vibration datasets that are common in long-term monitoring systems and to identify behavioral changes in the structure.

Within this framework, unsupervised learning is typically employed for anomaly detection or deviation from the healthy state, whereas supervised learning is more commonly used for tasks such as classification or estimation of damage severity. Despite their strong capability in analyzing high-dimensional data, the outputs of these methods are often presented in the form of statistical features, divergence measures, or decision values. Therefore, as noted in the related literature, the direct interpretation of these results in terms of the physical behavior of the structure is not always straightforward for non-expert users and requires additional analysis or appropriate decision-making frameworks [35, 36].

2.6 Data Visualization in SHM and Bridge Monitoring

Within modern SHM approaches—particularly in DT-based systems and real-time monitoring—there is a strong emphasis on the idea that raw data acquire practical value only when they can be transformed into decision-oriented information. With the increasing volume of sensor data, high sampling rates, and the complexity of structural behavior, the primary challenge is not merely measuring or extracting indicators, but presenting and interpreting this information in a manner that is understandable and actionable for engineers as well as non-expert stakeholders.

As highlighted in the article, linking data to "decision parameters" and enabling rapid identification of abnormal conditions plays a decisive role in risk management, maintenance planning, and response to critical situations. Therefore, appropriate layers of analysis and presentation—such as dashboards, interpretable indicators, and visual tools—constitute an integral component of an effective SHM system [37].

2.6.1 Prevailing Visualization Practices in Bridge SHM

The most common and traditional form of presenting results consists of classical two-dimensional visualizations, which primarily include time-series plots, frequency spectra, modal displays (frequencies and damping), and indicator-based dashboards [38]. This presentation style has long served as the default output in many monitoring systems and, when combined with Internet-of-Things (IOT) infrastructures, has evolved into web-based dashboards [39]. In more recent works, this visualization pattern still persists, but it is typically combined with layers for asset management and health-status representation.

Moving forward, part of the research has shifted toward linking monitoring results to a spatial or structural model so that the data can be mapped to the physical location of the structural member or region. One prominent direction in this area is the integration of SHM with BIM; in this approach, sensor locations, structural assets, and analysis results are displayed within the BIM environment, with the aim of reducing manual errors and providing a more user-friendly setting for maintenance and risk management. Examples of this research line have been reported both in the form of BIM-based frameworks for visualizing bridge-monitoring data and in the automation of SHM processes through *BIMification* [40].

In continuation of this trajectory, the concept of the DT in bridges has emerged, in which the digital model—often based on BIM or numerical modeling—is continuously or periodically updated with sensor data, and the outputs may be presented as state visualization, health-parameter displays, or even near–real-time analyses. Numerous studies in recent years have addressed DTs for bridge monitoring; some focus on data architecture and the integration of sensors with the digital model (for example, in railway bridges equipped with wireless accelerometers and data-driven analytics), while others report examples of real-time DTs at laboratory scale or in the form of multilayered frameworks [41, 42].

A parallel and increasingly human-centered branch of visualization in SHM has moved toward immersive environments and Augmented Reality (AR)/Virtual Reality (VR). In this category, the objective is not merely to display plots or indicators,

but to present results within an interactive three-dimensional environment, enabling the user to observe the damage location or dynamic response directly on the physical structure (in AR) or within its digital representation (in VR). Examples have been reported of real-time AR visualization for vibration and displacement data, as well as the development of VR environments for viewing bridges and SHM data and supporting multi-user collaboration. More recent studies also emphasize precise alignment of the 3D model with the actual structure and the use of AR for documenting and visualizing damage areas in maintenance contexts [43, 44, 45, 46].

2.7 Technological Landscape of Visualization Platforms for Bridge DTs

In recent years, with the expansion of DT and SHM concepts, visualization has become one of the key components of these systems. Visualization in bridge DTs is not limited to displaying the three-dimensional geometry of the structure; rather, it serves as an interface for understanding the structure's dynamic behavior, interpreting sensor data, and analyzing the effects of passing loads such as vehicles. Numerous studies have shown that an appropriate visualization layer can play a significant role in enhancing data interpretability and supporting decision-making in infrastructure management [47].

2.7.1 Infrastructure-Oriented DT Platforms

Several existing platforms have been specifically designed for the development and visualization of DTs for transportation infrastructure, including bridges. These platforms typically enable the integration of three-dimensional models, operational data, and temporal information within a unified visual environment. In the scientific literature, this category of platforms is introduced as practical implementations of the DT concept in the infrastructure domain [48].

Within this framework, industrial platforms such as *Bentley iTwin* have been reported as practical examples of DT implementations for infrastructure assets. Research related to DTs in the construction and infrastructure sectors indicates that such platforms provide capabilities for visualizing the three-dimensional asset model, integrating temporal and operational data, and delivering a visual representation of the asset's condition [49]. According to the technical documentation provided by the developer, *iTwin* enables three-dimensional visualization of infrastructure assets and the integration of operational data within the platform [50]. However, as noted

in the literature, the primary focus of these platforms is often on asset management and providing an overall view of the system's condition, while the visualization of detailed load–structure dynamic interactions typically remains at a conceptual or summarized level [47].

2.7.2 Geospatial and Web-Based Engines for 3D Visualization

A significant category of tools used for visualizing large-scale three-dimensional infrastructure models consists of web-based *geospatial* engines, which are designed for rendering large models and spatial datasets. Among these, *CesiumJS* is considered one of the most widely used open-source engines, providing efficient visualization of 3D models—including BIM models—through its Web Graphics Library (WebGL)-based rendering capabilities. Previous studies have shown that *CesiumJS*, through its support for the 3D Tiles format, offers a suitable platform for displaying building and infrastructure models within a spatial context and enables the overlay of multiple data layers. These features have led to *CesiumJS* being recognized as an effective option for visualizing BIM models and integrating them with *geospatial* data in web-based environments [51].

Similarly, several studies have employed three-dimensional Geographic Information System (GIS) platforms such as *ArcGIS 3D Scene* as part of the visualization and interaction layer in spatial DTs. These tools enable the integrated display of three-dimensional structural models, *geospatial* datasets, and sensor-derived temporal data, making them suitable for analyses in which understanding the spatial relationship of the bridge to the transportation network and surrounding environment is essential [52].

2.7.3 Game Engine Driven Real-Time Visualization

In several advanced research studies and projects, real-time 3D engines such as *Unity and Unreal Engine* have been employed for DT visualization. Originally developed for computer-game production, these engines have attracted attention due to their high real-time rendering performance and animation capabilities, making them suitable for visualizing dynamic phenomena such as vehicle movement on bridges and the structural response under moving loads. Studies indicate that the use of these engines can enable interactive visualization of vehicle passage and structural response within three-dimensional environments, in which simulation or sensor data are transformed into real-time visual animations [53].

2.7.4 Scientific and Data-Driven Visualization Tools

In addition to general three-dimensional platforms, scientific visualization tools also play an important role in visualizing engineering data for bridges. Software packages such as *Visualization Toolkit (VTk)* and *ParaView* are widely used for displaying numerical analysis results, deformation fields, and vibration data. These tools provide capabilities for visualizing scalar and vector fields, animating dynamic structural responses, and processing large-scale datasets, and they are recognized as standard scientific-visualization tools in SHM research [54, 55].

Overall, the available tools and platforms for visualizing bridge DTs span a diverse spectrum, ranging from infrastructure-oriented industrial platforms to web-based geospatial engines, real-time 3D engines, and scientific visualization environments. Each of these tools offers specific capabilities for representing the bridge model and associated data, and they have been reported across various studies. However, differences exist in how they visualize dynamic vehicle–structure interactions, the degree of flexibility in controlling visualization logic, and the complexity of implementation. These aspects will be critically examined in the next section as part of the discussion on existing limitations and gaps.

2.8 Gaps and Technical Constraints in Bridge DT Visualization Tools

Despite the diversity of existing tools and platforms for visualizing bridge DTs, a review of the research literature and operational systems indicates that these solutions still face limitations and gaps, particularly when the objective is to provide a clear and data-driven representation of the dynamic interaction between vehicle motion and bridge sensor responses. These limitations can be categorized into several main areas.

2.8.1 Limits in Visualizing Moving-Load Effects

A substantial portion of infrastructure-oriented DT platforms focuses on high-level monitoring, overall asset condition visualization, and lifecycle management. Although these systems display the three-dimensional bridge model together with operational data within a unified visual environment, the cause-and-effect relationship between the passage of a specific vehicle and the corresponding sensor-based temporal response is typically not presented in an explicit, direct, and event-driven

manner. Consequently, the user is unable to clearly, immediately, and interpretably observe the structural response associated with a particular event—such as the passage of a vehicle with different weight or speed.

This limitation is also evident in scientific-visualization tools and general 3D platforms, where, despite their strong capabilities in analytical data display or visual effects, user interaction and the direct, time-dependent linkage between vehicle motion and sensor response are not supported in an integrated manner. Overall, this gap makes the simultaneous and comprehensible visualization of both vehicle movement and structural response over time difficult to achieve.

2.8.2 Limited Flexibility in Visualization Logic

Many industrial platforms and even some advanced visualization engines employ closed or semi-closed architectures that limit the researcher's ability to directly manipulate the visualization logic. This becomes a significant challenge particularly in academic studies, where the objective is to test different scenarios, modify the mapping of data to visual elements, or design customized visual representations. Under such conditions, the researcher is often forced to conform to the platform's predefined patterns rather than designing the visualization precisely according to the needs of the study.

2.8.3 Computational and Implementation Constraints

Real-time 3D engines and industrial-gradeDT platforms, despite their high rendering performance and advanced capabilities, are typically associated with high implementation complexity, the need for specialized knowledge in graphics or game-engine development, and significant computational overhead. These characteristics make them challenging to use for developing lightweight, fast, and easily reproducible research prototypes. In many cases, a substantial portion of the research effort is devoted to setting up and maintaining the software infrastructure, rather than improving the visualization concept itself.

Overall, although existing tools and platforms have made significant progress toward visualizing bridge digital twins, a closer examination reveals that achieving a simple, transparent, and controllable representation of the dynamic interaction between vehicle motion and bridge sensor responses remains challenging. These limitations stem primarily from platform complexity, a predominant focus on asset management rather than conceptual visualization, and restricted flexibility in designing data-driven and event-oriented visual representations. These gaps pro-

vide the main motivation for exploring alternative and lighter-weight approaches to visualizing vehicle–bridge interaction, which will be addressed in the following section.

Chapter 3

Tools and Frameworks

This section provides an overview of the software tools and development environment employed in the project, highlighting the technologies that supported the implementation, analysis, and real-time visualization of structural health monitoring data.

3.1 Software Tools and Development Environment

In this project, the primary objective was the development of a software tool for processing and visualizing data obtained from structural health monitoring systems. Since the goal was to create an application usable for any type of similar data, the selection of tools and programming languages was made in a way that ensures both flexibility and simplicity, while also enabling fast and accurate development and the ability to handle diverse and large datasets.

3.1.1 Program Core and Development Environment

The core of the application was implemented in Python, a language that—due to its simplicity, readability, large user community, and the availability of specialized libraries for data analysis and visualization—is considered one of the best options for such projects. All components of the application—from reading data and performing statistical computations to building the graphical interface and communicating with the data source—were implemented within a unified Python environment.

For code development, environment VS Code was used (see figure 3.1), providing features such as code autocompletion, debugging, and library management. Library management was handled through the standard tool `pip`, which enabled the installation of appropriate versions and prevented potential incompatibilities.

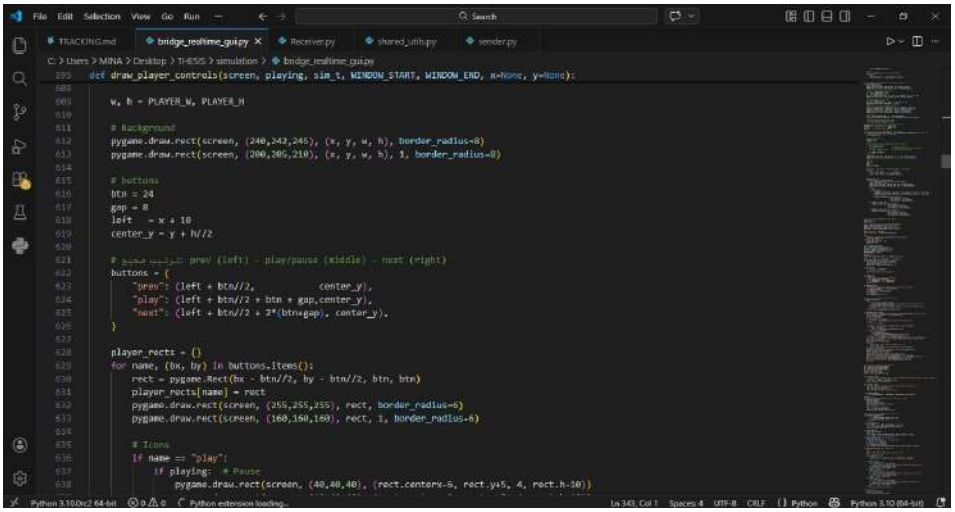

Figure 3.1: The development environment of the project in Visual Studio Code.

3.1.2 Libraries Used

For processing input files and managing data, the `Pandas` and `NumPy` libraries were employed. Pandas served as the main tool for reading CSV and Excel files, cleaning data, and constructing time series, while NumPy supported numerical computations and array operations. For statistical modeling and analyzing data distributions, functions from `SciPy` were used. Additionally, for examining trends and simple relationships between variables, the linear regression model from the `Scikit-Learn` library was applied.

3.1.3 Data Visualization and Graphical Interface

For data visualization and generating analytical plots, the `Matplotlib` library was utilized. A more significant part of the project was the interactive graphical interface, which was built using `Pygame`. This interface enables the application to display input data in real time; for example, the status of sensors, vibration or bending intensity, vehicle passages, and many other components are presented within a simple and comprehensible graphical environment. For date and time range selection, simple `Tkinter`-based tools were also used.

3.2 Data Structure and Input File Formats

One of the important components of this project is the method of receiving and interpreting the data obtained from structural health monitoring systems. Since

the application must be able to work with different datasets, the structure of the input files has been selected in a way that ensures both high readability and the possibility of automated processing. In this project, the data consist of several types of standard files, each containing information related to a specific aspect of the structural behavior.

3.2.1 CSV Files

A major portion of the data is provided in CSV files, a format that is highly suitable for storing temporal and numerical data. These files typically include data related to vibration, bending, torsion, and temperature. Each file has a simple structure: one column for the timestamp and additional columns for the values measured by each sensor. This simplicity allows the application to read the data and convert them into time series without requiring complex preprocessing.

3.2.2 Excel Files

Alongside the CSV files, some supplementary information—such as data related to vehicle passages or equipment identification codes—is provided in Excel format. This format is particularly suitable for data with a table-based structure, as it enables more precise categorization and organization of information. The application reads these files in the same manner as CSV files and uses them to support the analysis or visualization process.

3.2.3 JSON Files

To display the spatial positions of sensors on the bridge or any similar structure, a JSON file is used in which the two-dimensional coordinates of each sensor are recorded. This file plays an important role in the graphical component of the application, as it determines the location at which each sensor should be displayed in the user interface. The simplicity and flexibility of JSON make it possible to define sensor layouts for other structures easily, without modifying the core code.

3.2.4 System Integration

The combination of these formats enables the application to read different types of data in a coordinated manner, convert them into an internal structure, and use them across various parts of the system—from statistical processing to graphical

visualization. This flexible and generalizable structure turns the application into a tool that can be employed for any monitoring project with similar data.

3.3 Communication System and Data Acquisition

To enable the application to receive and display data in real time, a simple yet efficient communication mechanism has been designed. This mechanism operates based on TCP Socket communication and is responsible for the continuous transmission of data from the source (Sender) to the graphical interface and analysis environment (Receiver). The purpose of designing such a structure was to ensure that the application can receive and process any data stream generated by structural health monitoring systems without dependency on specific hardware.

3.3.1 Sender Component

In this process, the Sender component acts as the data transmission point. This component is typically deployed on a device that has direct access to data files, monitoring equipment, or the primary source of information. The Sender reads the data, converts them into JSON format, and transmits them line by line through a TCP connection to the receiving system. The use of JSON ensures high readability and simplifies the alignment of incoming data with the internal structure of the application.

3.3.2 Receiver Component

On the other side, the Receiver component is responsible for receiving and organizing the incoming data. This component runs in a separate thread to ensure that the data acquisition process occurs concurrently and without interrupting the operation of other parts of the application. The received data are immediately parsed and stored in internal structures so that the application can simultaneously use them across different sections, such as charts, numerical indicators, or the graphical interface.

3.3.3 Advantages of the Communication System

This communication structure offers several important advantages. First, the application can continuously receive data without the need for manual or periodic file loading. Second, the communication mechanism is designed in a way that allows it to be easily reused for other structures, projects, or data sources. Finally, the

simplicity of this method ensures that even under conditions involving high data volume or fast transmission rates, the application can continue operating without issues.

3.3.4 Conclusion

Overall, the Sender/Receiver system functions as the backbone of data acquisition in this project and plays a crucial role in enabling real-time and live data visualization.

3.4 Hardware Environment for Running the Application

The software component of this project was executed on a personal laptop, which served as the primary environment for development, testing, and running the application. The system used was an `HP ProBook 4540s` laptop which, despite being equipped with mid-range hardware, was able to meet the project's requirements effectively.

3.4.1 Hardware Specifications

This system, equipped with a penta-core Intel-series processor, sufficient RAM for processing large datasets, and an integrated graphics card, provided an adequate platform for running the graphical interface and data-processing tasks simultaneously.

3.4.2 Performance and Stability

Throughout the development of the application and during various test executions, this laptop successfully supported time-series processing, chart rendering, and the operation of the live graphical interface. The stable performance of the system demonstrated that the application is designed in a way that does not require specialized or high-performance hardware and can run smoothly on a standard machine.

3.4.3 Usability in Different Environments

This characteristic makes the developed tool suitable for use in different environments and on various devices. The ability to run on typical mid-range systems makes the application appropriate for use in most research centers and practical monitoring projects.

Chapter 4

Implementation

In this chapter, the implementation process of the real-time bridge monitoring system is presented. The primary objective of this system is to establish an integrated framework for the acquisition, transmission, processing, and visualization of data collected from the sensors installed on the bridge, enabling instantaneous monitoring of the structural response during vehicle passage. Vibration, bending, torsion, and temperature data are recorded by the sensors, undergo preliminary processing within the transmitter module, and are subsequently transferred to the receiver module through the network. The receiver then performs data synchronization and imports the processed information into a dedicated software environment for real-time analysis and visualization.

To ensure reliable data transmission, accurate synchronization, and user-friendly visualization, the system architecture is organized into multiple layers, including sensing, preliminary processing, data transmission, receiver-side processing, and visualization. The subsequent sections describe the overall architecture of the system, followed by the detailed design of the transmitter, router, receiver, and visualization environment, thereby providing a comprehensive understanding of the implementation procedure and the system's potential for future expansion.

4.1 Implementation Architecture

The system architecture is developed with the aim of efficiently managing data generated at high volume and high frequency. The layered structure assigns a distinct functional role to each component, ensuring that all elements operate cohesively within a unified workflow, as illustrated in Figure 4.1.

Within this architecture, the sensors initially record the structural response

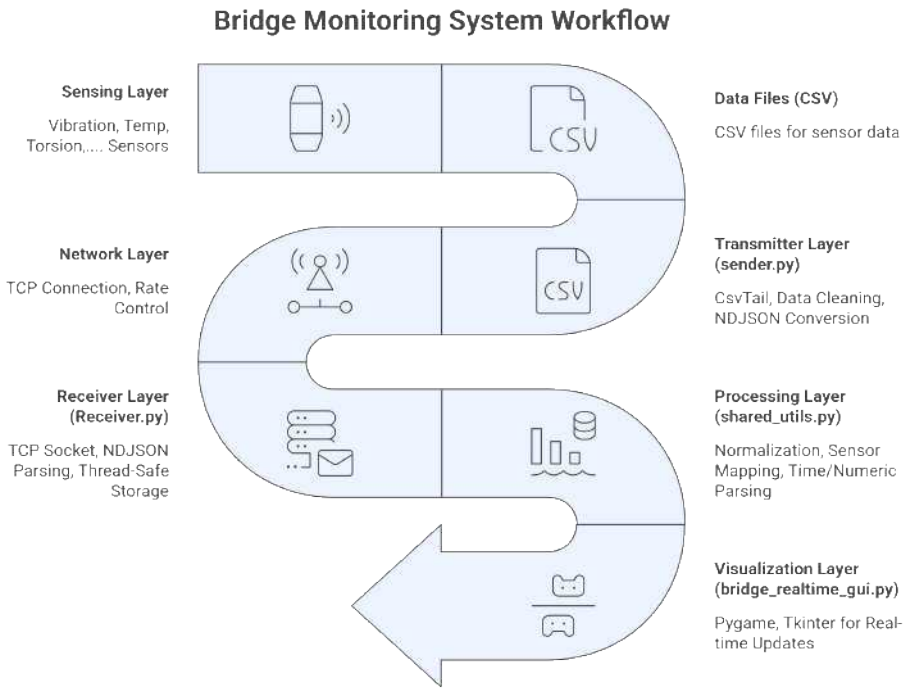

Figure 4.1: Bridge Monitoring System Workflow illustrating sensing, transmission, reception, processing, and visualization layers.

data. The transmitter module then retrieves the newly generated data from the input files and, after converting them into a standardized format, forwards them to the central system. On the receiving side, the receiver module obtains the data and applies temporal organization. Subsequently, the graphical interface presents the synchronized data for real-time observation and analysis.

The main components of the system include vibration, bending, torsion, and temperature sensors, the transmitter module, the receiver module, the graphical visualization environment, and a set of supporting functions.

- **Section 4.1 – Transmitter Module (sender.py):** This module monitors the input files and, using the mapping defined in the `CODESENSOR.xlsx` file, extracts the most recent valid data and transmits them in `NDJSON` format.
- **Section 4.2 – Receiver Module (Receiver.py):** This module processes the incoming data and stores the most recent valid value for each sensor.
- **Section 4.3 – Auxiliary Functions (shared_utils.py):** This module performs tasks such as data cleaning, time extraction, and sensor mapping.
- **Section 4.4 – Graphical Environment (bridge_realtime_gui.py):** This environment visualizes the organized data through charts, alerts, and a vehicle-passage simulation.

4.2 Transmitter Module (senderṗy)

The transmitter module is responsible for receiving raw data from various sources in real time, performing initial processing and synchronization, generating a unified message, and transmitting it to the receiver module through a TCP connection. This component represents the starting point of the real-time data flow within the system.

4.2.1 Configuration and Data Sources

The transmitter module is designed to load a set of configuration parameters at startup, ensuring that the system's behavior with respect to various data sources and operational conditions remains controllable and reproducible. In this module, the paths to the input files are defined individually, and the files are named based on the current date. This mechanism allows the system to automatically connect to the new files generated each day, eliminating the need for manual path adjustments.

One of the key components of the configuration layer is the `CODESENSOR.xlsx` file, which specifies the mapping between data columns and sensor identifiers (AICD). This structure enables the raw data from each file to be extracted in a unified and standardized format corresponding to the defined set of sensors. In addition, the data transmission rate `(SEND_EVERY_MS)` and the server-related settings—including host address and port number—are also defined in this section. These parameters determine the update frequency of the data stream and the manner in which communication with the receiver system is established.

4.2.2 Incremental File Streaming

One of the fundamental requirements in real-time structural data processing is the ability to continuously receive and analyze newly generated values without reloading entire files at each step. In the transmitter module, this requirement is fulfilled through an incremental streaming mechanism. In this approach, the data files are opened only once at startup, and during each processing loop the system reads and processes solely the newly appended lines.

This mechanism is implemented using a dedicated class that operates similarly to the tail `-f` utility in Linux systems; that is, the file pointer is consistently maintained at the end of the last read entry, and only new records are extracted. Such a strategy minimizes the processing load in each cycle and keeps the system latency within the limits required for real-time analysis.

In addition to incremental reading, the data undergo an initial cleaning process at this stage. This includes converting time strings into a standardized format, transforming numerical values into valid floating-point representations, and removing incomplete or invalid records. This preliminary processing ensures data quality and consistency before the integration and synchronization stages.

```
class CsvTail:
    def __init__(self, path):
        self.path = path
        self.fp = None
        self.sep = None
        self.header = None

    def open(self):
        if self.fp:
            try: self.fp.close()
            except: pass
    self.fp = open(self.path, "r", encoding="utf-8", errors="ignore")
    header_line = self.fp.readline()
    if not header_line:
        raise RuntimeError(f"Empty␣file:␣{self.path}")
    self.sep = detect_sep(header_line)
```

```
        self.header = [h.strip() for h in header_line.strip().split(self.sep)]
```

4.2.3 Data Synchronization and Unified Message Construction

After the incremental retrieval of data from multiple files, the next critical stage in the transmitter module involves combining and synchronizing these data streams to construct a coherent and reliable data packet suitable for real-time analysis. Files that follow a synchronized sampling structure are paired based on the exact equality of their timestamps, and only records with matching time values are included in the processing pipeline. This procedure ensures that the structural dynamic information at each moment is formed from valid and temporally aligned data.

In parallel, independently received data streams are handled by continuously storing their most recent valid values. This mechanism enables the system to maintain, at any given moment, a complete representation of the current structural state, vehicle-induced loading, and environmental conditions.

Subsequently, the transmitter module extracts the relevant values for each sensor and maps them to their predefined identifiers (AICD), generating a unified message. This message, accompanied by an accurate timestamp, is constructed in `JSON` format to facilitate straightforward parsing and interpretation by the receiver module.

In summary, this component is responsible for synchronizing, integrating, and standardizing multi-source data to produce a coherent snapshot of the bridge's instantaneous condition—a snapshot that forms the foundation for the graphical and control-oriented analyses in the subsequent stages of the system.

4.2.4 Real-Time Data Transmission

The final stage in the operation of the transmitter module involves the real-time transmission of the unified data to the receiver system. After each standardized message is generated, the system sends the data in `NDJSON` format (line-delimited JSON), a method well-suited for time-dependent data transmission due to its simplicity, readability, and compatibility with streaming-based processing. In this format, each message is transmitted on a separate line, enabling continuous processing without the need to parse complex data structures.

The transmitter module establishes its network communication through a TCP connection. This choice is motivated by TCP's ability to ensure reliable data delivery, preserve message order, and maintain stable connectivity—features that are

essential for engineering applications involving sensitive data. Once the connection with the receiver is established, the module transmits a message in each cycle only if the minimum predefined interval (e.g., 10 milliseconds) has elapsed since the previous transmission. This mechanism, which functions as a form of rate control, prevents data congestion and maintains the stability of the information flow.

Consequently, this component ensures that the multi-source data combined and synchronized in the preceding stages are delivered to the receiver with minimal delay and maximum reliability. This real-time transmission capability forms the backbone of the system's performance in visualization, structural health monitoring, and instantaneous analytical processes.

```
    if last_sent_ns == 0:
        last_sent_ns = ts_ns
    if ts_ns - last_sent_ns < SEND_EVERY_MS * 1_000_000:
        continue
    last_sent_ns = ts_ns

    vib_dict = extract_vib_dict(vr, vib_map)
    def_dict, tor_dict = extract_def_tor_dicts(ir, def_map, tor_map)

    msg = {
        "ts": ts_v.strftime("%Y-%m-%d␣%H:%M:%S.%f")[:-3],
        "ts_ns": ts_ns,
        "vib": vib_dict,
        "def": def_dict,
        "tor": tor_dict,
        "temp": last_temp,
        "veh": last_veh[-20:],
    }
    send_ndjson(conn, msg)
```

4.3 Receiver Module

The receiver module is responsible for handling the continuous stream of data transmitted by the sender module and serves as the communication bridge between the data-processing layer and the graphical environment. By establishing a stable network connection, this module receives real-time messages line by line and, after converting them into a usable data structure, provides the latest updated state to the visualization and analysis components. The simplicity and robustness of its design ensure that the system operates without interruption even under continuous data inflow, consistently delivering the most recent structural state for visualization and analytical tasks.

4.3.1 TCP Connection Establishment

Upon initialization, the receiver module establishes a stable network connection with the transmitter module to ensure uninterrupted reception of real-time data. In this stage, the receiver creates a TCP socket and connects to the transmitter using the specified host address and port number. The choice of the TCP protocol is motivated by its high reliability, orderly packet delivery, and suitability for transmitting sensitive structural data. Once the connection is established, a line-based input stream is created, enabling continuous reception of incoming messages. This layer functions as the primary communication gateway and ensures that all messages sent by the transmitter module are successfully received on the receiver side.

```
def recv_loop(host, port):
    global LATEST
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    f = s.makefile("r", encoding="utf-8")
```

4.3.2 Streaming Receive and JSON Parsing

Once the connection is established, the receiver module enters the data-acquisition loop and begins receiving the messages transmitted by the sender in a streaming manner. The messages are delivered in NDJSON format, meaning that each message is sent as an individual JSON line. The receiver reads each line, removes empty entries, and then converts the content into a structured and processable data object. This approach enables the system to interpret and prepare messages for use immediately upon arrival, without relying on large buffers or batch-processing mechanisms. The simplicity and efficiency of this method make it particularly well-suited for real-time applications, especially for instantaneous visualization of structural data.

```
    for line in f:
        if not line.strip():
            continue
        msg = json.loads(line)
        with LOCK:
            LATEST = msg
```

4.3.3 Thread-Safe Shared State for GUI Access

To ensure that the graphical interface can always display the most up-to-date structural state, the receiver module stores the latest incoming message in a shared variable. Since data reception is performed within a separate thread, a concurrency lock is employed to prevent interference and simultaneous access. Using this lock,

each new message is written to the LATEST variable in a fully thread-safe manner, allowing the graphical interface to read and display the data without interruption or concerns about inconsistency. This simple yet effective design enables a complete separation between the tasks of data reception and real-time visualization, contributing to the overall stability and smooth operation of the system.

```
LATEST = None
LOCK = threading.Lock()
```

4.4 Auxiliary Functions (shared_utils)

The shared_utils module comprises a collection of general-purpose functions used throughout the system to prepare and organize data. These functions focus on cleaning, standardizing, and loading raw files, enabling the system to handle heterogeneous data recorded in various formats. The role of this module is to provide a common and reliable foundation for transforming raw inputs into structured and consistent datasets, ensuring that subsequent stages of processing, analysis, and visualization operate on clean and uniform data.

4.4.1 Normalization and Initial Cleaning

One of the key responsibilities of the auxiliary function set shared_utils is preparing the raw data before they enter the analysis or visualization stages. Data obtained from CSV or Excel files often contain inconsistencies in time formats, numerical representations, textual noise, or missing values. This module automates the normalization and initial cleaning process by providing a collection of dedicated functions.

In the first step, functions such as parse_time_any() and parse_excel_time() handle the unification and interpretation of various time formats. These functions are capable of converting time values—despite differences such as the use of dots, commas, or colons in the millisecond portion—into a valid timestamp format. Subsequently, the conversion of numerical data, including speed, vibration, or temperature, is performed using functions such as to_float() and parse_numeric_column(). These functions remove textual noise, replace commas with dots, and transform non-standard values into analyzable numerical data. This component of the auxiliary functions ensures that the incoming data are transformed into a clean, consistent, and reliable form before entering the modeling and advanced processing

stages. Such normalization provides the foundational layer required for the correct functioning of the system in subsequent modules.

```
def parse_time_any(s: str) -> pd.Timestamp:
    """
    Convert any time format to a Pandas Timestamp.

    Accepted formats:
    - "2025-03-03 10:30:45.123"
    - "2025-03-03 10:30:45:123" (colon instead of dot)
    - "2025-03-03 10:30:45,123" (comma instead of dot)

    Args:
    s: time string

    Returns:
    pd.Timestamp, or NaT if parsing fails
    """
    s = str(s).strip()

    if not s:
        return pd.NaT

    # HH:MM:SS:fff  ->  HH:MM:SS.fff
    if len(s) >= 4 and s[-4] == ":" and s[-3:].isdigit():
        s = s[:-4] + "." + s[-3:]

    # 10:30:45,123  ->  10:30:45.123
    if "," in s:
        s = s.replace(",", ".")

    fmts = [
    "%Y-%m-%d␣%H:%M:%S.%f",
    "%Y/%m/%d␣%H:%M:%S.%f",
    "%d/%m/%Y␣%H:%M:%S.%f",
    "%Y-%m-%d␣%H:%M:%S",
    "%Y/%m/%d␣%H:%M:%S",
    "%d/%m/%Y␣%H:%M:%S",
    ]

    for fmt in fmts:
        try:
            result = pd.to_datetime(s, format=fmt, errors="coerce")
            if not pd.isna(result):
            return result
            except:
            pass
```

4.4.2 File Structure Detection and Smart Loading

Given that the measurement data in the project originate from diverse sources and appear in various formats, an intelligent mechanism for detecting file structure and correctly loading the information is essential. The `shared_utils` module provides a set of functions that automate the identification of file components, delimiter

detection, and consistent data extraction without requiring manual intervention.

In the first step, the `detect_sep()` function examines the initial row of the file to determine the appropriate delimiter from several possible candidates (such as comma, semicolon, tab, or vertical bar). This capability is particularly important when working with files generated by heterogeneous systems or lacking a fixed formatting standard. Subsequently, the comprehensive `read_csv_smart()` function handles the file-loading process. Using the detected delimiter, this function cleans column names, identifies the time column and converts it to a valid timestamp format when necessary, and finally transforms numerical columns into analyzable data through correction and conversion procedures.

The smart-loading mechanism enables CSV and Excel files to be read with minimal dependency on their original structure. This feature is especially valuable in projects where data are recorded or exported by different systems, significantly enhancing the stability and flexibility of the analysis workflow.

```
with open(path, "r", encoding=encoding, errors="ignore") as f:
    header_line = f.readline()
if not header_line.strip():
    return pd.DataFrame()
sep = detect_sep(header_line)

df = pd.read_csv(path, sep=sep, engine="python",
encoding=encoding)

lower_map = {c.lower().strip(): c for c in df.columns}

time_key = next(
(lower_map[c] for c in ["time", "timestamp", "datetime",
"date_time", "t"]
if c in lower_map),
None
)

if parse_time and time_key is not None:
    df = df.rename(columns={time_key: "time"})
    df["time"] = df["time"].apply(parse_time_any)
```

4.4.3 Sensor Mapping and Data Modeling

One of the fundamental requirements in structural data analysis is access to a coherent and reliable representation of each sensor's position and functional role. The `CODESENSOR.xlsx` file typically contains raw information regarding sensor identifiers (AICD), their physical locations on the structure, and the corresponding columns in the measurement files. The `shared_utils` module automates the process of transforming this file into a standardized and usable data model through a

set of dedicated functions.

Functions such as `load_sensor_mapping()` and `load_codesensor_mapping()` read the initial data, extract the column mappings associated with vibration, deflection, and torsion, and link them to unique sensor identifiers. This structure ensures that each data column in the measurement files directly corresponds to a specific sensor. In addition, functions such as `load_distances_from_excel()` and `load_maps_from_excel()` retrieve information such as the distance of each sensor from the bridge reference point or other descriptive parameters, providing an additional layer of contextual metadata.

The outcome of this stage is the construction of an integrated data model in which each sensor is associated with an identifier, a physical location, a functional role, and geometric attributes. This modeling process forms the foundation for subsequent steps, including data analysis, sensor classification, and structural visualization, ultimately enhancing the accuracy and interpretability of the results.

```
    need = {"AICD", "VIB_COL_X", "DEF_COL_Y", "TOR_COL_Z"}
    miss = need - set(df.columns)
    if miss:
        raise ValueError(f"CODESENSOR.xlsx missing columns: {sorted(miss)}")

    vib_col_to_aicd = {}
    def_col_to_aicd = {}
    tor_col_to_aicd = {}
    aicds = []

    for _, r in df.iterrows():
        aicd = str(r["AICD"]).strip().upper()
        if not aicd:
            continue

        aicds.append(aicd)

        vib = str(r.get("VIB_COL_X", "")).strip()
        dfl = str(r.get("DEF_COL_Y", "")).strip()
        tor = str(r.get("TOR_COL_Z", "")).strip()

        if vib:
            vib_col_to_aicd[vib] = aicd
        if dfl:
            def_col_to_aicd[dfl] = aicd
        if tor:
            tor_col_to_aicd[tor] = aicd

        return sorted(set(aicds)), vib_col_to_aicd, def_col_to_aicd, tor_col_to_aicd
```

4.4.4 Data Loading and Resampling

This component of the auxiliary functions is designed to simplify the workflow associated with measurement files. Loading functions such as `load_vibration_csv()`, `load_incl_csv()`, and `load_temperature()` enable direct reading of various files and convert them into well-structured data formats. During this process, the data are standardized in terms of type and format, and the time and numerical columns are identified and corrected. Subsequently, the `resample_safe()` function is employed to adjust the sampling rate and generate a coherent temporal sequence. This capability ensures that the data— even when recorded at irregular intervals—are made uniformly structured and suitable for subsequent analytical procedures.

```
    rule = f"{int(ms)}ms"

    num = df[["time"] + num_cols].set_index("time") if num_cols else None
    if num is not None and not num.empty:
        num = num.groupby(level=0).mean()
        num = num.resample(rule).interpolate("time").ffill().bfill()
```

4.5 Graphical Environment (bridge_realtime_gui.py)

This chapter addresses the design and implementation of the graphical environment of the system—an environment that presents sensor data and traffic information in a real-time, visual, and comprehensible manner. In this section, the main components of the interface—including system architecture, data and time management, sensor visualization, playback control, vehicle-passing simulation, the real-time processing loop, and analytical tools—are described in a structured way. This chapter forms the operational framework of the system and illustrates how the various software components integrate to deliver a stable, accurate, and synchronized representation of the bridge's behavior.

4.5.1 Overall System Architecture

The graphical environment is designed to present sensor data and traffic information in a smooth, coherent, and comprehensible manner. The structure consists of several independent layers: a rendering layer that manages the main display and animations using Pygame, an analytical tools layer that employs Tkinter for supplementary charts, and a data-management layer responsible for coordinating live data or recorded files. This separation of responsibilities ensures that the system

remains stable, responsive, and extensible in both real-time visualization and replay mode.

Goal, Role & Technology Integration

The graphical environment of this system is designed to transform raw sensor data and traffic information into a clear, interpretable, and real-time visual representation. This interface enables the operator to monitor the instantaneous condition of the bridge on a single screen without manually inspecting files, and to react quickly in case of structural changes or vehicle crossings.

Key functions of the GUI: First, it provides real-time visualization of sensor outputs. Various structural indicators—such as vibration, bending, and torsion—are extracted from the selected time window and displayed according to the installation location of each sensor on the bridge. This live monitoring allows the operator to follow the bridge's behavior as events occur.

In addition, the system enables visualization of vehicle crossings. Information such as entry time, speed, and vehicle class is received and animated on the bridge. In this way, the relationship between traffic load and sensor-recorded variations becomes visually observable and analyzable.

Another important component is the color-coded display of sensor status. Each sensor is assigned a specific color based on its condition level—normal, warning, or critical. This approach allows the operator to identify sensitive points in the shortest possible time without dealing with numerical values.

Modular architecture of the graphical environment: In live-data mode, the graphical interface communicates directly with the Receiver module. The Receiver stores the latest valid message received from the transmitter, and the GUI updates the display by reading this value. This modular architecture separates network communication and data processing from the main rendering loop, ensuring smooth and stable system execution.

Integration of complementary technologies: The graphical environment is developed by combining several complementary technologies to meet real-time requirements while also providing analytical tools to the user. The main rendering and live display are implemented with Pygame, an appropriate choice for 2D graphics, animation, and fast screen updates. This platform enables smooth sensor visualization, vehicle motion rendering, and high-rate management of the main program loop.

Alongside Pygame, Tkinter is used to provide analytical capabilities and supplementary charts. Since Pygame has limitations in creating complex interactive

windows and user-interface controls, Tkinter is employed as an auxiliary layer. In this section, the user can select a sensor and a time window and view outputs such as time-series plots or histograms. This separation keeps the main interface lightweight and fast while offering analytical features in a suitable environment.

To prevent interference between the two environments and avoid interface freezing, part of the Tkinter operations is executed in a separate thread. This design decision ensures that opening windows or interacting with Tkinter does not block the Pygame rendering loop, allowing the system to continue updating data in real time. As a result, this multi-technology architecture creates a dynamic, stable, and coherent interface that simultaneously supports live and analytical functionalities.

Main Components of the GUI System

The graphical environment consists of several distinct components, each with an independent yet complementary role in the system's real-time visualization. These components are designed so that data received from the underlying infrastructure is displayed continuously and in an organized structure within the user interface.

Main Display Surface: This surface, created using Pygame, serves as the foundational space for all graphical elements, including sensors, the time window, and moving objects.

Sensor Layout Layer: This layer is responsible for placing the sensors according to their actual positions on the bridge and displaying each sensor's instantaneous value through simple and intuitive symbols. This part only visualizes processed data and does not engage in modeling computations or deep analysis.

Traffic Events Display Layer: This layer handles the rendering of vehicles based on input data (such as time and speed) and generates their smooth movement across the bridge. To maintain efficiency, it uses lightweight and optimized animations to avoid increasing computational load.

Timeline: Another key component, the timeline provides the user with control tools. It allows adjusting the time window, pausing or playing the data, and navigating through past information. This section is designed independently and interacts with the display only through time synchronization.

Status Bar: Finally, the status bar presents concise information such as the current time, playback speed, or short system messages. Without cluttering the screen, it provides essential data to the operator and contributes to a smoother user experience.

Together, these components form the core structure of the GUI and establish a

framework that will be further detailed in the following sections—such as rendering, time control, and user interaction.

4.5.2 Data Handling & Time Management

This section describes the mechanisms through which the graphical environment keeps sensor data and playback time synchronized. By supporting both live data and recorded files, converting time into graphical coordinates, and extracting sensor values at any given moment, the system enables smooth, accurate, and controllable visualization of information.

File Mode and Live Mode

The graphical system supports two different data sources for visualization: File Mode and Live Mode. Selecting between these modes allows the user either to replay previously recorded data or to observe the real-time condition of the bridge based on incoming data.

In File Mode, data is read from stored CSV and JSON files. These datasets typically represent the output of a recording session and are used for historical analysis, evaluating structural performance over a specific period, or demonstrating sample scenarios. In this mode, all temporal information is extracted from the file contents, and the user can select and navigate through any point within the recorded time range.

In Live Mode, the graphical environment connects to the Receiver module and continuously reads and displays the latest received message. Here, the system's current time is synchronized with the real-time timestamps of the incoming data, and the GUI functions as a live dashboard. This structure ensures that even under high data volume or fast sampling rates, the graphical interface displays the bridge's condition without interruption.

Using these two modes provides high flexibility: File Mode is suitable for offline analysis and reporting, while Live Mode is used for real-time monitoring and detecting critical conditions (see Figure 4.2).

Time–Pixel Mapping

To correctly visualize data in the graphical environment, time must be converted into drawable coordinates. Since the GUI's data representation is time-based, all related elements—such as the timeline, playback cursor position, and selected inter-

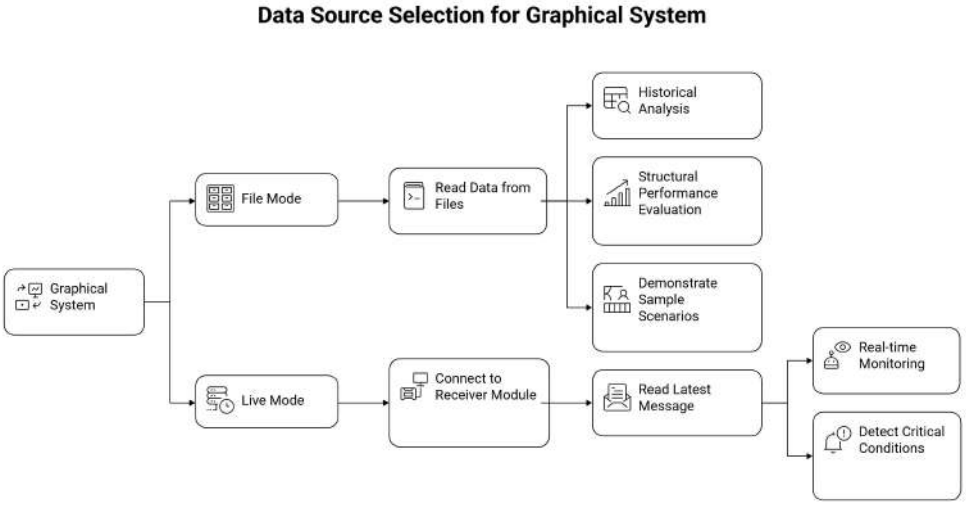

Figure 4.2: Interface options for switching between real-time monitoring (Live Mode) and offline playback (File Mode).

val—are displayed numerically and graphically in pixel coordinates. This process is performed using two main functions for converting time to pixel and pixel to time.

First, the system's active time window is defined by two values, `WINDOW_START` and `WINDOW_END`, which represent the beginning and end of the displayed data range. When the user zooms in or out, these values change, and the entire display area is recalibrated accordingly.

The function 'ts_to_x()' converts a given timestamp into a horizontal pixel value using the defined time window. This conversion enables precise placement of elements such as the current-time indicator, data positions, and window boundaries. Conversely, the function 'x_to_ts()' converts pixel coordinates back into their corresponding timestamps—an essential capability when dragging handles or selecting a specific portion of the timeline.

This bidirectional mapping allows the user to visually select time intervals, navigate through data, or adjust playback speed without entering numerical values. As a result, the system remains both interactive and accurate, ensuring that data visualization stays aligned with the temporal axis.

```
def ts_to_x(ts, rect):
    """Convert time -> x-coordinate on the timeline based
    on the entire CSV range (not just the window)."""
    if csv_max == csv_min:
        return rect.x
    tfrac = (ts - csv_min) / (csv_max - csv_min)
```

```
        return int(rect.x + tfrac * rect.w)

    def x_to_ts(x, rect):
        """Convert timeline x-coordinate -> absolute time
        within the CSV range."""
        frac = (x - rect.x) / max(1, rect.w)
        frac = max(0.0, min(1.0, frac))
        return csv_min + (csv_max - csv_min) * frac
```

Real-Time Sensor Value Extraction

To display the value of each sensor at a specific moment in time, the system must extract the appropriate value from the stored or incoming dataset. This process is handled by a dedicated function that determines the sensor value closest to the selected timestamp. The goal is to provide the graphical interface with a valid and displayable value for the current moment quickly and continuously, without loading the entire dataset.

In this method, the target timestamp is first converted into a comparable index. The function then performs a lightweight search to find the nearest recorded value in the sensor's time series. If the data is not stored at uniform intervals, the system uses simple interpolation or nearest-value selection to ensure smooth, jump-free visualization.

This mechanism supports two operational modes: Instant and Rolling. In Instant mode, the value corresponding exactly to the selected moment is displayed and used for determining sensor color or status. Rolling mode is typically used to display short-term averages or more stable values, ensuring reliable visualization even under noisy conditions.

Through this approach, the system can extract the correct value at any moment—despite high data volume or irregular sampling—and update sensor status colors, intensity bars, or other visual elements accordingly. This component is one of the key elements ensuring synchronization between data and graphical display.

4.5.3 Sensor Visualization & Status Encoding

This section explains how sensor values are transformed into an intuitive and visually interpretable representation within the graphical environment. First, the spatial position of each sensor is loaded based on the predefined location map, and then its measured value is displayed as a graphical indicator at the correct position on the bridge. Next, the status of each sensor is encoded using a simple color scheme (green, yellow, red) so that the user can quickly distinguish between normal, warn-

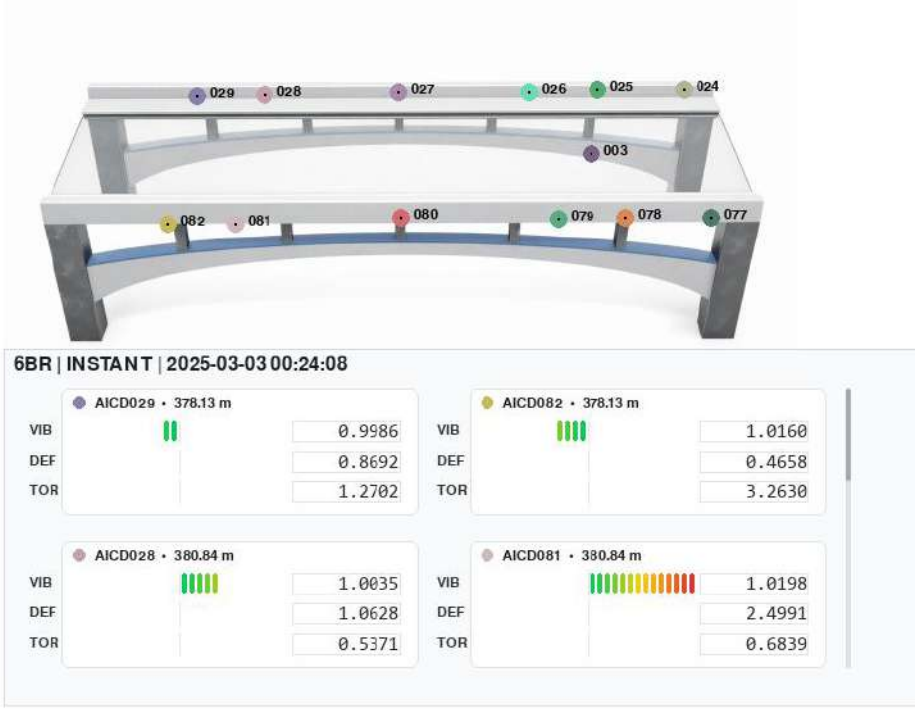

Figure 4.3: Sensor visualization on the Po Bridge showing spatial placement, real-time values, and color-coded bar indicators (green–yellow–red) representing normal, warning, and critical states.

ing, and critical conditions. This visual layer combines value, intensity, and status, enabling fast and intuitive monitoring of the structural behavior (see Figure 4.3).

Loading the Spatial Sensor Map

For the graphical environment to display each sensor's status at its correct location on the bridge, the system must first have access to a spatial map of all sensor positions. This information is stored in a separate JSON file and contains the two-dimensional coordinates of each sensor based on the coordinate system defined for the bridge.

At program startup, this file is read and the sensor coordinates are stored in an appropriate data structure. Each sensor is identified by its unique AICD, and its position is made available as a pair of horizontal and vertical coordinates. This separation between sensor data and sensor placement ensures that any modification in the bridge design or sensor locations can be applied simply by updating the JSON file, without altering the graphical logic.

This spatial map forms the foundation for all subsequent visual representations.

Using these coordinates, the system can convert numerical data into graphical positions and display each sensor's status as a visual element at the correct location.

Listing 4.1: Loading sensor coordinates from the JSON spatial map

```
with open("sensor_positions.json", "r") as f:
    sensor_map = json.load(f)

sensor_positions = {}

for entry in sensor_map:
    aicd = entry["id"]
    x = entry["x"]
    y = entry["y"]
    sensor_positions[aicd] = (x, y)
```

Sensor Value Visualization & Status Color Coding

After loading the spatial positions of the sensors, the next step is to visually represent each sensor's measured value and status. The goal is to allow the user to detect intensity changes or potential anomalies at a glance, without examining numerical data.

Intensity Visualization: The system uses simple and intuitive graphical indicators to display each sensor's instantaneous value at its corresponding spatial location. When a sensor value is extracted for the current moment, it is converted into a visual representation—typically a small bar or block next to the sensor's position, where the intensity is shown through height, thickness, or color. These miniature charts make it possible to immediately observe rapid fluctuations or sudden increases in sensor readings.

Since sensor values may represent different physical quantities—such as vibration, bending, or torsion—the visual representation is designed to remain uniform and readable regardless of the data type. This level of abstraction helps the operator interpret sensor conditions without dealing with technical details.

Status Color Coding: To enable quick recognition of each sensor's condition, the system uses a simple and standardized color scheme. This scheme is based on the sensor's status level—normal, warning, or critical—and serves to translate numerical values into an intuitive visual signal.

To determine the status, the system retrieves the current value of each sensor and compares it with predefined Warn and Alarm thresholds. The result is one of three states—"normal," "warning," or "critical"—represented respectively by green, yellow, and red. These colors appear as a small indicator (typically a vertical bar or colored dot) next to the sensor's location, conveying the status instantly.

The strength of this color-coding system lies in its clarity and simplicity. Even when the number of sensors is large and values change rapidly, the user can quickly identify sensitive or abnormal areas without numerical analysis or complex charts. This approach effectively reduces the operator's cognitive load and is highly suitable for real-time monitoring in data-intensive environments.

Ultimately, the intensity visualization, status color coding, and spatial positioning together form three complementary layers of information that collectively provide a clearer understanding of the structural condition.

Listing 4.2: Status evaluation logic based on dynamic threshold ranges

```
def get_status_level(val, thr):
    """
    Outputs:
    0 = Normal
    1 = Alarm
    2 = Warning
    """
    if not np.isfinite(val) or not thr:
        return None

    warn_lo  = thr.get("warn_lo",  -np.inf)
    warn_hi  = thr.get("warn_hi",   np.inf)
    alarm_lo = thr.get("alarm_lo", -np.inf)
    alarm_hi = thr.get("alarm_hi",  np.inf)

    if val < alarm_lo or val > alarm_hi:
        return 2     # Critical / Alarm
    if val < warn_lo or val > warn_hi:
        return 1     # Warning
    return 0         # Normal
```

Listing 4.3: Color mapping function used for converting sensor status levels into RGB values

```
def level_to_color(level):
    if level == 2:
        return LED_RED     # Critical / Alarm
    if level == 1:
        return LED_YELLOW  # Warning
    if level == 0:
        return LED_GREEN   # Normal
    return (120,120,120)   # Unknown / Invalid
```

4.5.4 Timeline & Playback Controls

This section describes the mechanisms through which the user can manage the flow of data visualization. The timeline allows selecting the desired interval, navigating through past data, and adjusting the inspection point, while the playback-speed

Figure 4.4: Timeline and playback-speed controls for interactive navigation of the visualization.

controls enable the user to slow down, speed up, or pause the display. Together, these features ensure that data navigation and analysis within the graphical environment remain fully interactive, precise, and synchronized with the visualization process, as shown in Figure 4.4.

Timeline Structure & User Interaction

The timeline is one of the key components of the graphical environment, enabling the user to navigate and control the displayed data range. It appears as a horizontal bar at the bottom of the screen, where the active data window—the period the user intends to view and analyze—is clearly marked. This bar consists of three main elements: the start handle, the end handle, and the current-time cursor.

Timeline Components: The start and end handles allow the user to shrink or expand the desired time window and select only the portion of data needed for analysis. The current-time cursor shows the instantaneous playback position and updates dynamically as it moves along the timeline. The timeline is designed to maintain precise alignment between time and graphical display, ensuring smooth and controllable navigation forward or backward through the data.

User Interaction: The user can click or drag the handles to select a desired interval, or move the cursor to change the inspection time. This direct interaction enables a fast and fluid data-viewing experience without entering numerical values or navigating complex menus. In offline browsing mode, the user can move backward, re-examine specific segments, or review events before reaching a critical point.

These interactive controls operate in real time without interrupting the main rendering loop. Converting pixel positions on the timeline to actual timestamps—and vice versa—is handled by dedicated functions to ensure that every movement is applied precisely within the correct temporal range (see Figure 4.4).

Playback Speed Control

To provide greater flexibility in data inspection, the system also allows adjustment of playback speed. Depending on the user's needs, the display speed can be increased or decreased. This feature is particularly useful when data has been recorded at high sampling rates or when a short time segment requires detailed review.

Playback speed directly affects the movement of the time cursor and the rate at which traffic events are simulated. However, the playback structure is designed so that synchronization among data, sensors, and vehicle motion is preserved under all conditions. As a result, even at high speeds, the system's visualization remains coherent and fully synchronized (see Figure 4.4).

4.5.5 Vehicle Simulation & Animation

This section describes how the passage of vehicles across the bridge is graphically simulated so that the relationship between traffic loading and sensor variations becomes visually and simultaneously observable. Data related to vehicle entry time, speed, and class is loaded, and based on the system's current time, each vehicle's position is dynamically calculated. The vehicles are then drawn on the bridge using lightweight and optimized rendering in Pygame, and once they exit the display area, they are removed from the simulation cycle. This mechanism produces smooth and realistic vehicle motion and supports better analysis of the structural response under traffic loads.

Vehicle Data Loading

To visualize vehicle crossings on the bridge, the system must receive vehicle-related information from input files and convert it into a format usable within the graphical environment. These data include the vehicle's entry time, travel speed, direction of movement, and vehicle class. The vehicle class typically determines its apparent size and influences how it is displayed on the bridge. After loading, the data are organized into a structured list, and during program execution, each vehicle enters the simulation cycle at the appropriate moment.

Position & Motion Calculation

The position of each vehicle on the bridge is calculated based on its speed and the system's current time. As time progresses, the vehicle's horizontal coordinates are continuously updated, producing smooth motion across the bridge. This calculation

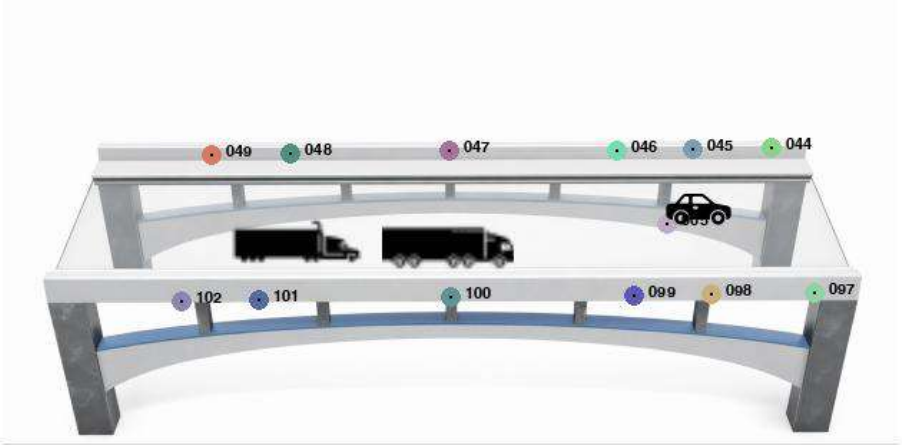

Figure 4.5: Simulated vehicle movement across the bridge, showing class-based rendering and time-synchronized position updates.

is designed to remain simple while maintaining full synchronization with the data-display timeline. If a vehicle enters the bridge earlier or later, the simulation begins precisely at that corresponding moment. This mechanism ensures an accurate link between sensor variations and the exact passage time of each individual vehicle.

Vehicle Rendering

After the position is calculated, vehicles are drawn on the screen using Pygame. Their graphical representation is simplified to keep computational load low, but their size is fully consistent with their vehicle class. The color and appearance of each vehicle are chosen to enhance visual distinction. When a vehicle exits the bridge area, it is removed from the simulation cycle to maintain system efficiency. This process ensures that the graphical environment remains smooth and responsive even under dense traffic-simulation conditions (see figure 4.5.

4.5.6 Real-Time Main Loop

This section describes the cycle that forms the core of the graphical environment's operation. Within this loop, user events are processed, the display time is updated, and all graphical elements are redrawn. The rapid and continuous repetition of these steps ensures that data visualization, sensor status updates, and vehicle simulations remain fully live, smooth, and synchronized with the actual flow of data.

Event Handling

The core execution of the graphical environment operates within a real-time loop responsible for managing interactive behavior and continuously updating the display. In each cycle of this loop, user events are processed first—events such as mouse clicks, dragging the timeline handles, pressing play or pause buttons, and exiting the program. This stage ensures that the system responds immediately and without delay to user inputs. System-level Pygame events, such as window-close requests, are also handled here.

Time Updating

After event processing, the next step is updating the current time. If playback mode is active, the system time advances based on the selected playback speed and the frame-to-frame time difference. In pause mode, time remains fixed and changes only when the user moves along the timeline. This design enables both slow, precise inspection of data and accelerated or decelerated playback. The current time is one of the most critical variables in the main loop, as other components—such as vehicle simulation and sensor visualization—depend directly on it.

Scene Rendering

During the rendering stage, all graphical elements of the screen are reconstructed. This process includes clearing the screen, drawing the bridge background, displaying sensors with their status colors, rendering intensity indicators, drawing vehicles at their computed positions, updating the timeline, and showing system messages. These operations are performed using lightweight and optimized Pygame functions to maintain a high frame rate and preserve smooth visual performance.

At the end of this stage, the rendered frame is updated on the display so that the results of all computations and drawings become visible. This cycle repeats dozens of times per second, enabling the graphical environment to deliver a live, synchronized, and delay-free visualization.

4.5.7 Tkinter Analysis Tools (Chart & Analysis Windows)

This section describes the supplementary capabilities of the graphical environment that allow the user to view charts and perform more detailed analysis of sensor data in separate windows. These tools are implemented using Tkinter and enable

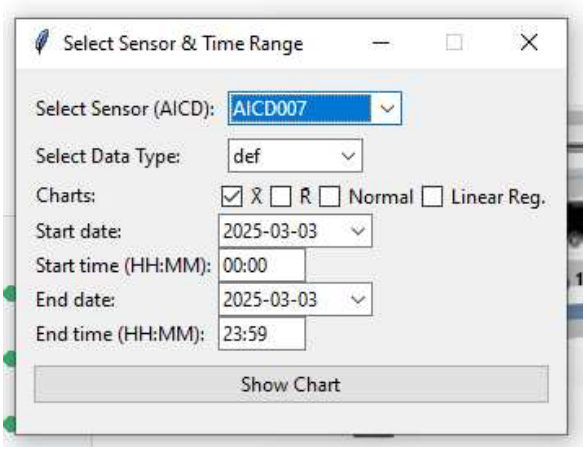

Figure 4.6: Independent Tkinter analysis window for selecting sensor and time range to generate detailed charts.

closer inspection of a sensor's behavior—or a specific segment of the data—without disrupting the real-time display.

Purpose of Analysis Windows

Alongside real-time visualization, the system provides a set of lightweight analytical tools that allow the user to examine the data of a specific sensor in greater detail when needed. These capabilities are implemented as separate windows, where the user can select a sensor and a desired time interval to view supplementary charts. The purpose of this component is not to replace full-scale analytical workflows, but to offer a quick tool for more precise inspection of a sensor's condition at a given moment or within a short time span.

User Interaction & Chart Selection

In these windows, the user typically begins by selecting the desired sensor and then specifying the chart type or time interval. This approach enables focused data inspection without cluttering the main display. The advantage of this design is that the main screen remains dedicated to fast, high-level monitoring, while detailed examinations are moved to auxiliary windows. This separation improves interface readability and enhances operator focus, as shown in Figure 4.6.

Non-blocking Execution

Since the main graphical loop runs in Pygame, opening analytical windows must not interrupt or slow down the primary display. For this reason, the Tkinter component is managed in a way that minimizes interference with the rendering loop. In practice, tasks related to the analytical windows are executed independently, ensuring that even while charts are open, real-time visualization, vehicle animations, and timeline controls continue without interruption. This design choice keeps the user experience smooth and ensures stable system performance under operational conditions.

4.5.8 System Integration & Auxiliary Files

This section addresses the supporting components that enable proper data integration and accurate visualization within the graphical environment. The `shared_utils` module preprocesses and standardizes the data before passing it to the GUI, while auxiliary files—such as the spatial sensor map and graphical assets—provide the essential information required for positioning and rendering visual elements. Together, these components form a stable foundation that ensures the interface operates in an organized, extensible, and fully synchronized manner.

GUI Connection with `shared_utils`

To display the data, the graphical interface uses the data that has already been structured in the `shared_utils` module. This module is responsible for preparing and standardizing the data, and the GUI only receives its final output. Full details related to `shared_utils` are provided in the corresponding chapter.

Sensor Location Files and Graphic Assets

Several auxiliary files also play an important role in the interface's functionality. The `sensors_map.json` file specifies the spatial position of each sensor on the bridge, and the GUI uses these coordinates to render the sensors in their correct locations. Additionally, a set of graphic files (Assets), including background images and visual components of vehicles and the user interface, is used to ensure proper and lightweight rendering of the environment. Keeping these files outside the codebase enables easy updates and further development of the interface.

4.5.9 Summary

This chapter explained the architecture and main components of the graphical environment of the bridge monitoring system. By combining real-time rendering capabilities in Pygame with the analytical tools of Tkinter, the environment provides a platform in which sensor data and vehicle passages are displayed in a synchronized and comprehensible manner. Mechanisms for time management, display-window control, sensor-state encoding, and traffic simulation form an integrated system that enables rapid monitoring, trend observation, and detection of abnormal conditions. Auxiliary files and processed data supply the necessary infrastructure for delivering this smooth and organized visualization. This chapter establishes the visual foundation of the system and prepares the ground for the analysis and evaluation presented in the following chapters.

Chapter 5

Results

In this chapter, the results of running the developed visualization system are presented. The aim is to demonstrate how the software performs when working with real data from the `Ponte Po - Careggiata Nord` bridge. The chapter begins with an overview of the bridge and the collected datasets, followed by an explanation of the user interface behavior, sensor visualization, vehicle movement representation, and the system's graphical elements. Next, examples of the actual program execution are provided along with the corresponding screenshots, and finally, a general summary of the system's achievements in graphical data representation is presented. All sections of this chapter focus exclusively on displaying and describing the visual output of the system, and no structural analysis is performed.

5.1 Case Study Introduction

In this case study, the visualization system developed in this project has been tested and executed using real data from the `Ponte Po - Careggiata Nord` bridge. This bridge, which is part of the A7 highway network and managed by the Milano–Serravalle group, has undergone structural rehabilitation, strengthening, and installation of a permanent monitoring system in recent years as part of the national *Safe Roads* program. The available official documents, including the *Calibration and System Delivery Report* [56] (`Certificato di collaudo_Ponte P`) and the `General Technical Report RT01` [57] (`RT01 - Relazione Tecnica Genera`), indicate that a wide set of dynamic sensors has been installed on this structure to enable continuous monitoring of the bridge's condition.

According to the information provided in these documents, a significant number of triaxial accelerometers and inclinometers (AICD), along with temperature

	time	03091002_y	03091002_z	03091003_y	03091003_z	03091005_y	03091005_z	03091006_y	03091006_z	03091007_y	03091007_z	03091008_y	03091008_z	0309100
1	2025/03/03 00:00:00	-0.733525	-2.42195	1.80391	0.425125	0.078631	2.33085	-2.63432	-1.05145	0.674375	-0.790341	-2.19520	0.300334	0.802
2	2025/03/03 00:00:01	-0.733500	-2.42231	1.80463	0.424678	0.078663	2.33310	-2.63539	-1.05071	0.673283	-0.789268	-2.19431	0.301028	0.801
3	2025/03/03 00:00:02	-0.733387	-2.42232	1.80427	0.424761	0.077574	2.33062	-2.63427	-1.05116	0.674053	-0.789250	-2.19415	0.300165	0.803
4	2025/03/03 00:00:03	-0.732593	-2.42228	1.80441	0.425491	0.077256	2.33014	-2.63368	-1.04976	0.673932	-0.788119	-2.19478	0.300582	0.803
5	2025/03/03 00:00:04	-0.733541	-2.42145	1.80391	0.424930	0.077765	2.33168	-2.63454	-1.05000	0.673572	-0.787910	-2.19385	0.300928	0.802
6	2025/03/03 00:00:05	-0.731957	-2.42186	1.80416	0.424425	0.076817	2.33195	-2.63427	-1.05081	0.674724	-0.788976	-2.19498	0.299947	0.803
7	2025/03/03 00:00:06	-0.733673	-2.42253	1.80361	0.423646	0.078082	2.33212	-2.63359	-1.04977	0.674270	-0.789471	-2.19540	0.300882	0.802
8	2025/03/03 00:00:07	-0.733382	-2.42144	1.80423	0.425288	0.077987	2.33211	-2.63379	-1.05003	0.674737	-0.790190	-2.19458	0.300637	0.802
9	2025/03/03 00:00:08	-0.733578	-2.42111	1.80364	0.425119	0.078595	2.33090	-2.63495	-1.04978	0.673494	-0.788657	-2.19484	0.300661	0.803
10	2025/03/03 00:00:09	-0.733633	-2.42120	1.80387	0.425759	0.078786	2.33132	-2.63452	-1.05073	0.673926	-0.789200	-2.19476	0.301848	0.802
11	2025/03/03 00:00:10	-0.734091	-2.42301	1.80409	0.425342	0.078259	2.33253	-2.63544	-1.05074	0.674549	-0.789456	-2.19376	0.301148	0.803
12	2025/03/03 00:00:11	-0.733143	-2.42280	1.80365	0.424247	0.078125	2.33081	-2.63380	-1.05042	0.673513	-0.790596	-2.19414	0.301906	0.801
13	2025/03/03 00:00:12	-0.733215	-2.42261	1.80417	0.425193	0.078069	2.33152	-2.63540	-1.05102	0.673107	-0.788950	-2.19508	0.299671	0.802
14	2025/03/03 00:00:13	-0.733301	-2.42262	1.80337	0.425740	0.078479	2.33270	-2.63553	-1.05058	0.673576	-0.789286	-2.19553	0.301391	0.802
15	2025/03/03 00:00:14	-0.731956	-2.42271	1.80379	0.424775	0.077545	2.33272	-2.63407	-1.05096	0.673795	-0.789430	-2.19508	0.301281	0.802
16	2025/03/03 00:00:15	-0.733210	-2.42208	1.80367	0.423953	0.077834	2.33266	-2.63493	-1.05102	0.674218	-0.789558	-2.19466	0.300672	0.802
17	2025/03/03 00:00:16	-0.733002	-2.42283	1.80353	0.426001	0.078163	2.33114	-2.63411	-1.05053	0.672794	-0.789130	-2.19481	0.300635	0.802
18	2025/03/03 00:00:17	-0.732745	-2.42265	1.80353	0.425294	0.078205	2.33271	-2.63464	-1.05133	0.674285	-0.789335	-2.19360	0.301885	0.802
19	2025/03/03 00:00:18	-0.732987	-2.42263	1.80398	0.423067	0.078156	2.33082	-2.63488	-1.04976	0.673809	-0.789890	-2.19502	0.300933	0.802
20	2025/03/03 00:00:19	-0.733139	-2.42216	1.80452	0.423975	0.078097	2.33113	-2.63411	-1.04981	0.674012	-0.790394	-2.19575	0.300958	0.802
21	2025/03/03 00:00:20	-0.733269	-2.42278	1.80404	0.423985	0.078143	2.33198	-2.63488	-1.04948	0.674088	-0.789418	-2.19481	0.300228	0.802
22	2025/03/03 00:00:21	-0.733708	-2.42227	1.80352	0.425793	0.078149	2.33201	-2.63453	-1.04976	0.674137	-0.789777	-2.19558	0.300293	0.802
23	2025/03/03 00:00:22	-0.733365	-2.42362	1.80403	0.425005	0.078832	2.33194	-2.63449	-1.05012	0.674502	-0.788944	-2.19417	0.301541	0.802
24	2025/03/03 00:00:23	-0.732858	-2.42239	1.80369	0.425088	0.078080	2.33170	-2.63462	-1.05032	0.673040	-0.789822	-2.19387	0.300801	0.803

Figure 5.1: Example of raw sensor measurements stored in CSV format. Each row corresponds to a timestamped record, while the columns represent the measured values from different sensors installed on the bridge.

sensors and local processing modules, have been installed on the bridge. These sensors are positioned across various spans, on the girders, and in key structural locations, and their exact distribution is documented in the official layout drawings T01 [58] (`T01 - Layout dell'impianto di m`) and T02 [59] (`T02 - Layout dell'impianto di m`). Based on the `Collaudo` report, the complete system includes 106 AICD sensors, five temperature sensors, several local digital modules, and data management and transmission units, all of which together form the measurement chain of the monitoring system [56].

The data used as input for the visualization system in this project consists of real measurements recorded by the bridge sensors. These data include time-series values of vibration, bending, torsion, and structural temperature, along with information related to vehicle movements on the bridge. The data are stored in tabular files in Excel and CSV formats. A sample of these data is presented in tabular form in Figures 5.1 and 5.2. The purpose of using real data is to demonstrate the capabilities of the visualization system under practical conditions, rather than to perform structural analysis. Therefore, the data are imported directly into the software without technical processing or engineering interpretation and are displayed solely in visual form.

This case study represents a real execution scenario of the system, in which the behavior of the bridge over a defined time interval—together with vehicle traffic—is presented through animations and graphical displays. This chapter demonstrates

	Id	StartTimeStr	LaneNo	LaneName	BaseClass	Scheme	ClassId	MoveStatus	FrontTo	BackToFr	Duration	VehicleLength	Gr
55389	690017	03/03/2025 14:52		3 Corsia 2 - Marcia veloce	21	EUR13	1	-1	19.891	19.723	0.277	4.58	
55390	690018	03/03/2025 14:52		2 Corsia 1 - Marcia	1	EUR13	13	0	1.202	1.102	0.304	2.15	
55391	690019	03/03/2025 14:52		2 Corsia 1 - Marcia	21	EUR13	1	1	10.997	10.276	0.397	4.53	
55392	690023	03/03/2025 14:52		3 Corsia 2 - Marcia veloce	21	EUR13	1	1	1.466	1.298	0.272	4.32	
55393	690024	03/03/2025 14:52		3 Corsia 2 - Marcia veloce	21	EUR13	1	1	2.548	2.392	0.272	4.45	
55394	690025	03/03/2025 14:52		3 Corsia 2 - Marcia veloce	21	EUR13	1	1	3.497	3.341	0.28	3.88	
55395	690026	03/03/2025 14:52		3 Corsia 2 - Marcia veloce	21	EUR13	1	-1	3.774	3.63	0.272	4.58	
55396	690027	03/03/2025 14:52		3 Corsia 2 - Marcia veloce	21	EUR13	1	0	5.3	5.144	0.25	4.55	
55397	690020	03/03/2025 14:53		2 Corsia 1 - Marcia	21	EUR13	1	-1	18.341	18.148	0.395	4.84	
55398	690021	03/03/2025 14:53		2 Corsia 1 - Marcia	22	EUR13	2	1	4.579	4.375	0.474	7.57	
55399	690022	03/03/2025 14:53		2 Corsia 1 - Marcia	21	EUR13	1	1	3.954	3.642	0.397	3.92	
55400	690028	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	21	EUR13	1	-1	13.209	13.064	0.286	4.24	
55401	690029	03/03/2025 14:53		4 Corsia 3 - Sorpasso	21	EUR13	1	-1	0.968	0.776	0.272	4.92	
55402	690030	03/03/2025 14:53		2 Corsia 1 - Marcia	21	EUR13	1	-1	5.432	5.252	0.348	3.91	
55403	690031	03/03/2025 14:53		2 Corsia 1 - Marcia	21	EUR13	1	1	2.692	2.536	0.335	3.65	
55404	690032	03/03/2025 14:53		2 Corsia 1 - Marcia	15	EUR13	9	-1	11.899	11.754	0.895	15.87	
55405	690033	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	21	EUR13	1	1	11.057	10.901	0.273	4.6	
55406	690034	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	21	EUR13	1	-1	0.853	0.697	0.27	4.53	
55407	690035	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	21	EUR13	1	1	3.485	3.329	0.246	4.72	
55408	690036	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	22	EUR13	2	1	7.404	7.259	0.294	5.71	
55409	690037	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	21	EUR13	1	0	1.322	1.13	0.342	5.01	
55410	690038	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	21	EUR13	1	1	2.836	2.632	0.27	4.53	
55411	690039	03/03/2025 14:53		3 Corsia 2 - Marcia veloce	21	EUR13	1	1	2.127	1.971	0.304	4.28	

Figure 5.2: Example of vehicle movement data stored in Excel format. The table contains information about vehicles crossing the bridge, including their identification, entry time, exit time, and other attributes used for traffic visualization in the system.

that the system can continuously show sensor locations on the bridge image, real-time recorded values through specific color mappings, and vehicle movements based on their entry and exit times. The primary goal of this section is to provide a clear picture of how the software performs when working with real data and to illustrate the consistency and stability of its execution in a practical scenario.

According to the official documentation, a wide set of dynamic sensors has been installed on the structure. An overview of the bridge location is shown in Figure 5.3, and detailed views of the underside structural configuration are illustrated in Figures (5.4 and 5.5).

5.2 User Interface and Data Visualization

The user interface designed for this project is built so that the user can, at a glance, observe the overall condition of the bridge, the location of the sensors, and the movement of vehicles. The execution environment is based on the Pygame library, and at startup, a schematic image of the Po Bridge is placed in the background. On this image, the exact positions of the sensors—previously defined in the official monitoring network documents—are displayed. Each sensor is represented by a small icon, allowing the user to identify its location relative to other parts of the bridge while simultaneously viewing its real-time measured value.

One of the most important elements of the interface is the sensor value display. For each sensor, three main quantities—vibration, bending, and torsion—are shown

Figure 5.3: Satellite view of the Ponte Po – Careggiata Nord location on the A7 motorway.

Figure 5.4: Side view of the underside of the Ponte Po – Careggiata Nord bridge structure.

Figure 5.5: Underside structural configuration of the bridge deck showing the girder system.

as small colored bars. The color of these bars changes dynamically based on the input value, enabling the user to understand at a glance whether the sensor reading is within a normal range or moving toward warning levels. This color-coding, similar to industrial monitoring systems, is designed to visually convey sensor status without requiring numerical analysis. Under normal conditions, the bars appear in calm, stable colors; as the input value increases, the color spectrum shifts toward warning tones, and if predefined thresholds are exceeded, the bar's appearance changes to draw attention.

The interface also includes a time-control section that allows the user to navigate through different time intervals. A timeline is placed at the bottom of the screen, which can be moved to observe the bridge's behavior at various moments. This timeline shows both the current time and the full range of available data, and the user can manually adjust the start and end points of the displayed interval. Play and pause controls, similar to a video player, are also provided to allow continuous viewing of vehicle movements and sensor changes.

Vehicle movement is extracted from real data, and each vehicle is animated on the screen based on its recorded entry and exit times. Vehicles are displayed in different sizes depending on their type and weight class, and they move across the bridge at speeds consistent with the input data. This feature enables the user to visualize the effect of real traffic during the playback period and understand how many vehicles were crossing the bridge at any given moment.

Another part of the interface provides access to numerical charts. By selecting a specific sensor, the user can view a time-series plot of its recorded values over a chosen interval. These charts, displayed in a separate window, allow the user to observe general trends in the sensor data without requiring specialized analysis. Additional charts—such as averaging, distribution, and other simple visualizations—are also available to complement the system's graphical capabilities, though within this project they serve only a visual purpose and are not used for structural analysis.

Overall, the user interface is designed so that all its elements—from the bridge and sensor display to the timeline, vehicle animations, and charts—work together to provide an integrated visual experience. The primary goal is to present the bridge's behavior and real data in the simplest and most intuitive form, without requiring technical interpretation, within a clear and comprehensible visual environment.

Figure 5.6: System state with no vehicles on the bridge.

5.3 Examples and Screenshots of System Execution

To illustrate how the system operates and to demonstrate the quality of its graphical output, this section presents several examples of the program running with real data from the Po Bridge. The images show the state of the bridge at different moments—ranging from periods with no vehicle traffic to intervals in which one or more vehicles are crossing the structure.

In the first set of images, the bridge is shown in a state with no vehicles present. In this situation, the sensors appear with stable, unchanging colors, and the vibration and bending values fluctuate only within calm, steady ranges. This condition represents traffic inactivity, and the user interface displays it as a completely static and tranquil scene. The uniform colors of the sensor indicators inform the user that no significant variation is occurring, and the timeline simply marks the current moment without any change in vehicle positions (Figure 5.6).

In contrast, the images corresponding to the passage of a vehicle show the presence of a moving object on the bridge surface. A vehicle enters from one side of the span and moves continuously toward the exit, with a speed consistent with the recorded data. The animation is designed to convey the smooth motion of a real vehicle. Simultaneously, the small sensor bars reflect moment-to-moment variations, and the colors of some sensors shift slightly. These changes do not imply structural analysis; they simply represent numerical variations in the input data, visualized through color mapping (Figure 5.7).

Figure 5.7: Visualization of a single vehicle crossing the bridge.

Another screenshot example shows a situation where two vehicles are on the bridge at nearly the same time. In this image, the user can see each vehicle following its path, with their relative positions along the bridge clearly visible. The simultaneous movement of two vehicles adds more dynamism to the scene, and the program maintains an appropriate frame rate to display both animations without interference. The timeline is fully synchronized with this state, indicating the exact moment within the dataset when this event occurs (Figure 5.8).

A different set of images demonstrates the effect of adjusting the time interval using the timeline. Here, the user moves the time backward or forward, and the interface immediately updates the sensor values and vehicle positions to match the selected moment. The result is a visual display that allows the user to review different moments and observe how the bridge appeared at each point in time. This feature is particularly useful for visually examining traffic flow and the apparent condition of the bridge throughout the data sequence.

In the end, several practical examples relate to the system's chart-plotting capability. When the user selects one or more charts and specifies a desired time range, a chart window opens displaying the data corresponding to the selected charts. One of the key features of this system is its flexibility in visualizing charts; that is, the user can view any number of charts simultaneously. In such cases, the user interface automatically adjusts the layout based on the number of charts. For example, if two charts are selected, they are displayed side by side, and if four charts are selected,

Figure 5.8: Example showing two vehicles simultaneously on the bridge.

they appear in a structured grid. This dynamic layout allows the user to view and compare multiple charts at the same time while keeping the visual structure of the page organized and easy to understand (Figure 5.9, 5.10, and 5.11).

One of the examples presented in Figure 5.9 relates to the AICD005 sensor, which records vibration (VIB) data. In this case, a set of charts is provided to illustrate different aspects of the vibration signal. The first chart is the Moving Average Chart (X Chart), which depicts the trend of vibration values over time. In this chart, the horizontal axis represents the time within the selected interval (from 03:00:00 to 03:01:00), while the vertical axis represents the vibration magnitude, which varies approximately between 0.98195 and 0.98225. This chart visualizes the small and gradual fluctuations of the vibration signal, reflecting the natural dynamic behavior of the structure.

Another chart provided for this sensor is the Range Chart (R Chart), which illustrates the degree of variability in the data within each sampling interval. The peaks observed in this chart indicate sudden increases in the amplitude of vibrations. Such variations may be associated with factors such as the passage of vehicles over the bridge or minor changes in the bridge's surface conditions.

In addition, the system offers a statistical representation of the data through a normal distribution chart. In this chart, the blue bars represent the frequency of the actual recorded vibration values, while the orange curve corresponds to the fitted normal distribution. This distribution is centered around a mean value of approx-

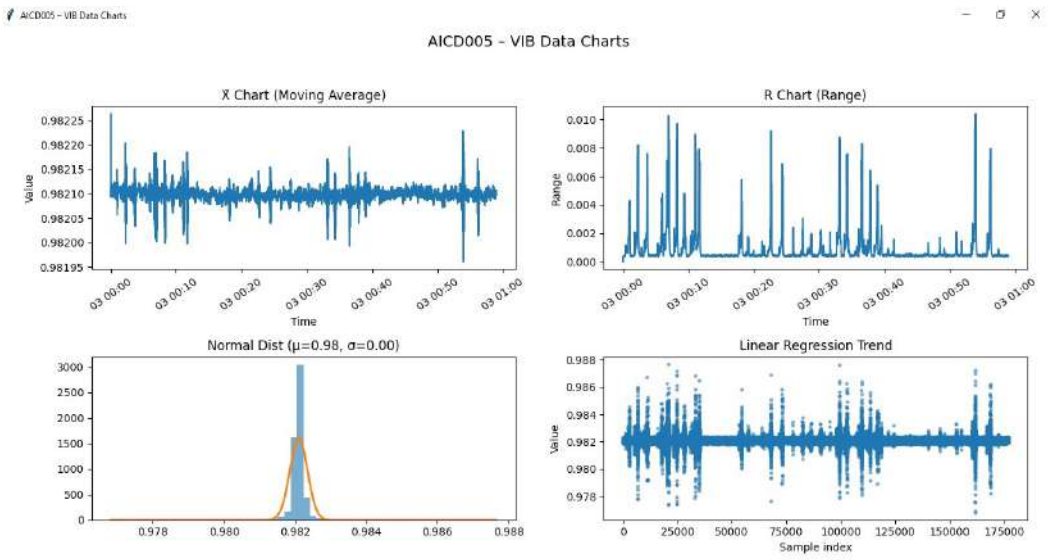

Figure 5.9: Vibration (VIB) charts for sensor AICD005: (a) moving average chart, (b) range chart, (c) normal distribution plot, and (d) linear regression trend.

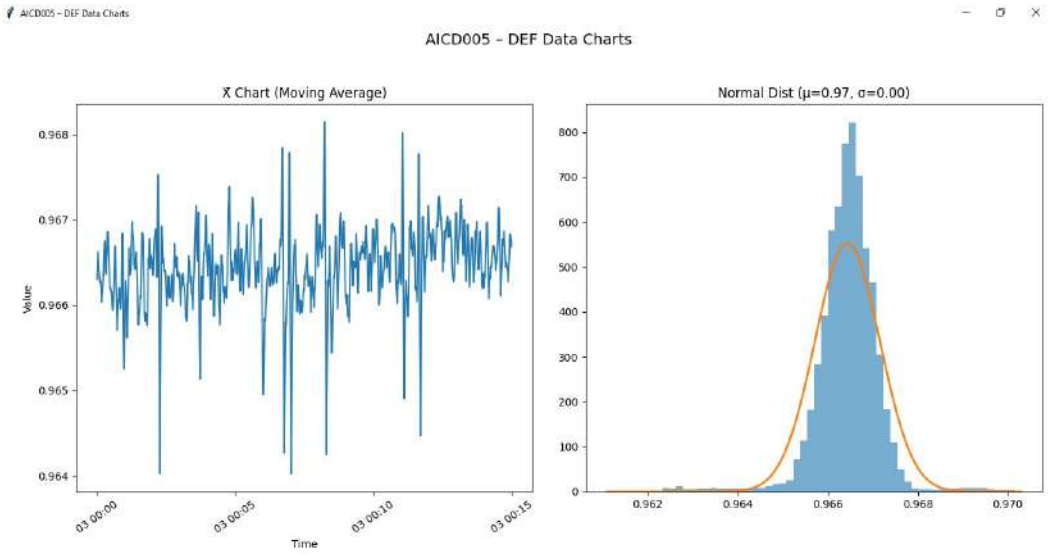

Figure 5.10: Deformation (DEF) charts for sensor AICD005 including: (a) moving average chart and (b) normal distribution plot.

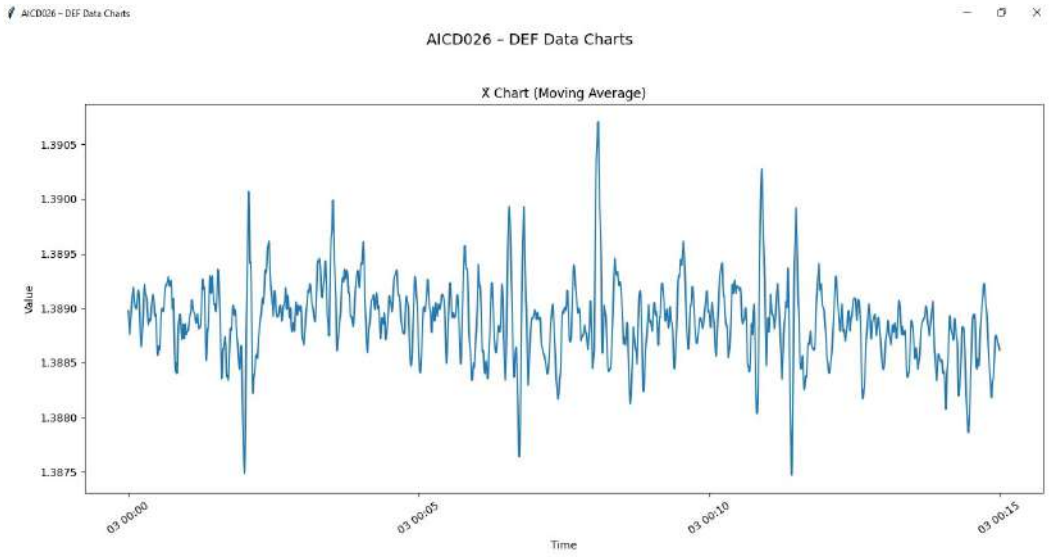

Figure 5.11: Moving average chart showing deformation (DEF) values for sensor AICD026.

imately 0.982 with a very small standard deviation, indicating that the vibration data remain within a relatively stable range and close to the mean value.

One of the additional charts provided in this set is the linear regression trend chart. In this chart, the blue points represent the measured vibration values. This type of visualization helps the user observe the overall trend of the signal over time and identify any long-term changes in the structural vibration behavior, should they occur.

Overall, these examples demonstrate that the visualization system is capable of presenting real bridge data in a clear, traceable, and visually understandable manner. The various states—from stillness to the passage of one or multiple vehicles, and from static displays to replaying different moments—are shown consistently and coherently across all images, giving the user a clear view of the system's successful performance.

5.4 Summary of the Visualization System Execution Results

Running the visualization system with real data from the Po Bridge showed that the software works smoothly and can clearly display a large, multi-source dataset in a visual and easy-to-understand way. The dataset used in this project consisted of

real measurements recorded by the accelerometers, inclinometers, and temperature sensors installed on the bridge. These raw data were fed directly into the program—without preprocessing or engineering interpretation—with the goal of visually presenting vehicle movement, real-time sensor variations, and how the measurements evolve over time within a graphical interface.

The results confirmed that the Pygame environment could render the bridge image, sensor locations, and vehicle animations in full synchronization, without interruptions or delays. Real-time changes in sensor values were displayed through color updates, allowing the user to quickly understand the current state of each sensor without performing additional calculations. The timeline played a central role in navigating the dataset: when the time position was adjusted, all visual elements—from vehicle locations to sensor color mappings—updated immediately to match the selected moment in time, demonstrating the stability and consistency of the system.

Additional charts for selected sensors provided complementary information by showing overall trends in separate plotting windows. This confirmed that the system can present the same dataset both through the graphical bridge interface and through simple numerical visualizations.

Overall, this case study confirmed that the main objective of the project—developing a tool for the graphical representation of real bridge-monitoring data—was successfully achieved. The software effectively displayed traffic flow, sensor conditions, and time-dependent changes, creating an environment in which an operator can understand the overall state of the bridge at any moment through visual inspection alone. Although the system is not designed for structural analysis, it can be useful for preliminary monitoring, general behavioral inspection, training activities, demonstrations of sensor performance, and visual reporting.

With further development—such as adding additional data layers, enabling long-term playback, or integrating intelligent analysis tools—the system could play a broader role in bridge-monitoring workflows. Nevertheless, even in its current form, the tool has proven to be lightweight, fast, and fully operational, capable of presenting real data from a major structure in a smooth, stable, and intuitive manner.

Chapter 6

Conclusions

This chapter summarizes the key findings of the research, evaluates the achievement of the defined objectives, and discusses the main limitations and potential directions for future development

6.1 Summary of Results and Achievements

The results of this research demonstrate that the developed system has successfully provided an efficient platform for real-time visualization of structural health monitoring data, fully achieving the main objective of the thesis—namely, transforming raw sensor data into understandable and actionable information for engineering decision-making. In this project, an integrated framework was designed that receives, synchronizes, and processes the data, and ultimately presents them through an interactive graphical interface. This process enables the observation of structural behavior during vehicle passages and under various environmental conditions, allowing engineers to examine and analyze the dynamic response of the structure in real time.

In addition to supporting real-time monitoring, the presented system includes the capability to adjust playback speed and review data across different time intervals—an important feature that enables more detailed analysis and closer examination of the structural response. The designed user interface, by simultaneously displaying multiple sensor types, providing real-time status information, and identifying potential anomalies, plays an effective role in assessing structural health and interpreting the corresponding data. Overall, the results of this thesis show that it is possible to effectively bridge the gap between raw SHM data and practical engineering analysis using lightweight and extensible tools.

6.2 Evaluation of Objectives

Considering the initial objectives of this research, the results indicate that all key intended elements have been successfully accomplished. The first objective—loading and processing sensor data—was effectively achieved through the implementation of a modular structure for reading CSV, Excel, and JSON files and converting them into a standardized temporal format. This process enabled data extraction, cleaning, and synchronization, thereby preparing the foundation for analytical processing in the real-time visualization stage. Furthermore, another important objective—the development of preliminary algorithms for anomaly detection and identification of critical points—was fulfilled through the design of a sensor-status detection logic (Normal–Warning–Critical), temporal variation analysis, and color-coded status representation within the GUI.

In line with the main goal of the project, namely the creation of a real-time visualization system, a Pygame-based graphical environment was developed that can receive incoming data instantly and display them with high processing speed. This component not only presents the dynamic behavior of sensors in real time, but—thanks to the adjustable playback speed and Timeline Control—also enables data analysis across different temporal scales. Additionally, the implementation of a complementary user interface using Tkinter, which provides analytical tools such as charts and time-range selection, contributed to a more complete and efficient user experience.

Another significant achievement of the project is the capability to filter and analyze data based on vehicle characteristics—such as vehicle class, number of axles, and tire type—which plays an important role in improving analytical accuracy and identifying structural behavior under different loading conditions. This capability, combined with the system's performance indicators—including high processing and rendering speed, acceptable accuracy in detecting critical variations, ease of use of the interface, and the scalability of the software architecture—demonstrates that the research objectives have not only been achieved but that the system has exceeded expectations in several functional aspects.

Improvements made in areas such as code structure quality, maintainability, detailed documentation, and intuitive interface design also played a significant role in the successful completion of the project. These features have ensured that the system is not only suitable for testing on the Po Bridge dataset but is also prepared for development and deployment in similar projects. Consequently, the set of objectives defined in this thesis has been successfully fulfilled, and the resulting

achievements provide a solid foundation for developing more advanced tools in the field of structural health monitoring.

6.3 Limitations and Opportunities

Despite achieving satisfactory results and fulfilling the main objectives, this research has been accompanied by a set of limitations and challenges that must be acknowledged to provide a realistic picture of the system's performance. The first category of limitations relates to the quality of the sensor data. Data obtained from SHM systems are inherently accompanied by varying levels of noise, instability, and potential errors, all of which may affect the accuracy of the analyses. Temperature variations, environmental fluctuations, and electromechanical interferences may also introduce slight deviations in the recorded values. These factors make part of the results dependent on sensor quality and necessitate precise cleaning, smoothing, and normalization during preprocessing.

From a technical perspective, the system also faced certain constraints. Real-time processing of large volumes of data requires adequate computational resources, and although the current implementation performed well for the available dataset, an increase in the number of sensors or sampling rate could impose additional load on the system. Moreover, network latency in real-time mode—particularly when transmitting data via TCP Socket—may, under certain conditions, cause short delays in visualization. Although such delays generally remained within acceptable limits, they still represent a potential limitation.

From a methodological standpoint, challenges also exist. Some of the models and thresholds used for sensor-status detection are based on simplified assumptions or practical experience and may require recalibration when applied to larger datasets or different loading conditions. In particular, in the anomaly-detection component, the relative dependence on the quality of training data or predefined initial parameters may reduce accuracy under atypical conditions.

Despite these limitations, the scientific and practical value of this research remains significant. Many of the challenges can be mitigated or resolved, and future versions of the system may be improved through the development of more advanced models, the use of noise-resistant methods, or the adoption of more powerful hardware. Therefore, these limitations not only do not undermine the validity of the results but also provide a clear pathway for future research and development.

6.4 Suggestions for Future Developments

Based on the results of this research and the limitations discussed, multiple pathways exist for further development and enhancement of the system, each of which can significantly increase the practical value and analytical accuracy of structural health monitoring. One of the most important areas for advancement is the use of machine learning and deep learning methods for automated damage and anomaly detection. These methods are capable of adaptively analyzing complex patterns within the data and identifying abnormal structural behavior with higher precision. Additionally, employing advanced signal-processing algorithms—such as adaptive filters, time–frequency methods, and empirical mode decomposition (EMD)—can improve the accuracy of detecting transient events and critical points. In the domain of technical optimization, adopting distributed architectures for data processing and reducing latency, as well as improving real-time performance through code optimization and the use of more powerful hardware, represent effective future steps.

In the data domain, several significant enhancements can be implemented. First, collecting larger datasets under diverse operational conditions can improve the reliability of decision-making models. Adding new sensors can further expand the analytical scope and enable multilayered assessment of structural behavior. Moreover, moving toward long-term monitoring and trend analysis can facilitate the prediction of structural changes and the identification of gradual patterns.

At the system level, there are also numerous opportunities for development. An important step is integrating the system with cloud-based processing platforms for scalable data storage, management, and analysis. Developing web-based dashboards to provide fast, multi-user access to information, as well as designing automated alert systems to notify maintenance personnel in real time, can significantly enhance the operational value of the system. Furthermore, extending the system for use in other structures and industrial facilities—such as steel structures or similar bridges—creates opportunities for broader application and generalization of the tool.

Finally, from a fundamental research perspective, it is recommended that the system's effectiveness be evaluated under real conditions and under the supervision of field experts, with results analyzed comparatively against industrial methods and standards. Additionally, developing adaptive threshold models, trend-analysis methods, and predictive-maintenance approaches can elevate the system to a higher level of intelligence and efficiency. Collectively, these pathways provide a rich framework for future research and can significantly enhance the capabilities of structural health monitoring in upcoming engineering projects.

6.5 Final Conclusion

Structural health monitoring of bridges and infrastructure plays a fundamental role in ensuring public safety and preventing catastrophic failures, and the results of this research demonstrate that developing a lightweight, reliable, and real-time tool can be an effective step in this direction. The system designed in this thesis, by providing an interactive environment for data visualization and by incorporating a scalable architecture suitable for industrial projects, has successfully met the primary operational requirements for real-time monitoring of structural behavior. This tool can serve as a solid foundation for the development of more advanced and comprehensive systems in the future, and it is hoped that in subsequent versions—through the use of advanced analytical methods and capabilities based on larger datasets—it will play a significant role in enhancing the safety and maintenance management of structures.

Bibliography

[1] Charles R Farrar and Keith Worden. An introduction to structural health monitoring. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):303–315, 2007.

[2] Bangcheng Zhang, Yuheng Ren, Siming He, Zhi Gao, Bo Li, and Jingyuan Song. A review of methods and applications in structural health monitoring (shm) for bridges. *Measurement*, 245:116575, 2025.

[3] Tinglong Jiang, Qi Xiang, and Zeming Li. Review of structural health monitoring in bridges. In *Advances in Frontier Research on Engineering Structures*. IOS Press, 2023.

[4] Branko Glisic. Long-term monitoring of civil structures and infrastructure using long-gauge fiber optic sensors. In *2019 IEEE SENSORS*, 2019.

[5] Seyyedbehrad Emadi, Seyedmilad Komarizadehasl, and Ye Xia. Application of intelligent low-cost accelerometers for bridge monitoring with a deep learning approach. *Structural Control and Health Monitoring*, 2025, 04 2025.

[6] Zhanxiong Ma, Jaemook Choi, and Hoon Sohn. Structural displacement sensing techniques for civil infrastructure: A review. *Journal of Infrastructure Intelligence and Resilience*, 2(3):100041, 2023.

[7] Jerome Peter Lynch. An overview of wireless structural health monitoring for civil structures. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):345–372, 12 2006.

[8] Marc Lizana and Joan R Casas. Optimal sensor placement methods and criteria in dynamic testing-comparison and implementation on a pedestrian bridge. *vectors*, 1:1, 2021.

[9] Wenhao Chai, Yaxun Yang, Haibo Yu, Fuli Yang, and Zhikui Yang. Optimal sensor placement of bridge structure based on sensitivity-effective independence method. *IET Circuits, Devices & Systems*, 16(2):125–135, 2022.

[10] Zhiyan Sun, Mojtaba Mahmoodian, Amir Sidiq, Sanduni Jayasinghe, Farham Shahrivar, and Sujeeva Setunge. Optimal sensor placement for structural health monitoring: A comprehensive review. *Journal of Sensor and Actuator Networks*, 14(2), 2025.

[11] Muhammad Fawad, Marek Salamak, Grzegorz Poprawa, Kalman Koris, Marcin Jasinski, Piotr Lazinski, Dawid Piotrowski, Muhammad Hasnain, and Michael Gerges. Automation of structural health monitoring (shm) system of a bridge using bimification approach and bim-based finite element model development. *Scientific Reports*, 13(1):13215, 2023.

[12] Zhanxiong Ma, Kyuwon Han, Jaemook Choi, Jigu Lee, Ohjun Kwon, Hoon Sohn, Jingxiao Liu, Doyun Hwang, Jatin Aggarwal, Haeyoung Noh, Enjian Cai, and Yi Zhang. Development and field deployment validation of a low-cost and high-precision displacement sensing system by fusing millimeter-wave radar and accelerometer. *Engineering Structures*, 321:118926, 2024.

[13] Gang Chen, Weixiang Shi, Lei Yu, Jizhuo Huang, Jiangang Wei, and Jun Wang. Wireless sensor placement optimization for bridge health monitoring: A critical review. *Buildings*, 14(3), 2024.

[14] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 254–263, 2007.

[15] Jiannong Cao and Xuefeng Liu. *Wireless Sensor Networks for Structural Health Monitoring*. 01 2016.

[16] Alvaro Araujo, Jaime Garcia-Palacios, Javier Blesa, Francisco Tirado, Elena Romero, Avelino Samartin, and Octavio Nieto-Taladriz. Wireless measurement system for structural health monitoring with high time-synchronization accuracy. *IEEE Transactions on Instrumentation and Measurement*, 61(3):801–810, 2012.

[17] Gunther Steenackers and Patrick Guillaume. Structural health monitoring of the z-24 bridge in presence of environmental changes using modal analysis.

Conference Proceedings of the Society for Experimental Mechanics Series, 01 2005.

[18] Mahendra Singh, Nandan Harsh, and Surot Thangjitham. Thermal environmental effects on modal parameters and health monitoring of bridge structures. *IABSE Symposium Report*, 96:21–30, 01 2009.

[19] Bart Peeters and Guido De Roeck. One-year monitoring of the z24-bridge: environmental effects versus damage events. *Earthquake engineering & structural dynamics*, 30(2):149–171, 2001.

[20] Adam Noel, Abderrazak Abdaoui, Ahmed Badawy, Tarek El-Fouly, Mohamed Ahmed, and Mohamed Shehata. Structural health monitoring using wireless sensor networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, PP:1–1, 04 2017.

[21] Xiao Yu, Yuguang Fu, Jian Li, Jianxiao Mao, Tu Hoang, and Hao Wang. Recent advances in wireless sensor networks for structural health monitoring of civil infrastructure. *Journal of Infrastructure Intelligence and Resilience*, 3(1):100066, 2024.

[22] Omar S. Sonbul and Muhammad Rashid. Towards the structural health monitoring of bridges using wireless sensor networks: A systematic study. *Sensors*, 23, 2023.

[23] Antonia M. Kohl, Kim-Denise Clement, Jens Schneider, Andrei Firus, and Geert Lombaert. An investigation of dynamic vehicle-bridge interaction effects based on a comprehensive set of trains and bridges. *Engineering Structures*, 279:115555, 2023.

[24] Maja Kreslin, Peter Češarek, Aleš Žnidarič, Darko Kokot, Jan Kalin, and Rok Vezočnik. Vehicle–bridge interaction modelling using precise 3d road surface analysis. *Sensors*, 24(2), 2024.

[25] Lu Deng, Yang Yu, Qiling Zou, and C. S. Cai. State-of-the-art review of dynamic impact factors of highway bridges. *Journal of Bridge Engineering*, 20(5):04014080, 2015.

[26] Lina Ding, Hong Hao, and Xinqun Zhu. Evaluation of dynamic vehicle axle loads on bridges with different surface conditions. *Journal of Sound and Vibration*, 323(3):826–848, 2009.

[27] Omar Mohammed, Arturo Gonzalez, and Daniel Cantero. Dynamic impact of heavy long vehicle with equally spaced axles on short-span highway bridges. *The Baltic Journal of Road and Bridge Engineering*, 13:1–13, 03 2018.

[28] Helu Yu, Bin Wang, Yongle Li, Yankun Zhang, and Wei Zhang. Road vehicle-bridge interaction considering varied vehicle speed based on convenient combination of simulink and ansys. *Shock and Vibration*, 2018:1–14, 07 2018.

[29] Bin Yang and Siyuan Sun. Real-time structural health monitoring system based on streaming data. *SMART STRUCTURES AND SYSTEMS*, 28:275–287, 08 2021.

[30] Kaya Yavuz. Real-time analysis and interpretation of continuous data from structural health monitoring (shm) systems. *Bulletin of Earthquake Engineering*, 13, 03 2014.

[31] Yilin Xie, Xiaolin Meng, Dinh Tung Nguyen, Zejun Xiang, George Ye, and Liangliang Hu. A discussion of building a smart shm platform for long-span bridge monitoring. *Sensors*, 24(10), 2024.

[32] J.M.W Brownjohn. Structural health monitoring of civil infrastructure. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 365(1851):589–622, 12 2006.

[33] Rune Brincker, Lingmi Zhang, and Palle Andersen. Modal identification from ambient responses using frequency domain decomposition. 2000.

[34] S. Doebling, Charles Farrar, Michael Prime, and D. Shevitz. A review of damage identification methods that examine changes in dynamic properties. *Shock and Vibration Digest*, 30, 01 1998.

[35] Keith Worden and Graeme Manson. The application fo machine learning to structural healt monitoring. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 365:515–37, 12 2006.

[36] Alireza Entezami, Hassan Sarmadi, Behshid Behkamal, and Stefano Mariani. Big data analytics and structural health monitoring: A statistical pattern recognition-based approach. *Sensors*, 20(8), 2020.

[37] Arnulf Hagen and Trond Michael Andersen. Asset management, condition monitoring and digital twins: damage detection and virtual inspection on a

reinforced concrete bridge. *Structure and Infrastructure Engineering*, 20(7-8):1242–1273, 2024.

[38] Natasha Vipond, Abhinav Kumar, Joseph James, Frederick Paige, Rodrigo Sarlo, and Zhiwu Xie. Real-time processing and visualization for smart infrastructure data. *Automation in Construction*, 154:104998, 2023.

[39] Sanjeev Bhatta and Ji Dang. Use of iot for structural health monitoring of civil engineering structures: a state-of-the-art review. *Urban Lifeline*, 2:1–19, 11 2024.

[40] Xiaofei Li, Yuyu Xiao, Hainan Guo, and Jisong Zhang. A bim based approach for structural health monitoring of bridges. *KSCE Journal of Civil Engineering*, 26, 10 2021.

[41] Maryam Nasim, Abbas Rajabifard, Yiqun Chen, and Bijan Samali. A demonstration of a digital twin framework for structural health monitoring: Application to bridge infrastructures. *Journal of Infrastructure Intelligence and Resilience*, 5(1):100184, 2026.

[42] Alberto Armijo and Diego Zamora-Sánchez. Integration of railway bridge structural health monitoring into the internet of things with a digital twin: A case study. *Sensors*, 24(7), 2024.

[43] Marlon Aguero, Derek Doyle, David Dennis Lee Mascarenas, and Fernando Moreu. Visualization of real-time displacement time history superimposed with dynamic experiments using wireless smart sensors (WSS) and augmented reality (AR). *CoRR*, abs/2110.08700, 2021.

[44] Furkan Luleci, Liangding Li, Jiapeng Chi, Dirk Reiners, Carolina Cruz-Neira, and Fikret Necati Catbas. Structural health monitoring of a foot bridge in virtual reality environment. *CoRR*, abs/2112.03470, 2021.

[45] Omar Awadallah, Katarina Grolinger, and Ayan Sadhu. Augmented reality-based smart structural health monitoring system with accurate 3d model alignment. *Journal of Infrastructure Intelligence and Resilience*, 5(1):100186, 2026.

[46] Muhammad Fawad, Marek Salamak, Qian Chen, and Kalman Koris. Development of an immersive digital twin framework to support infrastructure management: a case study of bridge asset health monitoring. In *ISARC. Proceedings*

of the International Symposium on Automation and Robotics in Construction, volume 41, pages 808–814. IAARC Publications, 2024.

[47] Calin Boje, Annie Guerriero, Sylvain Kubicki, and Yacine Rezgui. Towards a semantic construction digital twin: Directions for future research. *Automation in Construction*, 114:103179, 2020.

[48] Yuqian Lu, Chao Liu, Kevin I-Kai Wang, Huiyue Huang, and Xun Xu. Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robotics and Computer-Integrated Manufacturing*, 61:101837, 2020.

[49] Behrouz Shafei and Ibrahim Odeh. Automated assessment of defects in bridge structures. 2025.

[50] Bentley Systems. itwin platform. `https://www.bentley.com/software/itwin-platform/`, 2025. Accessed: Dec 29, 2025.

[51] Yiqun Chen, Erfan Shooraj, Abbas Rajabifard, and Soheil Sabri. From ifc to 3d tiles: An integrated open-source solution for visualising bims on cesium. *ISPRS International Journal of Geo-Information*, 7(10), 2018.

[52] Mohammed Eunus Ali, Muhammad Aamir Cheema, Tanzima Hashem, Anwaar Ulhaq, and Muhammad Ali Babar. Enabling spatial digital twins: Technologies, challenges, and future research directions. *PFG–Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, 92(6):761–778, 2024.

[53] Muhammad Fawad, Marek Salamak, Muhammad Usman Hanif, Kálmán Koris, Muhammad Ahsan, Hadiya Rahman, Michael Gerges, and Mostafa Mohamed Salah. Integration of bridge health monitoring system with augmented reality application developed using 3d game engine–case study. *IEEE Access*, 12:16963–16974, 2024.

[54] Cory Quammen. Scientific data analysis and visualization with python, vtk, and paraview. In *SciPy*, 2015.

[55] Naveen Shiju Joseph, R Panneer Selvam, and Andre S Hanley. Visualization using open-source software paraview for finite element class and research. In *2025 ASEE Midwest Section Conference*, 2025.

[56] Certificato di collaudo – ponte po via nord. Technical report, Nplus S.r.l., 2023.

[57] General technical report rt01 – monitoring system for ponte po via nord. Technical report, Nplus S.r.l., 2023.

[58] T01 – layout dell'impianto di monitoraggio strutturale dinamico, 2023.

[59] T02 – layout dell'impianto di monitoraggio strutturale dinamico, 2023.