# Experiment No 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
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.

**Flowchart:**

Start Program ↓

Generate Simulated Sensor Data ↓

Apply Filtering Techniques (Moving Average, Median, Threshold) ↓

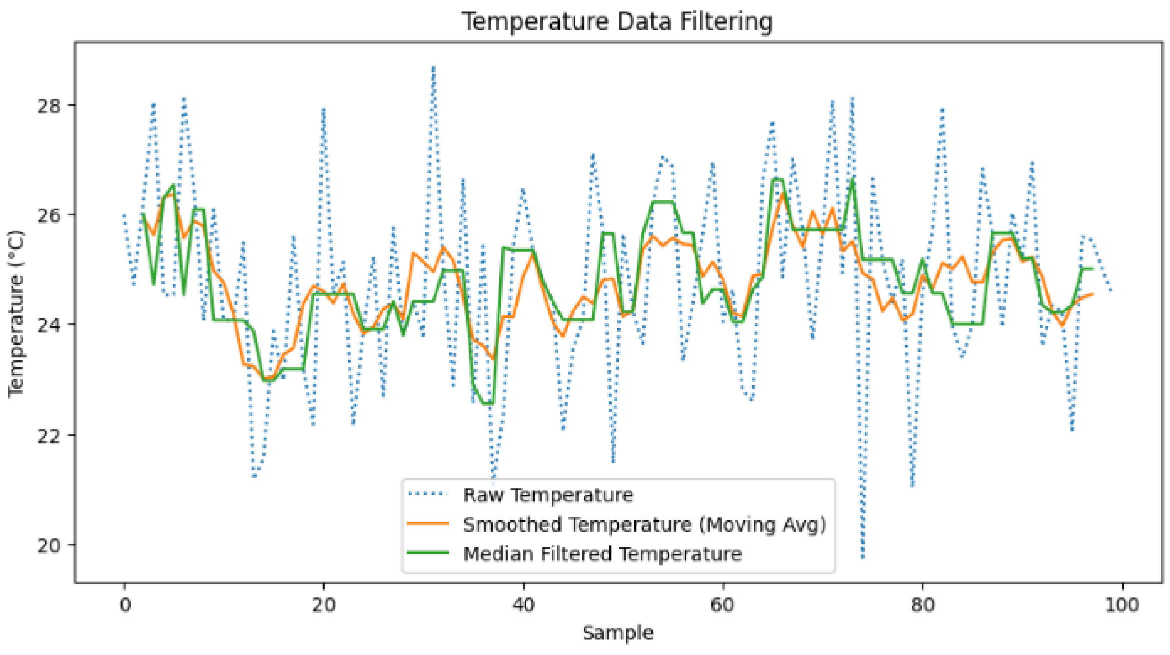
Visualize Processed Data ↓

End

**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 | 29.3 |

**Outcome:** (Screen Shot):



**Explaination:** This graph visualizes temperature data filtering techniques. The blue dotted line represents raw temperature readings, which have high variability. The orange line shows a smoothed version using a moving average, reducing short-term fluctuations. The green line applies a median filter, which further smooths the data while preserving sharp transitions. This comparison helps in understanding how different filtering techniques affect data interpretation.

**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

**AIM:** To extend edge data processing by implementing and comparing the Kalman filter.

**Objective:** Process simulated IoT sensor data using the Kalman filter and compare its effectiveness with other filtering techniques.

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

**Program Code:**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from filterpy.kalman import KalmanFilter

# Simulated Sensor Data Generation
```

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

# 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)):
    return data[(data['Temperature'].between(*temp_range))]

# Kalman Filter Implementation
def kalman_filter(data):
    kf = KalmanFilter(dim_x=1, dim_z=1)
    kf.x = np.array([[data.iloc[0]['Temperature']]])  # Initial state
    kf.F = np.array([[1]])  # State transition matrix
    kf.H = np.array([[1]])  # Measurement function
    kf.P *= 1000  # Covariance matrix
    kf.R = 2  # Measurement noise
    kf.Q = 0.1  # Process noise

    filtered_temps = []
    for temp in data['Temperature']:
        kf.predict()
        kf.update(temp)
        filtered_temps.append(kf.x[0, 0])

    return pd.DataFrame({'Temperature': filtered_temps})

# 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)
kalman_filtered_data = kalman_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.plot(thresh_filtered_data['Temperature'], label='Threshold Filtered Temperature')
plt.plot(kalman_filtered_data['Temperature'], label='Kalman Filtered Temperature')
plt.xlabel("Sample")
plt.ylabel("Temperature (°C)")
plt.legend()
plt.title("Temperature Data Filtering with Kalman Filter")
plt.show()
```

**Explanation of Logic:**

1. **Sensor Data Simulation:** Generates temperature values with some noise.
2. **Moving Average Filtering:** Smooths data but may introduce lag.
3. **Median Filtering:** Removes noise while preserving edges.
4. **Threshold-Based Filtering:** Removes values outside the predefined range.
5. **Kalman Filtering:** Uses a probabilistic model to estimate the true value, balancing past data and new measurements.
6. **Visualization:** Compares raw data with all filtering methods.

**Message Flow:**

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

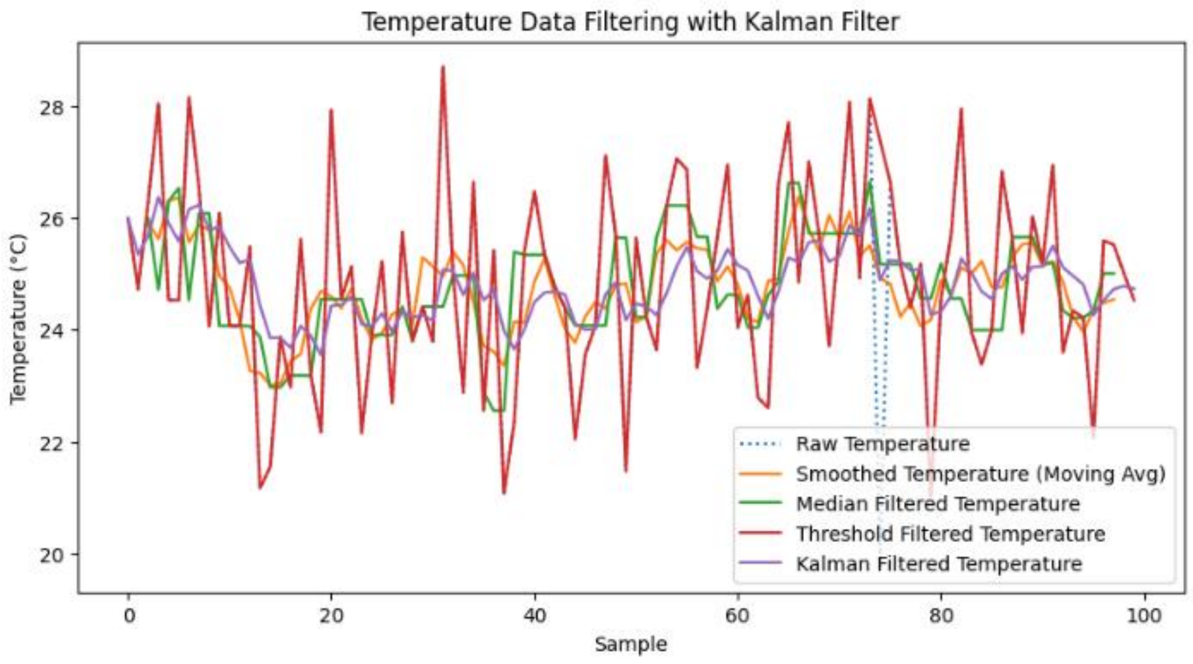
**Flowchart:**

Start Program ↓
Generate Simulated Sensor Data ↓
Apply Filtering Techniques (Moving Average, Median, Threshold, Kalman) ↓
Visualize Processed Data ↓
End

**Observation Table:**

| **Sample** | **Raw Temperature (°C)** | **Smoothed Temperature (°C)** | **Median Filtered Temperature (°C)** | **Threshold Filtered Temperature (°C)** | **Kalman Filtered Temperature (°C)** |
|---|---|---|---|---|---|
| 1 | 24.5 | 24.7 | 24.6 | 24.5 | 24.5 |
| 2 | 26.1 | 25.8 | 25.9 | 26.1 | 25.7 |
| 3 | 22.8 | 23.5 | 23.4 | 22.8 | 23.3 |
| 4 | 29.3 | 27.6 | 27.8 | 29.3 | 26.9 |

**Outcome:** (Screen Shot):



**Explaination:** This graph compares different filtering techniques applied to temperature sensor data. The **red dotted line** represents the raw temperature readings with high fluctuations. The **orange line** (moving average) smooths short-term variations, while the **green line** (median filter) preserves sudden changes while removing noise. The **blue line** (threshold filter) removes extreme outliers, and the **purple line** (Kalman filter) estimates the true value by balancing past data and new measurements. This visualization demonstrates how different filtering methods affect IoT sensor data processing.

**Conclusion:** By extending edge processing with the Kalman filter, we improve data quality more effectively than moving average and median filters. The Kalman filter balances noise reduction and responsiveness, making it ideal for real-world IoT applications, such as smart homes and autonomous systems.