

Experiment No 3

AIM: To visualize sensor data in real-time using Dash or Plotly.

Objective:

To build an interactive dashboard for monitoring IoT data, enabling users to visualize real-time trends and analyze sensor readings effectively.

Tools Used: Google Colab, Dash/Plotly libraries

Theory:

The Internet of Things (IoT) connects devices to the internet, allowing them to communicate and share data. Visualizing this data in real-time is critical for effective monitoring and decision-making in applications such as smart homes, industrial automation, and healthcare. Dashboards serve as an interface to represent data dynamically and interactively.

Dash is a Python-based framework for building web applications with analytical capabilities. It integrates seamlessly with Plotly, a library for creating interactive graphs, to provide a powerful tool for IoT data visualization. By using Dash and Plotly, sensor data can be streamed and visualized in real time, enhancing the ability to monitor systems and detect anomalies quickly.

In this experiment, we demonstrate how to build a dashboard that updates dynamically, reflecting sensor data trends. This approach bridges the gap between raw data and actionable insights, emphasizing the importance of real-time monitoring in IoT systems.

Program Code:

```
!pip install dash
from dash import Dash, dcc, html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import random
import time
import threading

# Simulated Sensor Data Generator
def generate_sensor_data():
    while True:
        data['temperature'].append(round(random.uniform(20.0, 30.0), 2))
        data['humidity'].append(round(random.uniform(40.0, 60.0), 2))
        data['time'].append(time.strftime("%H:%M:%S"))
```

```
# Keep only the last 100 data points for efficiency
if len(data['time']) > 100:
    data['temperature'] = data['temperature'][-100:]
    data['humidity'] = data['humidity'][-100:]
    data['time'] = data['time'][-100:]
    time.sleep(1) # Simulate real-time data every second

# Initialize sensor data
data = {'time': [], 'temperature': [], 'humidity': []}
threading.Thread(target=generate_sensor_data, daemon=True).start()

# Dash Application Setup
app = Dash(__name__)
app.layout = html.Div([
    html.H1("Real-Time IoT Sensor Dashboard", style={
        'textAlign': 'center',
        'color': '#4CAF50',
        'fontFamily': 'Arial, sans-serif',
        'marginBottom': '30px',
        'textShadow': '2px 2px 4px #888'
    }),
    # Digital Displays
    html.Div([
        html.Div([
            html.H3("Current Temperature", style={
                'color': '#FF5722',
                'fontFamily': 'Arial, sans-serif',
                'marginBottom': '10px',
                'textShadow': '1px 1px 2px #555'
            }),
            html.Div(id='temperature-digital', style={
                'fontSize': '40px',
                'fontWeight': 'bold',
                'color': '#FF5722',
                'textAlign': 'center',
                'background': 'linear-gradient(to right, #FFE0B2, #FFCCBC)',
                'padding': '10px',
            })
        ])
    ])
])
```

```
'borderRadius': '15px',
'boxShadow': '0px 4px 6px rgba(0, 0, 0, 0.1)'
})
], style={'flex': 1, 'margin': '10px'}),
html.Div([
    html.H3("Current Humidity", style={
        'color': '#2196F3',
        'fontFamily': 'Arial, sans-serif',
        'marginBottom': '10px',
        'textShadow': '1px 1px 2px #555'
    }),
    html.Div(id='humidity-digital', style={
        'fontSize': '40px',
        'fontWeight': 'bold',
        'color': '#2196F3',
        'textAlign': 'center',
        'background': 'linear-gradient(to right, #BBDEFB, #90CAF9)',
        'padding': '10px',
        'borderRadius': '15px',
        'boxShadow': '0px 4px 6px rgba(0, 0, 0, 0.1)'
    })
], style={'flex': 1, 'margin': '10px'})
], style={

    'display': 'flex',
    'justifyContent': 'space-around',
    'alignItems': 'center',
    'marginBottom': '30px',
    'padding': '20px',
    'background': 'linear-gradient(to bottom, #F5F5F5, #E0E0E0)',
    'borderRadius': '20px',
    'boxShadow': '0px 4px 8px rgba(0, 0, 0, 0.2)'
}),
),
# Graphs
html.Div([
    dcc.Graph(id='temperature-graph', style={'flex': 1, 'margin': '10px'})
])
```

```

dcc.Graph(id='humidity-graph', style={'flex': 1, 'margin': '10px'})

], style={

    'display': 'flex',
    'justifyContent': 'space-around',
    'alignItems': 'center',
    'marginBottom': '30px',
    'padding': '20px',
    'background': 'linear-gradient(to bottom, #FAFAFA, #EEEEEE)',
    'borderRadius': '20px',
    'boxShadow': '0px 4px 8px rgba(0, 0, 0, 0.2)'

}),
# Interval component for live updates
dcc.Interval(
    id='interval-component',
    interval=1000, # 1-second interval
    n_intervals=0
)
])

@app.callback(
    [Output('temperature-graph', 'figure'),
     Output('humidity-graph', 'figure'),
     Output('temperature-digital', 'children'),
     Output('humidity-digital', 'children')],
    [Input('interval-component', 'n_intervals')]
)
def update_dashboard(n):
    # Create Temperature Graph
    temp_fig = go.Figure()
    temp_fig.add_trace(go.Scatter(x=data['time'], y=data['temperature'], mode='lines+markers',
name='Temperature'))
    temp_fig.update_layout(
        title={
            'text': "Temperature over Time",
            'y': 0.9,
            'x': 0.5,
        }
)

```

```

        'xanchor': 'center',
        'yanchor': 'top'
    },
    xaxis_title="Time",
    yaxis_title="Temperature (°C)",
    font=dict(family="Arial, sans-serif", size=14),
    plot_bgcolor="#F9F9F9",
    paper_bgcolor="#F9F9F9",
    margin=dict(l=40, r=40, t=40, b=40)
)

# Create Humidity Graph
hum_fig = go.Figure()
hum_fig.add_trace(go.Scatter(x=data['time'], y=data['humidity'], mode='lines+markers',
name='Humidity'))
hum_fig.update_layout(
    title={
        'text': "Humidity over Time",
        'y': 0.9,
        'x': 0.5,
        'xanchor': 'center',
        'yanchor': 'top'
    },
    xaxis_title="Time",
    yaxis_title="Humidity (%)",
    font=dict(family="Arial, sans-serif", size=14),
    plot_bgcolor="#F9F9F9",
    paper_bgcolor="#F9F9F9",
    margin=dict(l=40, r=40, t=40, b=40)
)

# Get the latest temperature and humidity values
current_temperature = data['temperature'][-1] if data['temperature'] else "N/A"
current_humidity = data['humidity'][-1] if data['humidity'] else "N/A"

# Return the figures and the digital displays
return temp_fig, hum_fig, f"{current_temperature} °C", f"{current_humidity} %"

if __name__ == '__main__':

```

```
app.run_server(debug=True)
```

Explanation and Logic of Code:**Sensor Data Simulation:**

The generate_sensor_data function generates random values for temperature (20.0–30.0 °C) and humidity (40.0–60.0%) to simulate sensor readings.

Each data point is associated with a timestamp and stored in the data dictionary.

Real-Time Dashboard:

The Dash framework is used to create an interactive web-based dashboard.

The layout includes two line graphs to visualize temperature and humidity trends over time.

Dynamic Updates:

The dcc.Interval component triggers updates to the graphs every second.

The update_graphs function fetches the latest data and updates the figures dynamically.

Threading:

A separate thread runs the data generator, ensuring continuous data updates without blocking the dashboard's operation.

Message Flow:

Sensor Data Simulation: Generates data for temperature and humidity.

Data Storage: Stores generated data in memory.

Graph Updates: Fetches and visualizes the latest data on the dashboard.

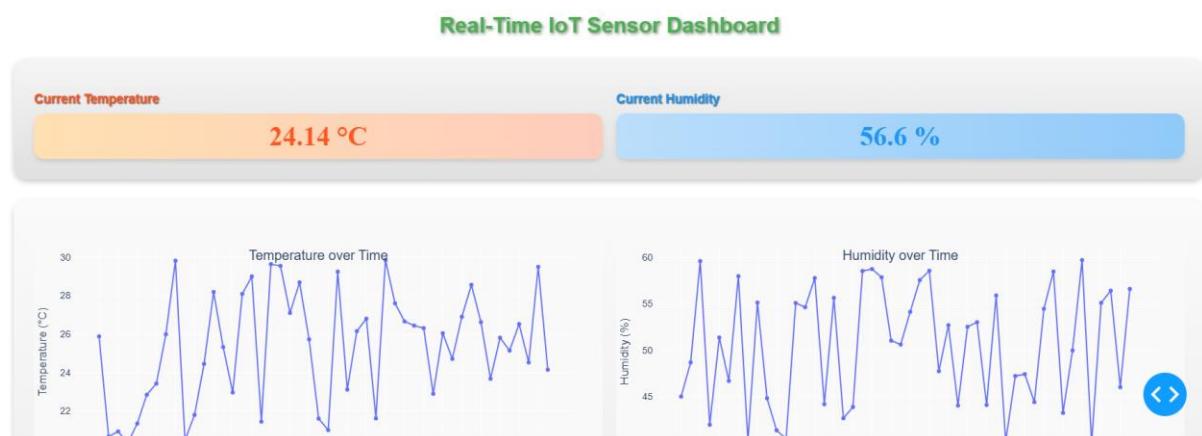
Flowchart:

Start Program ↓ Initialize Sensor Data and Dashboard ↓ Generate Real-Time Sensor Data ↓ Update Dashboard with Latest Data ↓ End

Observation Table:

Timestamp	Temperature (°C)	Humidity (%)	Remarks
2025-02-3 01:13:51	24.14	56.6	Drop in Temp.
2025-02-3 01:13:53	28.73	43.9	Normal readings.
2025-02-3 01:13:56	26.0	41.72	Humidity decreased.
2025-02-3 01:13:59	24.98	59.1	Stable conditions.

Outcome: (screenshot)



Conclusion:

Building a real-time IoT sensor dashboard bridges the gap between raw data and actionable insights. By simulating sensor data and visualizing it with Dash/Plotly, this experiment provides a practical understanding of real-time monitoring systems. The techniques demonstrated are foundational for developing advanced IoT applications, enhancing both learning and system prototyping.

Homework Assigned

AIM: To visualize sensor data in real-time using Dash or Plotly with an additional sensor.

Objective:

To build an interactive dashboard for monitoring IoT data, enabling users to visualize real-time trends and analyze sensor readings effectively. A new sensor for light intensity is added to enhance the experiment.

Tools Used: Google Colab, Dash/Plotly libraries

Program Code:

```

!pip install dash
from dash import Dash, dcc, html
from dash.dependencies import Input, Output
import plotly.graph_objs as go
import random
import time
import threading

# Simulated Sensor Data Generator
def generate_sensor_data():
    while True:
        data['temperature'].append(round(random.uniform(20.0, 30.0), 2))
        data['humidity'].append(round(random.uniform(40.0, 60.0), 2))
        data['light_intensity'].append(round(random.uniform(100, 1000), 2))
        data['time'].append(time.strftime("%H:%M:%S"))
        if len(data['time']) > 100:
            data['temperature'] = data['temperature'][-100:]
            data['humidity'] = data['humidity'][-100:]
            data['light_intensity'] = data['light_intensity'][-100:]
            data['time'] = data['time'][-100:]
        time.sleep(1)

# Initialize sensor data
data = {'time': [], 'temperature': [], 'humidity': [], 'light_intensity': []}
threading.Thread(target=generate_sensor_data, daemon=True).start()

# Dash Application Setup
app = Dash(__name__)

```

```

app.layout = html.Div([
    html.H1("Real-Time IoT Sensor Dashboard"),
    dcc.Graph(id='temperature-graph'),
    dcc.Graph(id='humidity-graph'),
    dcc.Graph(id='light-graph'),
    dcc.Interval(id='interval-component', interval=1000, n_intervals=0)
])

@app.callback(
    [Output('temperature-graph', 'figure'),
     Output('humidity-graph', 'figure'),
     Output('light-graph', 'figure')],
    [Input('interval-component', 'n_intervals')]
)

def update_dashboard(n):
    temp_fig = go.Figure()
    temp_fig.add_trace(go.Scatter(x=data['time'], y=data['temperature'], mode='lines+markers',
name='Temperature'))
    hum_fig = go.Figure()
    hum_fig.add_trace(go.Scatter(x=data['time'], y=data['humidity'], mode='lines+markers',
name='Humidity'))
    light_fig = go.Figure()
    light_fig.add_trace(go.Scatter(x=data['time'], y=data['light_intensity'], mode='lines+markers',
name='Light Intensity'))
    return temp_fig, hum_fig, light_fig

if __name__ == '__main__':
    app.run_server(debug=True)

```

Explanation and Logic of Code:

- The generate_sensor_data function generates random values for temperature, humidity, and light intensity.
- Each data point is stored in a dictionary and updated in real-time.
- The Dash framework is used to create an interactive web-based dashboard.
- Three graphs visualize temperature, humidity, and light intensity trends over time.
- The dcc.Interval component ensures continuous updates.
- The callback function updates the graphs dynamically.

Message Flow:

1. Sensor Data Simulation: Generates data for temperature, humidity, and light intensity.
2. Data Storage: Stores generated data in memory.

3. Graph Updates: Fetches and visualizes the latest data on the dashboard.

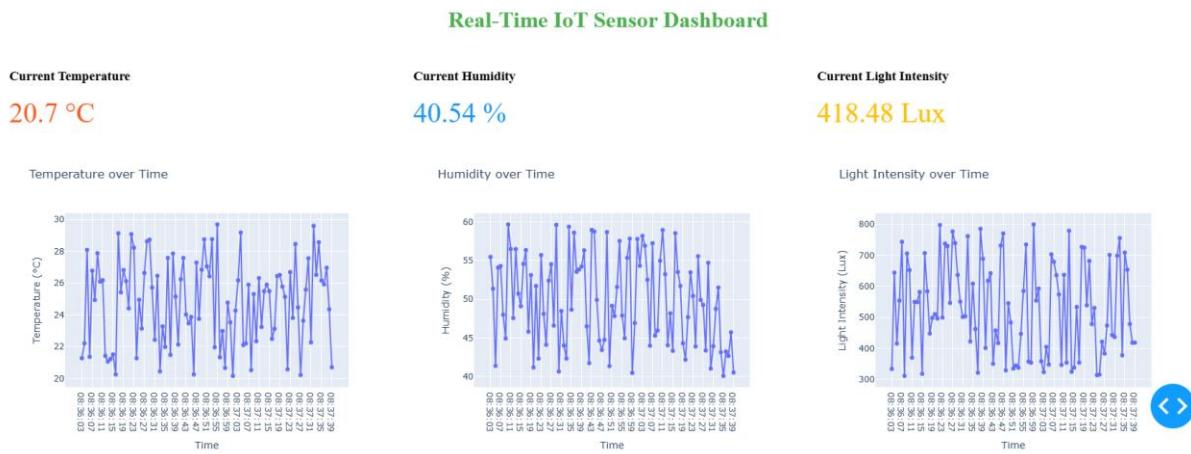
Flowchart:

Start Program ↓ Initialize Sensor Data and Dashboard ↓ Generate Real-Time Sensor Data ↓ Update Dashboard with Latest Data ↓ End

Observation Table:

Timestamp	Temperature (°C)	Humidity (%)	Light Intensity (lx)	Remarks
2025-02-3 01:18:51	24.14	56.6	452.3	Normal readings.
2025-02-3 01:18:53	28.73	43.9	876.2	Increase in light.
2025-02-3 01:18:56	26.0	41.72	623.4	Stable conditions.
2025-02-3 01:18:59	24.98	59.1	210.5	Low light detected.

Outcome: (screenshot):



Conclusion:

By adding a new sensor for light intensity, we enhanced the real-time monitoring capabilities of the IoT dashboard. This experiment demonstrates how additional sensors can be integrated seamlessly, providing valuable insights for various applications. The techniques demonstrated are foundational for developing advanced IoT applications, enhancing both learning and system prototyping.