TELEMETRY ANOMALY DETECTION SYSTEM

1. INTRODUCTION

Telemetry data plays a crucial role in the success and safety of space missions. Throughout a mission, a complex network of sensors continuously monitors vital parameters such as temperature, pressure, voltage, and vibration within both the spacecraft and astronaut environments. These measurements provide real-time insights into the operational status and health of spacecraft systems, enabling mission control to make informed decisions.

Detecting anomalies promptly in telemetry data is essential to prevent failures that could lead to mission aborts, equipment damage, or even loss of human life. However, the challenge lies in distinguishing between normal fluctuations and genuine anomalies within large volumes of data transmitted at high frequency. The Telemetry Anomaly Detection System is designed precisely to address this challenge by providing automated monitoring, early detection, and clear reporting of unusual telemetry patterns.

The primary objectives of this system include:

Reading telemetry data from CSV files: The system accepts pre-recorded or live-streamed telemetry data formatted as CSV, enabling straightforward data ingestion.

Anomaly detection through combined methods: It employs a hybrid approach involving threshold-based checks to catch values exceeding predefined safe limits, alongside statistical analysis using z-scores to identify outliers within the data distribution.

Visualization of telemetry trends and anomalies: The system generates intuitive graphical plots, allowing users to observe parameter behaviors and pinpoint detected anomalies visually.

Issuance of alerts: Upon detection, the system notifies users through audio and visual alerts to facilitate swift response.

Output generation for documentation: Detailed reports summarizing anomalies and system performance are created for record-keeping and further analysis.

By integrating these capabilities, the Telemetry Anomaly Detection System aims to enhance mission reliability, safeguard astronaut well-being, and provide engineers and analysts with powerful tools for real-time and post-mission data evaluation.

2. METHODOLOGY

The anomaly detection system employs a structured methodology combining threshold-based rules and statistical analysis to accurately identify irregularities in telemetry data.

DATA LOADING

Telemetry data is ingested from CSV files utilizing the **pandas** library, which offers efficient data handling and manipulation capabilities. Each CSV file contains sequential readings for key parameters such as temperature and pressure, indexed by timestamps. The system loads this data into a DataFrame, facilitating subsequent processing steps including filtering, threshold checks, and statistical calculations.

THRESHOLD-BASED DETECTION

The system applies predefined safety thresholds to monitor temperature and pressure values:

- **Temperature:** An anomaly is flagged if a recorded temperature value falls outside the range of 15°C to 75°C.
- **Pressure:** Anomalies arise when pressures are measured below 30 kPa or exceed 100 kPa.

These limits represent operational boundaries derived from spacecraft design specifications and environmental tolerances. Any telemetry point breaching these thresholds receives an initial anomaly flag.

STATISTICAL OUTLIER DETECTION USING Z-SCORE

In addition to fixed thresholds, the system implements a statistical approach leveraging the z-score to capture more subtle anomalies:

$z=(x-\mu)/\sigma$

Where:

- (x) is the observed parameter value,
- (\mu) is the mean of that parameter over the dataset,
- (\sigma) is the standard deviation of the parameter.

A telemetry reading is considered an outlier if its z-score magnitude ($|z|$) exceeds 3, indicating it lies beyond three standard deviations from the mean. This statistical filter helps identify rare or unexpected values that may still reside within threshold limits but differ significantly from usual patterns.

COMBINING DETECTION METHODS

Anomalies are conclusively flagged only when **both** conditions occur simultaneously: the value violates threshold limits **and** exhibits an extreme z-score. This dual-criteria approach reduces false positives by ensuring flagged events are not just borderline threshold breaches or statistical noise but genuine deviations warranting attention.

ANOMALY OUTPUT AND ALERTS

Detected anomalies are logged to a structured text file, documenting timestamps, parameter values, and anomaly types for audit and review purposes. Simultaneously, the system triggers voice alerts using **pyttsx3**, an offline text-to-speech library, which announces the presence of anomalies to operators, enabling rapid situational awareness without visual monitoring.

VISUALIZATION

To assist in intuitive understanding and analysis, **matplotlib** plots are generated showing telemetry trends over time. Anomalies are distinctly marked in red on these time series graphs, providing clear visual cues to facilitate swift identification and assessment by engineers and analysts. This graphical representation complements textual logs and audio alerts, enriching the multidimensional anomaly detection experience.

3. IMPLEMENTATION DETAILS

This section details the programming environment, key Python libraries, code structure, data formats, and instructions to run the Telemetry Anomaly Detection System, providing a comprehensive guide for developers and users.

PROGRAMMING ENVIRONMENT AND LIBRARIES

The system is implemented in Python 3 and relies on several major libraries to achieve its functionalities:

- **pandas:** Core library for data loading, manipulation, and filtering of telemetry datasets.
- **matplotlib:** Used for creating plots that visualize telemetry parameters and highlight anomalies.
- **tkinter:** Provides a graphical user interface (GUI) for user interaction, including file selection and parameter input.
- **pyttsx3:** An offline text-to-speech engine that delivers voice alerts upon anomaly detection.
- **logging:** Manages error tracking and system logs to enhance reliability and debugging.

These libraries combine to create an interactive, robust, and user-friendly anomaly detection application.

CODE STRUCTURE AND ORGANIZATION

The codebase is organized into five primary modules representing key functional components:

Module	Responsibilities
Data Handling	Reading CSV telemetry files into pandas DataFrames, validating input format, and preprocessing data.
Anomaly Detection	Implementing threshold checks and z-score statistical analysis to identify anomalies in telemetry data.
Output Handling	Writing detected anomaly records to structured text files and managing voice alerts via pyttsx3.

Module	Responsibilities
Visualization	Generating matplotlib time series plots with anomalies highlighted in red.
User Interface	Implementing Tkinter-based GUI for file selection, threshold inputs, running detection, and displaying results.

This modular organization facilitates easier maintenance, testing, and future extensions.

INPUT DATASET FORMAT

The expected input is a CSV file with the following three columns:

Column	Description
index	Sequential timestamp or record identifier
temperature	Temperature readings in degrees Celsius
pressure	Pressure readings in kilopascals (kPa)

A sample data excerpt:

index	temperature	pressure
1	22.5	85.0
2	24.1	90.2
3	80.3	110.5

Users must ensure the CSV adheres to this format without missing columns to enable proper processing.

OUTPUT FORMAT: ANOMALIES TEXT FILE

Detected anomalies are logged into a text file (`anomalies.txt` by default) with entries formatted as follows:

```
12 temperature 116.39
14 pressure 0.22
15 temperature 115.40
...
```

Each entry includes the record index, parameter name, recorded value, and the type of anomaly (whether threshold or combined threshold and z-score violation), enabling clear documentation and audit trails.

RUNNING THE SYSTEM

To run the Telemetry Anomaly Detection System, follow these steps:

1. **Install required dependencies:**

```
pip install pandas matplotlib pyttsx3
```

(Note: `tkinter` and `logging` come pre-installed with most Python distributions)

1. **Launch the GUI application:**

Run the main Python script (e.g., `telemetry_anomaly_detection.py`). The interface window will appear.

1. **Interact via GUI:**

- Use the **Browse** button to select the input CSV telemetry file.
- Enter threshold values for temperature and pressure if custom limits are preferred; otherwise, defaults are used.
- Click the **Run Detection** button to process data and detect anomalies.
- View generated matplotlib plots showing telemetry data with anomalies marked.
- Optionally, listen to voice alerts if anomalies are detected.
- Review the generated `anomalies.txt` file for detailed anomaly records.

This straightforward interaction workflow ensures users can efficiently load data, set parameters, analyze results visually, and receive prompt alerts.

4. ERROR HANDLING

The Telemetry Anomaly Detection System incorporates a robust error management strategy designed to enhance reliability, improve user experience, and facilitate debugging. The system anticipates and gracefully handles various error types associated with file operations, data integrity, input validation, and runtime exceptions.

FILE HANDLING ERRORS

When users select telemetry CSV files for processing, the system performs checks to confirm the existence and readability of the file. If a specified file is missing or inaccessible, a **FileNotFoundError** exception is raised. This triggers a user-friendly alert dialog that clearly indicates the error and prompts the user to upload a valid file. This prevents crashes due to invalid file paths or permissions.

Example handling:

```
try:
    data = pd.read_csv(file_path)
except FileNotFoundError:
    messagebox.showerror("File Error", "Selected file not found. Please choose a valid CSV file.")
    logging.error(f"FileNotFoundError: {file_path} not found.")
```

INVALID DATA DETECTION

The system validates telemetry data immediately after loading to ensure all expected values are numeric and present. Non-numeric entries or missing values raise a **ValueError**, which is caught and reported to the user via a clear, informative message. This prevents anomalous behavior or crashes caused by corrupted or improperly formatted data files.

Validation includes:

- Checking for non-numeric strings in temperature or pressure columns.
- Detecting NaN or empty cells.
- Confirming correct data types before proceeding with calculations.

THRESHOLD AND Z-SCORE INPUT VALIDATION

User inputs for threshold limits and z-score parameters are validated to prevent nonsensical or invalid configurations. For example, upper threshold values must be greater than lower thresholds, and z-score cutoff values must be positive numbers. Invalid entries generate validation errors, which the system catches and communicates back to the user, requesting correct input before running detection.

RUNTIME ERROR HANDLING

Certain calculations may produce runtime exceptions under edge cases. A notable example is division by zero when calculating z-scores if the standard deviation of a parameter is zero (indicating no variance in data). To handle this, the system wraps these operations in try-except blocks, logs the specific error, and informs the user with a graceful warning without terminating the program abruptly.

Example:

```
try:
    z_scores = (data[column] - mean) / std_dev
except ZeroDivisionError:
    messagebox.showwarning("Calculation Warning",
"Standard deviation is zero; z-score computation
skipped.")
    logging.warning("ZeroDivisionError: Standard
deviation zero for column '%s'.", column)
```

COMPREHENSIVE LOGGING

All errors—including exceptions, warnings, and validation failures—are logged to an `error_log.txt` file with timestamps and detailed messages. This facilitates post-incident diagnosis and maintenance by developers and support personnel. The logging configuration ensures that even if users do not report an issue, technical teams can review logs to identify recurring or critical problems.

This proactive and user-centric error handling approach ensures the Telemetry Anomaly Detection System remains robust, resilient, and accessible even when provided with imperfect inputs or faced with unexpected runtime conditions.

5. CHALLENGES AND SOLUTIONS

During the development of the Telemetry Anomaly Detection System, several key challenges were encountered that required targeted solutions to ensure performance, accuracy, and usability.

HANDLING LARGE TELEMETRY DATASETS

Telemetry data from space missions can be extensive, with millions of records generated continuously. Processing such large datasets efficiently is critical to maintaining system responsiveness. Initial attempts using iterative data processing methods resulted in slow performance and excessive memory consumption. To address this, the system leverages **pandas' vectorized operations**, which perform computations on entire columns or DataFrames simultaneously. This approach significantly improves processing speed and reduces memory overhead compared to row-wise processing.

For extremely large datasets that exceed available memory, the system supports data **chunking**, where the CSV file is read and processed in smaller portions sequentially. This strategy enables anomaly detection on datasets that are too large to load at once, while maintaining accuracy and manageable resource use.

BALANCING THRESHOLD SENSITIVITY

Another major challenge was selecting appropriate threshold values to identify anomalies without generating excessive false positives or missing critical events. Overly strict thresholds trigger frequent false alarms, burdening operators and potentially causing alert fatigue. Conversely, lenient thresholds risk overlooking genuine anomalies that may indicate dangerous conditions.

To overcome this, the system incorporates **user-controlled threshold tuning**, allowing users to customize the temperature and pressure limits according to mission requirements or sensor characteristics. Moreover, the combination of threshold checks with **z-score statistical analysis** refines detection by considering data distribution patterns rather than relying on fixed cutoffs alone. This hybrid approach balances sensitivity and specificity, reducing erroneous alerts while ensuring true anomalies are detected.

ENHANCING ROBUSTNESS AND USABILITY

These solutions collectively enhance the robustness of the anomaly detection system by improving processing efficiency, scalability, and detection accuracy. Additionally, user-configurable parameters contribute to usability by providing flexibility and control, enabling operators and analysts to adapt the system dynamically for different mission phases or data profiles. This adaptability positions the system as a practical tool for real-world space telemetry monitoring challenges.

6. TEST CASE RESULTS

Three primary test cases were conducted to evaluate the accuracy and reliability of the Telemetry Anomaly Detection System under varying input conditions.

TEST CASE 1: ANOMALY DETECTION WITH KNOWN OUTLIERS

Input data consisted of 6 records containing temperature and pressure values, including two clear anomalies: an excessively high temperature (exceeding 100°C) and an abnormally low pressure (below 0.5 atm). The system successfully identified both anomalies using combined threshold and z-score criteria. These events were accurately logged into the anomalies text file with correct timestamps and parameter details. Visualization plots distinctly highlighted these anomalies in red, and audio alerts were triggered accordingly.

TEST CASE 2: NORMAL DATA WITHOUT ANOMALIES

A dataset with temperature and pressure readings all within defined operational limits was processed. The system correctly detected the absence of anomalies, producing no entries in the anomaly log and generating plots showing smooth telemetry trends without flagged points. No voice alerts were issued, confirming the system’s ability to avoid false positives when data conforms to expected patterns.

TEST CASE 3: HANDLING INVALID NON-NUMERIC INPUT

A test input containing non-numeric temperature entries was supplied to assess error handling robustness. The system promptly raised a `ValueError` upon validating the data. This exception was caught, generating an error message displayed to the user via the GUI and a corresponding entry in the error log file. The program terminated gracefully without proceeding to detection, demonstrating effective validation and fail-safe behavior in response to corrupted input.

These test results validate that the system reliably detects genuine anomalies, avoids false alarms with normal data, and robustly handles invalid inputs, aligning with the intended design objectives for operational accuracy and stability.

7. FUTURE IMPROVEMENTS

To further enhance the Telemetry Anomaly Detection System, several advancements are proposed:

Integration of Machine Learning Models: Incorporating unsupervised algorithms such as Isolation Forest and DBSCAN can improve anomaly detection by capturing complex, nonlinear relationships and subtle deviations that threshold and z-score methods may miss. These models adapt dynamically to evolving telemetry patterns, offering more nuanced insights.

Real-Time Telemetry Processing: Enabling continuous streaming data analysis would provide immediate anomaly detection during active missions. This capability ensures timely alerts, giving operators the opportunity to respond swiftly to critical events and mitigate risks.

User Interface Enhancements: Upgrading the GUI with interactive features—such as zoomable time-series graphs, real-time anomaly highlighting, and sliders for dynamic threshold adjustment—would increase user engagement and operational flexibility. Real-time feedback on detection results will improve situational awareness and ease parameter tuning.

Implementing these improvements will strengthen the system's effectiveness, scalability, and usability in complex space mission environments.

8. CONCLUSION

The Telemetry Anomaly Detection System effectively combines threshold-based checks with statistical z-score analysis to identify anomalies in spacecraft telemetry data. This hybrid approach ensures accurate detection by capturing both clear threshold breaches and subtle statistical outliers. The system enhances mission safety through comprehensive visualizations and audible alerts, enabling operators to promptly recognize and respond to irregularities.ts modular, extensible design positions it as a valuable asset for ongoing and future space telemetry analysis applications.

9. REFERENCES

• Pandas Documentation: https://pandas.pydata.org/pandas-docs/stable/
• Matplotlib Documentation: https://matplotlib.org/stable/contents.html
• Pyttsx3 Documentation: https://pyttsx3.readthedocs.io/en/latest/
• kinter Documentation: https://docs.python.org/3/library/tkinter.html
• ChatGPT: https://chatgpt.com