**Spacecraft Telemetry Anomaly Detection System**

Team name: CODE CORTEX

KVG College of Engineering

# Table of Contents

1. 1. Introduction
2. 2. Objective
3. 3. Methodology
4. 4. Flowchart
5. 5. Implementation Details
6. 6. Code Structure
7. 7. How to Run the Code
8. 8. Dataset Handling
9. 9. Parsing Algorithm
10. 10. Output Format
11. 11. Challenges and Simulation
12. 12. Test Case Result
13. 13. Future Implementation
14. 14. Visualization Enhancement
15. 15. User Interface Improvement
16. 16. Reference
17. 17. Dataset Resource

**Spacecraft Telemetry Anomaly Detection System**

**Introduction**  
Modern spacecraft systems generate a massive volume of telemetry data from various onboard sensors, such as temperature, pressure, voltage, and acceleration. Monitoring this data in real-time is essential to ensure mission safety and success. The objective of this project is to design a real-time spacecraft telemetry anomaly detection system capable of analyzing sensor data, identifying anomalies, and presenting results through an intuitive graphical interface.

**Objective**  
- Detect anomalies in spacecraft sensor data based on configurable thresholds.  
- Support both offline (CSV upload) and real-time (telemetry stream) data sources.  
- Provide user-friendly visualizations and downloadable reports for further analysis.

**Methodology**  
- Use Python and Streamlit for building an interactive GUI for anomaly detection.  
- Simulate real-time telemetry data using a socket-based server-client model.  
- Log telemetry data and visualize it using live plots.  
- Detect anomalies using predefined thresholds for each sensor metric.  
- Display the anomalies in both GUI and downloadable formats.  
- Use Tkinter for a desktop-based anomaly viewer.

**Code logic**

# Default thresholds

default\_thresholds = {

    "temperature": 100.0,

    "pressure": 1.5,

    "voltage": 32.0,

    "current": 15.0,

    "radiation": 0.005,

    "acceleration": 3.0,

    "humidity": 70.0,

    "fuel\_level": 10.0,

    "coolant\_level": 25.0,

    "gyroscope\_drift": 0.05,

    "solar\_panel\_output": 100.0

}

for i, row in df.iterrows():

            for sensor in selected\_sensors:

                if sensor not in df.columns:

                    continue

                val = row[sensor]

                if val > custom\_thresholds[sensor]:

                    anomalies.append((i, sensor, val))

# Flowchart Start └──> Choose data source (CSV or Live) ├──> If CSV

│ ├── Upload file

│ ├── Select sensors

│ ├── Configure thresholds

│ ├── Detect anomalies

│ ├── Visualize & Export

└──> If Live:

├── Start Server

├── Start Client (detect & log anomalies)

├── Live GUI view

End

# Implementation Details Technologies Used

- Python: Core programming language

- Streamlit: Web-based dashboard and interface

- Tkinter: Desktop-based anomaly viewer

- Socket: Network communication for real-time data simulation

- Matplotlib/Plotly: Visualization tools

- Pandas: Data handling and processing

# Core Modules

1. main\_gui.py: Streamlit GUI for CSV/Live input anomaly detection

2. server.py: Simulates live sensor data

3. user.py: Client that detects anomalies and logs them

4. gui.py: Tkinter-based anomaly viewer

5. run\_telemetry.py: Launches all components

**Code Structure**  
```  
project/  
│  
├── main\_gui.py # Streamlit dashboard  
├── telemetry\_server.py # Live telemetry server  
├── telemetry\_user.py # Client + anomaly detection  
├── gui.py # Tkinter viewer  
├── run\_telemetry.py # Launcher script  
├── live\_anomalies\_log.txt # Generated log file  
```

# How to Run the Code For CSV Data

1. Run `main\_gui.py`

2. Upload CSV and select sensors

3. Configure thresholds

4. View anomalies and visualizations

# For Live Data

1. Run `run\_telemetry.py`

2. This launches:

- Telemetry server

- Client (detects anomalies)

- GUI viewer (Tkinter)

1. Optionally view in Streamlit via `main\_gui.py`

**Dataset Handling**- CSV files must have sensor columns like temperature, pressure, voltage, etc.  
- Live data generated randomly with bounds  
- `live\_anomalies\_log.txt` is used to record real-time anomaly logs

**Parsing Algorithm**  
- Read CSV or telemetry stream  
- For each row, check if any sensor value exceeds its threshold  
- If so, log as an anomaly

# Output Format - \*\*CSV Upload Mode\*\*

- Output table of anomalies

- Downloadable CSV

- \*\*Live Mode\*\*:

- Logged anomalies in `live\_anomalies\_log.txt`

- GUI list display in Tkinter

**Challenges and Simulation**  
- Handling data consistency in real-time stream  
- GUI refresh sync with live telemetry  
- Robust parsing and error handling

**Test Case Result**  
- Uploaded CSV with out-of-bound values triggered correct anomaly detection  
- Real-time simulated values properly logged and displayed  
- Refresh intervals balanced between UI responsiveness and data loss

**Future Implementation**  
- Integrate ML models to predict and detect anomalies  
- Historical data trend analysis  
- Alert system via email/SMS  
- Cloud-based deployment

**Visualization Enhancement**  
- Animated plots for time-series data  
- Comparative graphs (e.g., normal vs anomalous readings)  
- Export as images or interactive dashboards

**User Interface Improvement**  
- Dark/light mode toggle  
- Drag-and-drop file support  
- More filters and sorting options

# Reference - Python documentation (socket, tkinter, pandas, etc.) - Streamlit docs

https://docs.streamlit.io/

- Plotly: https://plotly.com/python/

- NASA’s open telemetry formats

# ## Dataset Resource - Custom-simulated via `telemetry\_server.py` - Alternatively, real data from

- [NASA Open Data Portal](https://data.nasa.gov/)