DIGITAL TWINS REPORT

ASSIGNMENT

ME F434 – Digital Twins, Semester II 2024 – 2025

By

Ritabrata Chakraborty 2022A4PS0843P

Devansh Parmar 2022A4PS0466P

Rishav Raj Verma 2022A4PS0738P

Under the supervision of :

Dr. Madhurjya Dev Choudhury (IC)

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

25 April 2025

ACKNOWLEDGEMENTS

I sincerely thank Dr. Madhurjya Dev Choudhury for your invaluable guidance and support throughout my research. Your insightful feedback and constructive criticism have been instrumental in shaping my research work. Your expertise and experience in the field have been of great help in the research process.

Once again, I express my heartfelt thanks to all those who have contributed to the successful completion of this project.

TABLE OF CONTENTS

1 Introduction

This project uses a Digital Twin framework to build a real-time monitoring system for an industrial bearing. At its core, the system replicates the behaviour of a real bearing by using data from a historical run-to-failure experiment conducted on a wind turbine. Over 50 days, the bearing experienced progressive deterioration due to an inner race fault, eventually leading to its failure. Using this dataset, the project aims to simulate how a real-time monitoring solution would perform if deployed in an industrial setting.

1.1 Objectives

The primary objective of this work is to simulate a real-time monitoring system using a Raspberry Pi to imitate live data collection. Once the data is streamed, it undergoes signal processing to extract meaningful insights into the health of the bearing. These insights are then fed into a virtual 3D bearing model, which dynamically reflects the bearing's condition by changing its appearance (colour) based on its health. This entire system is integrated into a real-time dashboard, allowing users to visualise the bearing status through time-series plots and interpret fault progression.

1.2 Dataset Description

The dataset used in this project originates from a wind turbine bearing experiment where the bearing was allowed to operate until failure. The data spans 50 consecutive days, from March 7 to April 25, and includes two primary signals collected each day — vibration and tachometer readings. The vibration signals were captured at a high sampling frequency of 97,656 Hz, providing detailed insights into mechanical oscillations. Throughout the test, an inner race fault gradually formed, leading to eventual bearing failure, offering a realistic scenario for degradation modelling.

1.3 System Architecture

The system is designed in three interconnected layers: data acquisition, processing, and visualisation. For data acquisition, a Raspberry Pi is set up to mimic real-time streaming by sequentially serving vibration and tachometer data to a connected laptop or cloud service. Once the data is received, it is processed to extract diagnostic features such as RMS, kurtosis, and frequency-domain metrics, forming a comprehensive health index. This health index is then used to drive a Digital Twin — a 3D virtual model of the bearing that visually represents its condition. Finally, a dashboard presents the processed data, plots the health trajectory, and updates the virtual model in real-time, enabling users to monitor the system intuitively.

2. Methodology

2.1 Preparation of Health Indicator

2.1.1 Data Loading and Visualization of Sample Signals

The initial stage of the project involves organising and visualising the vibration and tachometer signal data. A designated folder, 'Bearing CSV', stores all the relevant .csv files. The script first scans the folder and filters the files based on their suffixes to identify and separate the vibration files from the tachometer files. To ensure data integrity, the code then pairs the files with a common base name so that each day's vibration reading corresponds accurately with its respective tachometer reading.

A range of dates from March 7, 2013, to April 25, 2013, is generated to align the file pairs chronologically. The vibration data, represented as amplitude over samples, provides insight into mechanical oscillations, while the tachometer data shows the corresponding rotational speed in RPM. This visualisation helps gain a preliminary understanding of how the bearing behaviour evolves.

2.2 Feature Extraction using Time and Frequency Domain Analysis

To track the health of the bearing over time, the system extracts a combination of time-domain and frequency-domain features. Basic statistics such as mean, standard deviation, skewness, kurtosis, RMS, peak-to-peak, and energy are computed in the time domain. Additionally, specialised diagnostic metrics like crest, impulse, shape, and margin factors are derived to capture vibration patterns indicative of emerging faults.

The method leverages Spectral Kurtosis (SK) in the frequency domain using a custom implementation of the Short-Time Fourier Transform (STFT). The SK helps identify transient events and non-stationary behaviours, often early signs of mechanical failure. A 3D visualisation of spectral kurtosis values across frequency bins and days shows how the frequency content of the vibration signal evolves as the bearing wears out.

The daily feature vectors — comprising both time-domain statistics and summary statistics of the SK curve — are compiled into a DataFrame, indexed by date. This forms a robust and interpretable dataset, suitable for further modelling or machine learning applications.

2.3 Feature Smoothing and Monotonicity-Based Selection

To reduce noise and highlight long-term trends, a causal moving average smoothing filter is applied to the extracted features using a 6-day rolling window. This approach ensures that the current value is influenced only by past data, maintaining real-time applicability.

After smoothing, monotonicity analysis is carried out to identify features that show consistent increasing or decreasing trends, particularly over the first 20 days of degradation. A monotonicity score is calculated for each feature, measuring how directional the trend is.

Features with scores above a certain threshold (0.3 in this case) are deemed more informative and are retained for further analysis. This step ensures that the final model only relies on stable and predictive features, improving its robustness and accuracy.

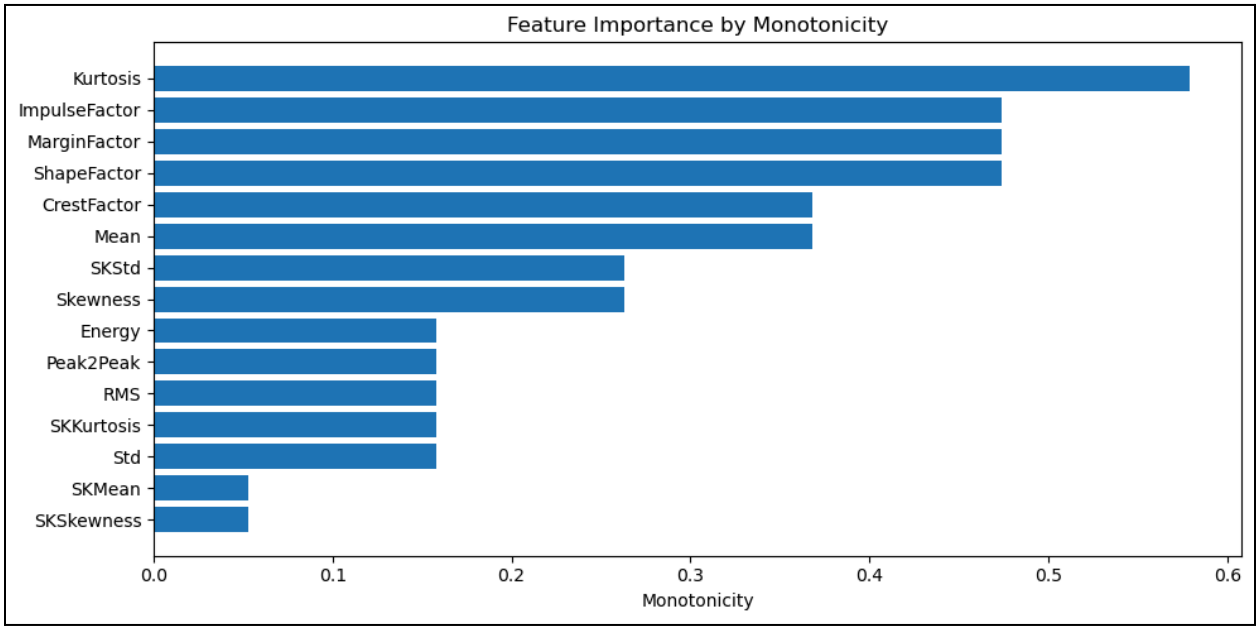

Figure: Feature Importance by Monotonicity

2.4 Dimensionality Reduction and Health Indicator Formulation using PCA

Principal Component Analysis (PCA) is applied to simplify and interpret the multivariate feature space. The process begins by standardising the selected features, which were previously filtered based on their monotonicity, using the mean and standard deviation derived from the first 20 days of data. This ensures that all features contribute equally to the PCA, regardless of their original scale or magnitude.

A PCA model is then fitted on this normalised training dataset to identify the principal directions of variance. The first two principal components are retained, capturing the most significant patterns in the data. The entire 50-day dataset is projected into this newly defined 2D PCA space using the same standardisation parameters. This transformation helps reveal any underlying structure or progression in the bearing's condition that might not be obvious in the original feature space.

The resulting PCA projections are visualised as a scatter plot, with each point representing a day's data and colour-coded by time. A smooth transition across the plot indicates a systematic and progressive degradation of the bearing. This projection selects the first principal component (PCA1) — which explains most of the variance, as the health indicator. By plotting PCA1 over time, a clear and continuous trend in the bearing's health can be observed. This trend correlates

strongly with the inner race fault development, providing a compact yet powerful representation of the bearing's degradation trajectory.

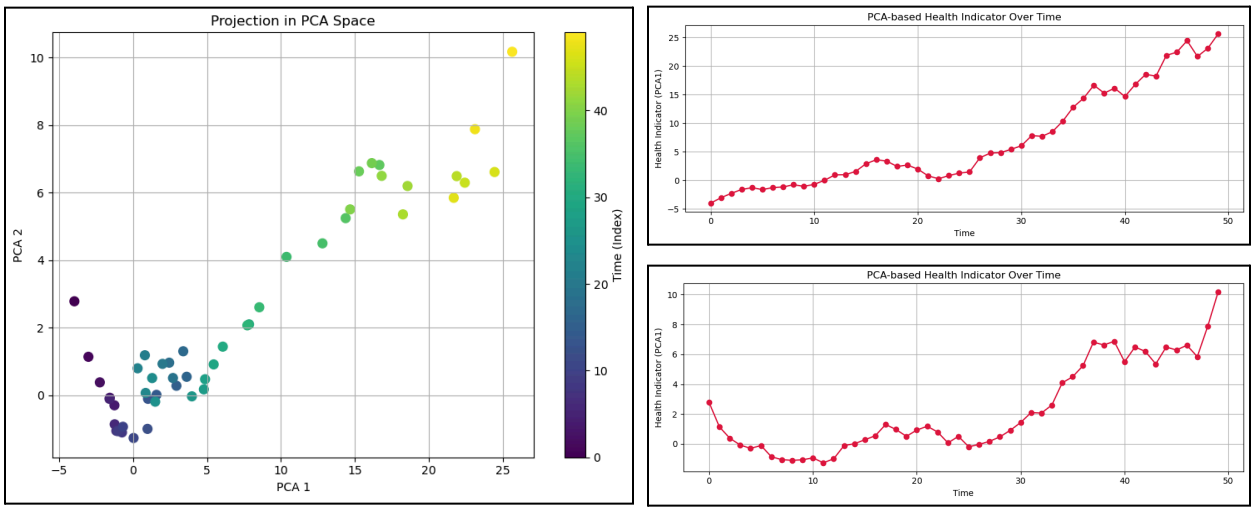

Figure: Projection in PCA Space (left), PCA1 vs. Time (top right), PCA2 vs. Time (bottom right)

2.5 Remaining Useful Life (RUL) Estimation Using Exponential Degradation Modelling

In the final phase of the methodology, the PCA-based health indicator is leveraged to estimate the bearing's remaining useful life (RUL). The health indicator is first normalised to start from zero, and the final value (at failure) is used as the degradation threshold.

A stochastic exponential degradation model predicts how the health indicator evolves. The model is defined by parameters ϕ, θ, and β, where:

- ϕ is the asymptotic minimum health level,
- θ is a scaling parameter, and
- β controls the rate of degradation.

The model assumes that the health indicator follows an exponential trajectory with Gaussian noise added to account for uncertainty. Using early observations, parameters θ and β are estimated dynamically at each time step by fitting a linear model in log-space.

At every time step, the model simulates future health indicator values and estimates the time to reach the failure threshold, which is interpreted as the estimated RUL. Additionally, a confidence interval (CI) is calculated by evaluating when the predicted trajectory reaches 95% and 105% of the threshold. These CIs provide a probabilistic range for RUL, reflecting modelling uncertainty.

The results are visualised in two parts:

1. The top plot shows the observed health indicator over time alongside the model's exponential fit and the defined threshold.
2. The bottom plot compares the estimated RUL against the true RUL (known from data) and shades the confidence interval to show the prediction spread.

This probabilistic modelling approach enables point estimation and uncertainty quantification, making it well-suited for real-time predictive maintenance in safety-critical applications like wind turbine bearings.

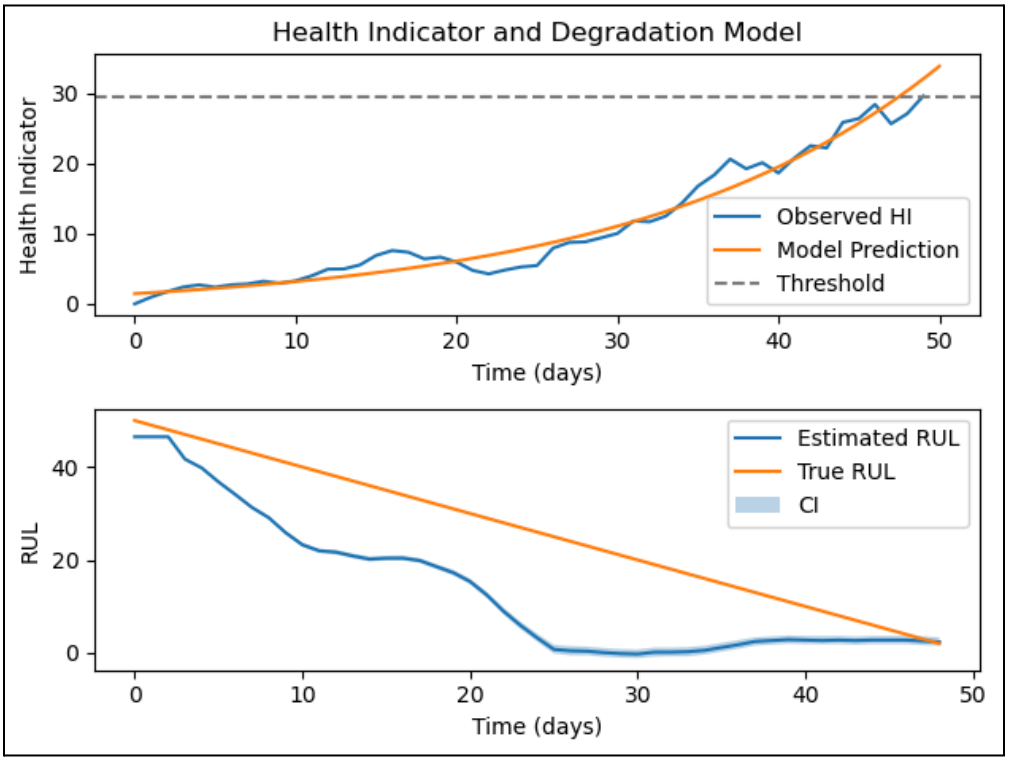

Figure: Health Indicator v/s Time (top), RUL v/s Time (bottom)

2.2 Data Transfer between Raspberry Pi and Fusion Add-in

The system comprises two primary components: a Flask server responsible for real-time data serving and a Fusion 360 add-in designed to update the model's appearance based on the received data.

2.2.1 Flask Server for Real-Time Data Serving

The server was developed using the Flask server for Python. It duplicates a real-time data stream by reading the health indicator values from a CSV file every 2.5 seconds and sending the most current value to the client. For demonstration purposes, a Raspberry Pi might be readily integrated as the data source in a more complex solution, mimicking the behaviour of sensor data being pushed from a physical device.

2.2.2 Fusion 360 Add-In Implementation

The Fusion 360 side is composed of a Python add-in that polls the Flask server every 2.5 seconds and adjusts the model's appearance based on the value it receives. Key functions and classes within the add-in include:

- run(context): Initialises the system, registers the event handler, and fetches the initial data value.
- pollFlaskServerThread(): Runs a background thread that periodically fetches data from the Flask server using libraries like urllib.request and then fires a custom event, passing the retrieved data.
- FlaskUpdateEventHandler: Responds to the custom event triggered by the background thread and calls the logic to update the model's appearance.
- updateAppearanceFromValue(value): Applies a predefined appearance (red or green paint in this case, chosen from Fusion's built-in material library) to all visible components within the root component of the model based on the received data value. In particular, a green enamel look is mapped to values below a certain threshold, while values beyond this threshold trigger a red enamel appearance.
- Real-time behaviour is achieved through Fusion 360's registerCustomEvent() and Python's threading mechanisms. Significantly, event callbacks limit any Fusion API calls, particularly those that alter the geometry or appearance of the model, to the context of the main thread. The graphics are refreshed using app.activeViewport.refresh() to ensure immediate visual feedback.

2.2.3 Real-time Data Flow:

The core of the system is the asynchronous data flow:

1. The Flask server continuously provides simulated sensor data (vibration values).
2. The Fusion 360 add-in's background thread periodically requests this data from the /value endpoint.
3. The add-in launches a custom event with the data value passed as event arguments when it receives new data.
4. The FlaskUpdateEventHandler then extracts the data value and calls the updateAppearanceFromValue function to modify the model's appearance within Fusion 360.
5. Finally, the viewport is refreshed to display the changes.

By doing this, Fusion 360 can respond to modifications in the data supplied by the Flask server almost instantly, eliminating the need for the user to initiate updates or restart the add-in explicitly.

5 Results

5.1 Post Processing

After processing the data, we applied Principal Component Analysis (PCA) to the highly monotonic features. This allowed us to derive a single health indicator—one principal component—that best represents the degradation trend. This health indicator can estimate the onset of bearing damage by analysing the time at which vibration signal deviations begin to emerge.

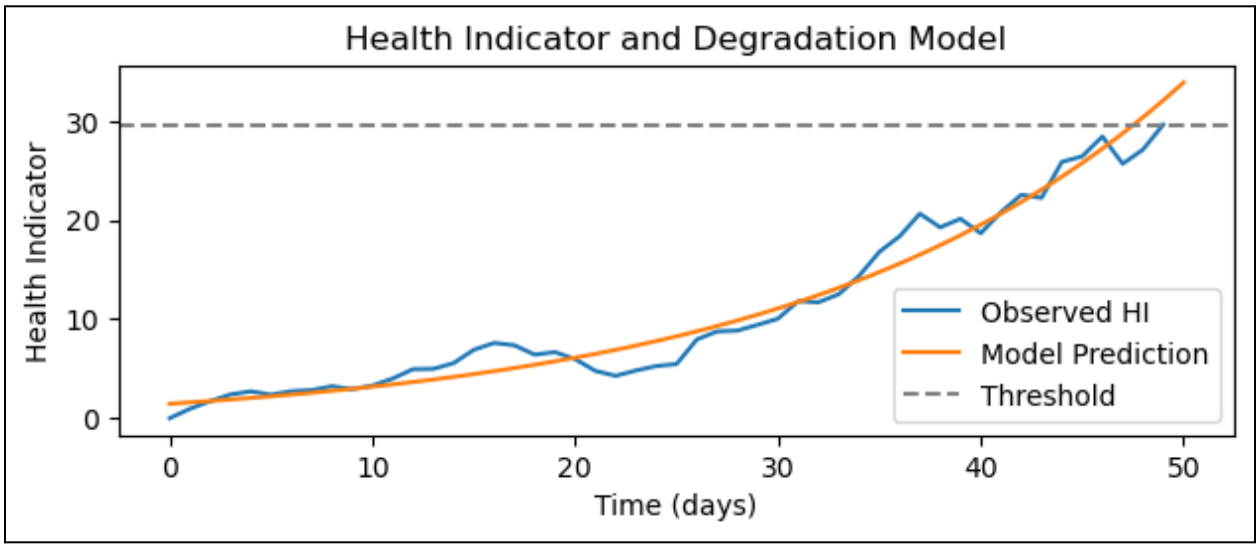

Figure: Health Indicator vs Time

5.2 Fault Visualisation

To visualise faults, we colour the data points red when the observed value of the health indicator exceeds the predicted value from the fitted exponential degradation model.

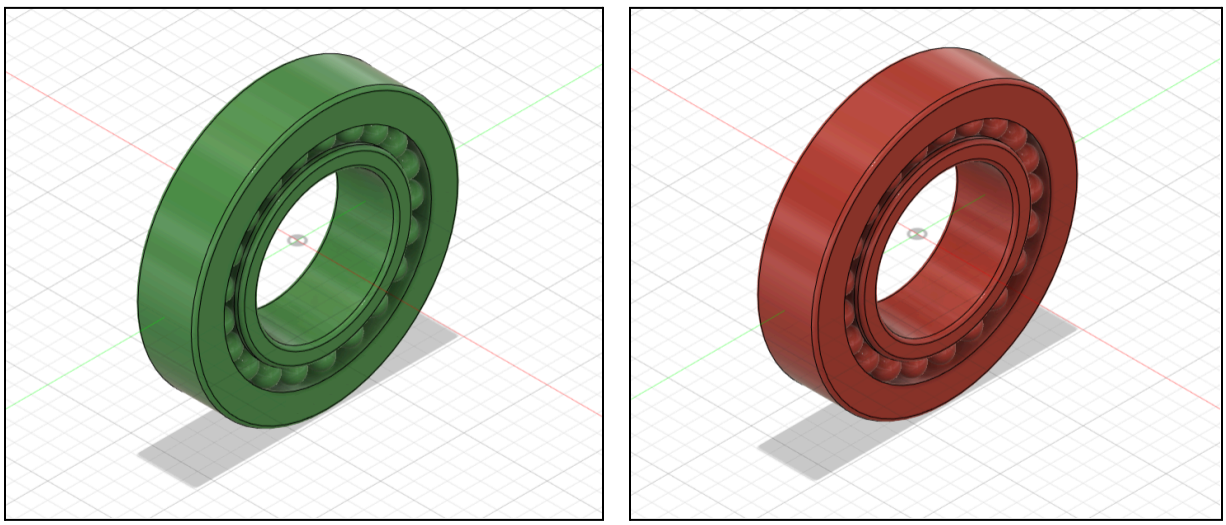

Figure: Healthy Bearing (left) and Faulty Bearing (right)

6 Conclusion

A reliable method for creating an efficient health indicator (HI) from vibration data is provided by combining smoothing techniques, Principal Component Analysis (PCA), and monotonicity analysis. This technique produces a transparent and trustworthy depiction of a system's health using monotonicity analysis to isolate the main deterioration features, PCA to reduce dimensionality, and smoothing to stabilise the Health Indicator trend.

The health indicator was then used to demonstrate fault preparation in the ball bearing. Python-based libraries like Flask, adsk and traceback were used and learned to form the code. Flask allows us to demonstrate the feasibility of integrating external data sources, such as a Raspberry Pi 4, with Autodesk Fusion 360 to drive real-time visual updates. Combining a Flask web server, the Fusion 360 API, and custom event handling provides a flexible and powerful way to create dynamic and interactive design environments.