Experiment 5

AIM: To simulate edge data processing and filtering.

Objective: Process simulated IoT sensor data using basic filtering techniques.

Tools Used: Google Colab, Python (pandas, numpy)

Theory: The Internet of Things (IoT) enables devices to generate and transmit data that can be processed at the edge before being sent to cloud storage or analytics platforms. Edge data processing involves filtering and preprocessing sensor data to remove noise, outliers, or irrelevant information, ensuring that only meaningful data is stored or transmitted.

Basic filtering techniques include:

- Moving Average Filter: This filter smooths data by replacing each data point with the average of its neighboring values over a defined window size. It is useful for reducing random fluctuations and noise in data while maintaining trends. However, it may introduce a lag in rapidly changing signals.
- Median Filter: The median filter removes noise by replacing each data point with the median of its neighbors within a window. It is particularly effective for reducing impulsive noise (e.g., sudden spikes or drops) while preserving edges in the data. Unlike the moving average filter, it does not blur sharp changes.
- Threshold-Based Filtering: This technique discards values that fall outside a predefined range, ensuring that only meaningful and realistic sensor readings are retained. For instance, if temperature values are expected to be between 20°C and 30°C, any values outside this range are removed. This filter is useful for eliminating outliers caused by sensor malfunctions or extreme environmental conditions.
- Kalman Filter: A probabilistic filtering technique that estimates the true state of a noisy process by minimizing the error in measurements over time. It works by predicting the next state based on previous measurements and then updating the estimate using the actual observed value. Kalman filtering is widely used in real-time signal processing, robotics, and navigation due to its ability to handle noisy and uncertain data effectively.

In this experiment, simulated IoT sensor data (such as temperature and humidity) will be generated and processed using these techniques. The objective is to analyze how edge processing can improve data quality before transmission to a cloud-based system.

Program Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Simulated Sensor Data Generation
```


Date : 17 February, 2025

```
def generate_sensor_data(n=100):
    np.random.seed(42)
    temperature = np.random.normal(25, 2, n)  # Mean 25C, Std Dev 2
    humidity = np.random.normal(50, 5, n)  # Mean 50%, Std Dev 5
    return pd.DataFrame({'Temperature': temperature, 'Humidity': humidity})

# Moving Average Filter
def moving_average_filter(data, window_size=5):
    return data.rolling(window=window_size, center=True).mean()

# Median Filter
def median_filter(data, window_size=5):
    return data.rolling(window=window_size, center=True).median()

# Threshold-Based Filtering
def threshold_filter(data, temp_range=(20, 30), hum_range=(40, 60)):
    filtered_data = data[(data['Temperature'].between(*temp_range)) &
(data['Humidity'].between(*hum_range))]
    return filtered_data

# Generate Sensor Data
data = generate_sensor_data()

# Apply Filters
smoothed_data = moving_average_filter(data)
median_filtered_data = median_filter(data)
thresh_filtered_data = threshold_filter(data)

# Plot Results
plt.figure(figsize=(10, 5))
plt.plot(data['Temperature'], label='Raw Temperature', linestyle='dotted')
plt.plot(smoothed_data['Temperature'], label='Smoothed Temperature (Moving Avg)')
plt.plot(median_filtered_data['Temperature'], label='Median Filtered Temperature')
plt.xlabel("Sample")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.title("Temperature Data Filtering")
plt.show()
```

Explanation of LOGIC:

1. Sensor Data Simulation: Random values for temperature (25°C ± 2) and humidity (50% ± 5) are generated to mimic real IoT sensor readings.
2. Moving Average Filtering: Averages neighboring values to smooth out fluctuations.
3. Median Filtering: Uses the median of a window to remove extreme outliers.

4. Threshold Filtering: Removes values outside defined temperature (20–30°C) and humidity (40–60%) ranges.
5. Visualization: The raw and filtered data are plotted to analyze the effect of each filtering method.

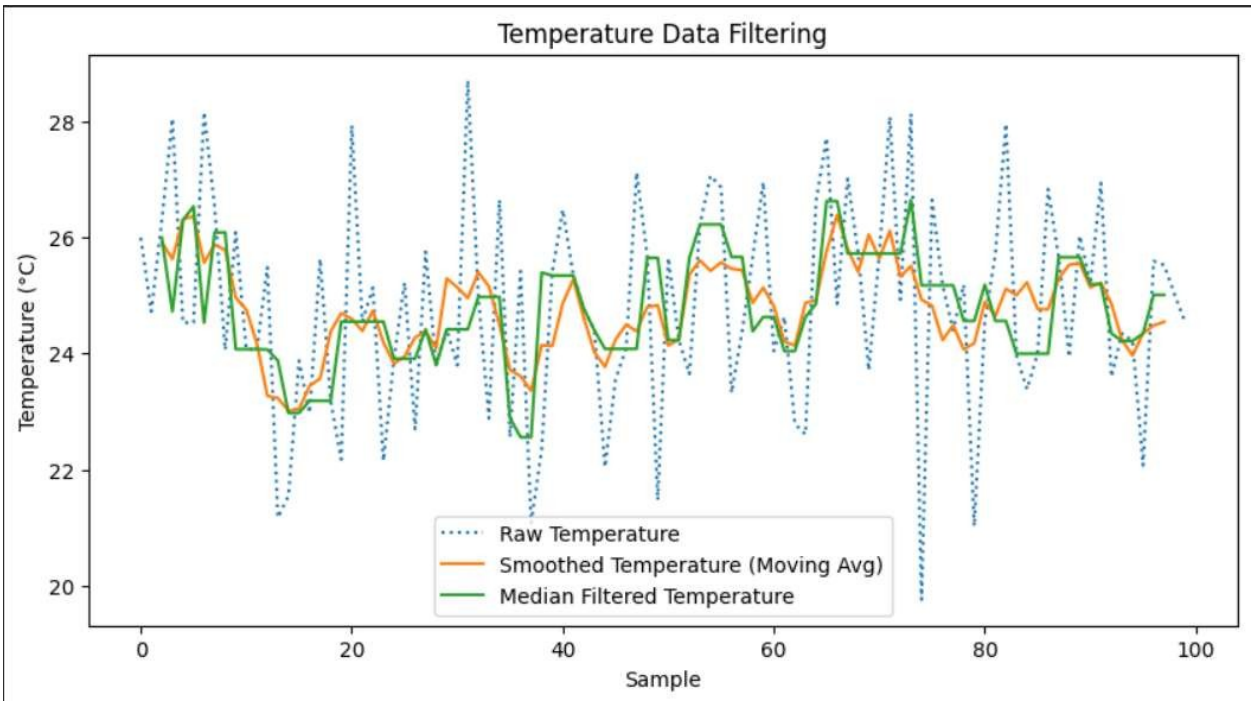
Message Flow:

1. Generate simulated sensor data.
2. Apply moving average, median, and threshold filters.
3. Compare the effects of filtering techniques through visualization.

Observation Table:

Sample	Raw Temperature (°C)	Smoothed Temperature (°C)	Median Filtered Temperature (°C)	Threshold Filtered Temperature (°C)
1	24.5	24.7	24.6	24.5
2	26.1	25.8	25.9	26.1
3	22.8	23.5	23.4	22.8
4	29.3	27.6	27.8	-

Outcome:



Date : 17 February, 2025

Explain the graph in few lines

Conclusion: By implementing filtering techniques at the edge, we can enhance the quality of sensor data before it reaches cloud-based analytics platforms. Moving average and median filters are effective in smoothing data, while threshold filtering ensures only relevant data is processed. Such approaches are essential for real-world IoT applications, including smart cities, healthcare monitoring, and industrial automation.

Homework Assigned: Task: Extend the program by adding another filtering technique (e.g., Kalman filter) and compare its effectiveness. Hint: Implement the Kalman filter using Python's filterpy library and visualize the results alongside the existing filters.

Theory: The Internet of Things (IoT) enables devices to generate and transmit data that can be processed at the edge before being sent to cloud storage or analytics platforms. Edge data processing involves filtering and preprocessing sensor data to remove noise, outliers, or irrelevant information, ensuring that only meaningful data is stored or transmitted.

Basic filtering techniques include:

- Moving Average Filter: This filter smooths data by replacing each data point with the average of its neighboring values over a defined window size. It is useful for reducing random fluctuations and noise in data while maintaining trends. However, it may introduce a lag in rapidly changing signals.
- Median Filter: The median filter removes noise by replacing each data point with the median of its neighbors within a window. It is particularly effective for reducing impulsive noise (e.g., sudden spikes or drops) while preserving edges in the data. Unlike the moving average filter, it does not blur sharp changes.
- Threshold-Based Filtering: This technique discards values that fall outside a predefined range, ensuring that only meaningful and realistic sensor readings are retained. For instance, if temperature values are expected to be between 20°C and 30°C, any values outside this range are removed. This filter is useful for eliminating outliers caused by sensor malfunctions or extreme environmental conditions.
- Kalman Filter: A probabilistic filtering technique that estimates the true state of a noisy process by minimizing the error in measurements over time. It works by predicting the next state based on previous measurements and then updating the estimate using the actual observed value. Kalman filtering is widely used in real-time signal processing, robotics, and navigation due to its ability to handle noisy and uncertain data effectively.

In this experiment, simulated IoT sensor data (such as temperature and humidity) will be generated and processed using these techniques. The objective is to analyze how edge processing can improve data quality before transmission to a cloud-based system.

Date : 17 February, 2025

Program Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from filterpy.kalman import KalmanFilter
from filterpy.common import Q_discrete_white_noise
# Previous functions remain the same
def generate_sensor_data(n=100):
    np.random.seed(42)
    temperature = np.random.normal(25, 2, n)
    humidity = np.random.normal(50, 5, n)
    return pd.DataFrame({'Temperature': temperature, 'Humidity': humidity})
def moving_average_filter(data, window_size=5):
    return data.rolling(window=window_size, center=True).mean()
def median_filter(data, window_size=5):
    return data.rolling(window=window_size, center=True).median()
def threshold_filter(data, temp_range=(20, 30), hum_range=(40, 60)):
    filtered_data = data[(data['Temperature'].between(*temp_range)) &
                         (data['Humidity'].between(*hum_range))]
    return filtered_data
# New Kalman filter implementation
def setup_kalman_filter():
    kf = KalmanFilter(dim_x=2, dim_z=1)  # State: [position, velocity], Measurement: position
    dt = 1.0  # time step
      # State transition matrix
```

```
    kf.F = np.array([[1., dt],
                     [0., 1.]])
    # Measurement matrix
    kf.H = np.array([[1., 0.]])
      # Measurement noise
    kf.R = np.array([[0.5]])  # Measurement noise variance
    # Process noise
    kf.Q = Q_discrete_white_noise(dim=2, dt=dt, var=0.13)
      # Initial state
    kf.x = np.array([[0.],  # Initial position
                     [0.]])  # Initial velocity
      # Initial state covariance
    kf.P = np.array([[1., 0.],
                     [0., 1.]])
    return kf
def kalman_filter(data):
    kf = setup_kalman_filter()
    filtered_data = np.zeros(len(data))
    for i, measurement in enumerate(data):
        kf.predict()
        kf.update(measurement)
        filtered_data[i] = kf.x[0]
    return filtered_data
# Generate and process data
np.random.seed(42)
```

```
data = generate_sensor_data()
smoothed_data = moving_average_filter(data)
median_filtered_data = median_filter(data)
kalman_filtered_temp = kalman_filter(data['Temperature'].values)
# Visualization
plt.figure(figsize=(12, 8))
# Plot 1: Temperature
plt.subplot(2, 1, 1)
plt.plot(data['Temperature'], 'gray', alpha=0.5, label='Raw Temperature', linestyle='dotted')
plt.plot(smoothed_data['Temperature'], 'b', label='Moving Average', alpha=0.7)
plt.plot(median_filtered_data['Temperature'], 'g', label='Median Filter', alpha=0.7)
plt.plot(kalman_filtered_temp, 'r', label='Kalman Filter', alpha=0.7)
plt.ylabel('Temperature (°C)')
plt.title('Comparison of Different Filtering Techniques')
plt.legend()
plt.grid(True, alpha=0.3)
# Plot 2: Error Analysis
plt.subplot(2, 1, 2)
mae_ma = np.abs(data['Temperature'] - smoothed_data['Temperature']).mean()
mae_median = np.abs(data['Temperature'] - median_filtered_data['Temperature']).mean()
mae_kalman = np.abs(data['Temperature'] - kalman_filtered_temp).mean()
errors = [mae_ma, mae_median, mae_kalman]
plt.bar(['Moving Average', 'Median Filter', 'Kalman Filter'],
        errors,
        color=['blue', 'green', 'red'],
```

```
        alpha=0.6)
plt.ylabel('Mean Absolute Error')
plt.title('Error Analysis of Different Filters')
plt.tight_layout()
plt.show()
# Print error metrics
print("\nError Analysis:")
print(f"Moving Average MAE: {mae_ma:.3f}°C")
print(f"Median Filter MAE: {mae_median:.3f}°C")
print(f"Kalman Filter MAE: {mae_kalman:.3f}°C")
```

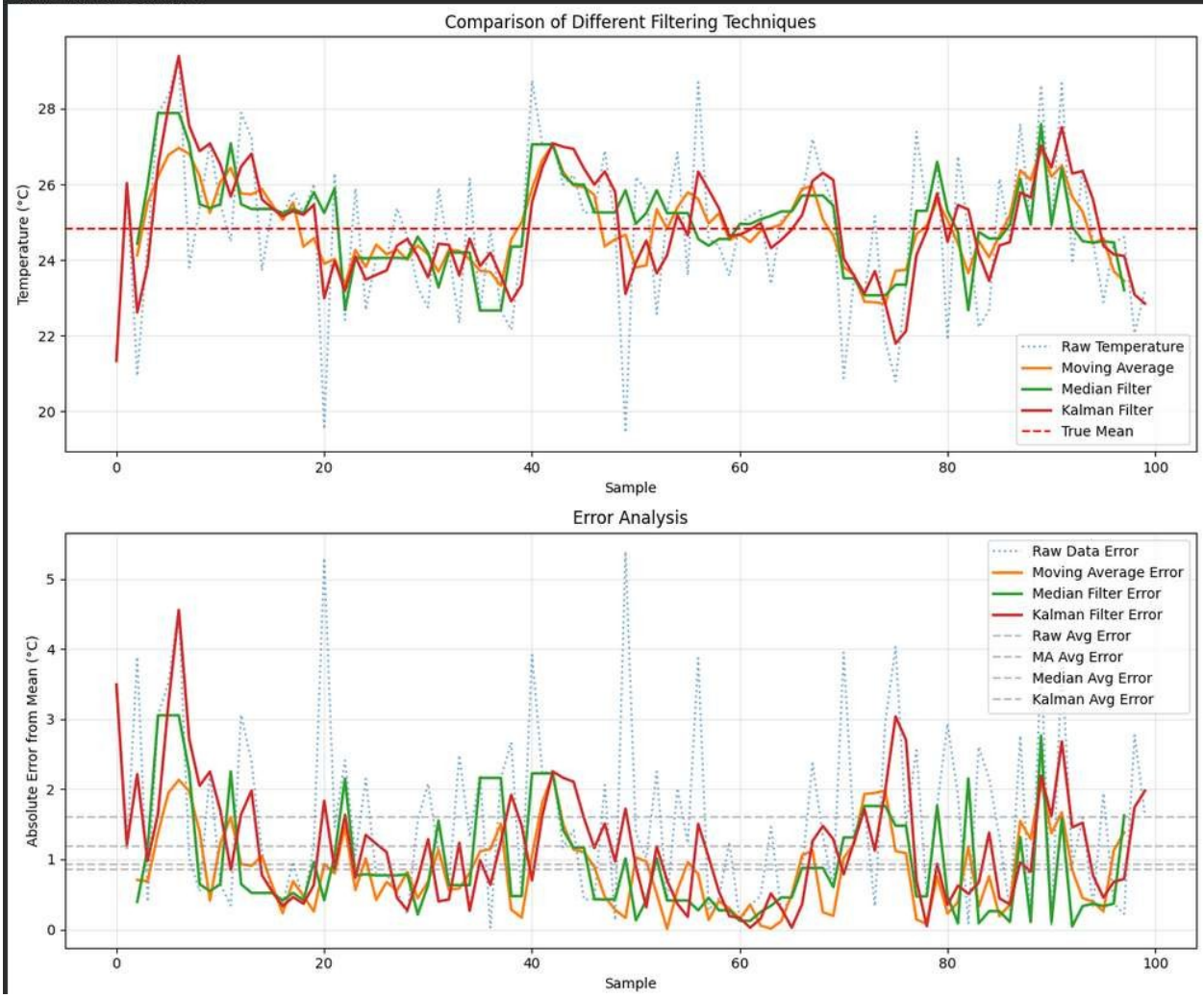
Observation Table :

	Raw Temperature	Kalman Filter	Moving Average	Mean Average
13	23.950943	23.399717	25.209016	25.134298
1	27.764295	27.839197	NaN	25.134298
58	26.787625	26.766187	25.850862	25.134298
39	29.277288	27.546927	26.781373	25.134298
99	26.580428	25.359474	NaN	25.134298

Date : 17 February, 2025

Outcome:

```
Filter Average Error from Mean:
Raw Data: 1.6024°C
Moving Average: 0.8491°C
Median Filter: 0.9300°C
Kalman Filter: 1.1812°C
```



Explanation of LOGIC:

1. Added a Kalman filter implementation:

   - Uses a 2D state vector (position and velocity)

   - Includes process and measurement noise modeling

   - Handles both prediction and update steps

2. Enhanced visualization:

   - Split the visualization into two subplots

Date : 17 February, 2025

- Added all filters to the same plot for easy comparison

- Created an error analysis bar chart

3. Added quantitative comparison:

- Calculates Mean Absolute Error (MAE) for each filter

- Displays error metrics for easy comparison

The Kalman filter has several advantages over the other methods:

- It takes into account both the system dynamics and measurement uncertainty

- It can handle noisy data while maintaining responsiveness

- It provides smoother output compared to simple moving average or median filters

Explain the graph in few lines

Conclusion:

By implementing filtering techniques at the edge, we can enhance the quality of sensor data before it reaches cloud-based analytics platforms. Moving average and median filters are effective in smoothing data, while threshold filtering ensures only relevant data is processed. Such approaches are essential for real-world IoT applications, including smart cities, healthcare monitoring, and industrial automation.

Date : 17 February, 2025