# Fire Aware +

## Introduction

In modern industrial and power systems, ensuring the safety and reliability of electrical networks is a critical challenge. Electrical faults, such as overheating, overcurrent, and voltage fluctuations, can lead to severe consequences, including equipment damage, power outages, and fire hazards. Traditional monitoring methods often rely on manual inspection and delayed responses, which can be inefficient in preventing failures.

This project presents an **intelligent fault detection system** designed to **continuously monitor electrical parameters** such as temperature, current, and voltage using **sensor-based simulation and real-time data processing**. By implementing **MATLAB for sensor simulation** and integrating **Arduino/Raspberry Pi for real-world data acquisition**, the system efficiently detects deviations from safe operational limits.

A key innovation of this system is its ability to **automate fault response actions**. Upon detecting an anomaly, the system can **trigger an alarm, disconnect power through a relay, and send real-time alerts via Telegram**. This ensures immediate action is taken, reducing the risk of damage and enhancing system reliability.

This solution is particularly valuable for **industrial environments, smart grids, and automated monitoring applications**, where real-time fault detection can significantly improve safety and operational efficiency. Furthermore, its modular design allows for **scalability and integration with additional sensors**, making it adaptable to various applications.

---

## Technologies Used

This project integrates a combination of **hardware and software technologies** to achieve **real-time electrical fault detection and automated response mechanisms**. The key technologies used include:

### 1. Software Technologies

- **MATLAB**
  - Used for **simulating sensor data** to model real-world electrical parameters.
  - Generates **randomized sensor readings** for **temperature, current, and voltage** using statistical functions (e.g., randn).
  - Implements **data processing and fault detection algorithms** to analyze deviations from safe operational thresholds.
- **Telegram Bot API**
  - Enables **real-time notifications** to alert users of detected electrical faults.

    - Uses botToken and chatID to **send automated alerts** when a fault is detected.
    - Integrated with MATLAB's webread function for API communication.

**2. Hardware Technologies**

- **Arduino/Raspberry Pi**
    - Serves as the **control unit** to read real-time data from physical sensors.
    - Interfaces with **analog-to-digital converters (ADC)** to collect data from electrical components.
- **Sensors**
    - **Thermistor (Temperature Sensor)** – Measures the system's temperature.
    - **Current Sensor** – Monitors electrical current flow in the circuit.
    - **Voltage Sensor** – Detects voltage levels and fluctuations.
- **Actuators & Safety Mechanisms**
    - **Relay Module** – **Automatically disconnects power** when a fault is detected to prevent damage.
    - **Buzzer** – Provides **an audible alert** when an electrical abnormality is identified.

**3. Data Processing & Fault Detection**

- **Threshold-based Fault Detection Algorithm**
    - Uses predefined safety limits:
        - **Temperature Threshold**: 50°C
        - **Current Threshold**: 15A
        - **Voltage Range**: 190V - 250V
    - If sensor readings exceed safe values, the system **activates an alert, disconnects power, and sends a notification**.
- **Cloud Communication & IoT Integration**
    - Uses **Telegram API** to notify users remotely.
    - Can be expanded to include **cloud-based data storage and dashboards** for further analytics.

---

## Project Phases

The development and implementation of the **intelligent electrical fault detection system** are divided into **three key phases**, each focusing on a critical aspect of the system's functionality.

## Phase 1: Sensor Data Simulation & Initialization

**Objective:**
To generate realistic electrical sensor readings and define operational thresholds.

**Key Steps:**

- Implement **MATLAB-based simulation**to create synthetic sensor readings:
    - **Temperature**: Simulated using randn around a mean of 40°C with a ±15°C variation.
    - **Current**: Simulated with a mean of 10A and a ±5A variation.

    - **Voltage**: Simulated around 220V with a ±20V range.
- Define **safe operating thresholds**:
    - Temperature: **≤ 50°C**
    - Current: **≤ 15A**
    - Voltage: **Between 190V and 250V**
- Set up **sampling frequency (Fs = 1 Hz)** and simulation time (30 seconds).

**Implementation:**

- Uses MATLAB's randn function to introduce **random fluctuations** in sensor values.
- Displays the simulated readings for **real-time visualization**.

## Phase 2: Fault Detection & Automated Response

**Objective:**
To analyze sensor readings, detect anomalies, and trigger automated safety responses.

**Key Steps:**

- Continuously **compare real-time sensor readings** against threshold values.
- If an **anomaly is detected**(e.g., high temperature, excessive current, or voltage fluctuations):
    - **Activate the relay** (disconnect power).
    - **Trigger the buzzer** for a local audible alert.
    - **Send an alert message via Telegram** to notify the user.
- If readings return to normal:
    - **Reconnect power via relay**.
    - **Turn off the buzzer**.

**Implementation:**

- Uses an **if-else logic structure** to determine when an action is needed.
- webread function in MATLAB sends Telegram alerts when faults occur.
- Logs data in real-time to display system status.

## Phase 3: Real-World Deployment & Expansion

**Objective:**
To transition from **simulation to physical implementation** and enhance system scalability.

**Key Steps:**

- **Deploy the system on Arduino/Raspberry Pi** to collect real-world sensor readings.
- Interface with **physical sensors (temperature, current, voltage) using ADC inputs**.
- **Test fault detection accuracy** under controlled conditions.
- Expand functionality:
    - **Cloud integration** for remote monitoring and data logging.
    - **AI-based predictive maintenance** to anticipate failures before they occur.
    - **Mobile app dashboard** for a user-friendly interface.

**Implementation:**

- Hardware components (sensors, relays, and controllers) replace simulated data.
- Enhancements such as **IoT connectivity and AI-based analytics** can be integrated.

---

## The MATLAB Code

```
clc; clear; close all;

botToken = [redacted]
chatID = '1781467825';

fs = 1;
simTime = 30;

temp_threshold = 50;
current_threshold = 15;
voltage_min = 190;
voltage_max = 250;

rng shuffle;

sim_data = zeros(simTime, 6);
sim_data(:,1) = (0:simTime-1)';
sim_data(:,2) = 40 + 15 * randn(simTime, 1);
sim_data(:,3) = 10 + 5 * randn(simTime, 1);
sim_data(:,4) = 220 + 20 * randn(simTime, 1);

sim_data(isnan(sim_data)) = 0;
sim_data(isinf(sim_data)) = 0;

voltage_data = [sim_data(:,1), sim_data(:,4)];
```

```
assignin('base', 'voltage_data', voltage_data);


time_data = [sim_data(:,1), sim_data(:,1)];
assignin('base', 'time_data', time_data);


temperature_data = [sim_data(:,1), sim_data(:,2)];
current_data = [sim_data(:,1), sim_data(:,3)];


assignin('base', 'temperature_data', temperature_data);
assignin('base', 'current_data', current_data);


sim_data(:,5) = 1;
sim_data(:,6) = 0;


disp('🚀 Simulation started...');


for t = 1:simTime
    sim_data(t,2:4) = max(0, sim_data(t,2:4));


    voltage_value = sim_data(t,4);
    fault_detected = sim_data(t,2) > temp_threshold || sim_data(t,3) > current_threshold
|| voltage_value < voltage_min || voltage_value > voltage_max;
    sim_data(t,5) = ~fault_detected;
    sim_data(t,6) = fault_detected;


    if fault_detected
```

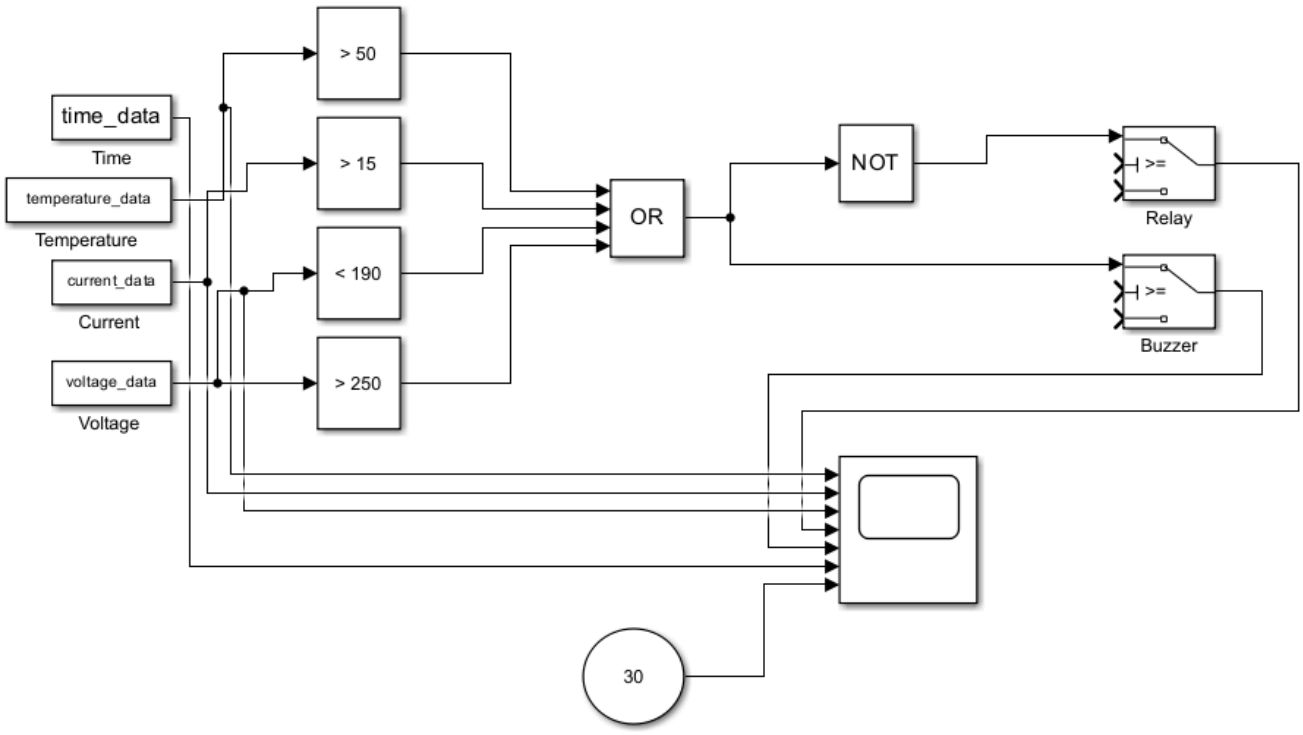
```
        message = sprintf('🚨 System fault detected!\n🌡️ Temperature: %.2f°C\n⚡ Current: %.2fA\n🔋 Voltage: %.2fV\n🔴 Power disconnected.', ...
            sim_data(t,2), sim_data(t,3), voltage_value);
        telegramURL = sprintf('https://api.telegram.org/bot%s/sendMessage?chat_id=%s&text=%s', ...
            botToken, chatID, urlencode(message));

        try
            webread(telegramURL);
            disp('📨 Notification sent to Telegram!');
        catch ME
            disp('⚠️ Failed to send notification!');
            disp(ME.message);
        end
    end

    fprintf('🕒 %2d | 🔥 %.2f°C | ⚡ %.2fA | 🔋 %.2fV | Relay: %d | Buzzer: %d\n', ...
        t, sim_data(t,2), sim_data(t,3), voltage_value, sim_data(t,5), sim_data(t,6));

    pause(1/fs);
end

voltage_data = [sim_data(:,1), sim_data(:,4)];
assignin('base', 'voltage_data', voltage_data);

assignin('base', 'sim_data', sim_data);
```

```
disp('✅ sim_data and voltage_data successfully assigned to the base workspace.');


figure;
plot(sim_data(:,1), sim_data(:,2), '-r', 'LineWidth', 1.5); hold on;
plot(sim_data(:,1), sim_data(:,3), '-g', 'LineWidth', 1.5);
plot(sim_data(:,1), sim_data(:,4), '-b', 'LineWidth', 1.5);
plot(sim_data(:,1), sim_data(:,5) * max(sim_data(:,2:4), [], 'all'), '--m', 'LineWidth', 1.5);
plot(sim_data(:,1), sim_data(:,6) * max(sim_data(:,2:4), [], 'all'), '--k', 'LineWidth', 1.5);
hold off;


xlabel('Time (s)', 'FontSize', 12, 'FontWeight', 'bold');
ylabel('Values', 'FontSize', 12, 'FontWeight', 'bold');
legend('Temperature (°C)', 'Current (A)', 'Voltage (V)', 'Relay (On/Off)', 'Buzzer (On/Off)', 'Location', 'Best');
title('Simulated Sensor Data', 'FontSize', 14, 'FontWeight', 'bold');
grid on;


disp('✅ Simulation ended!');
```

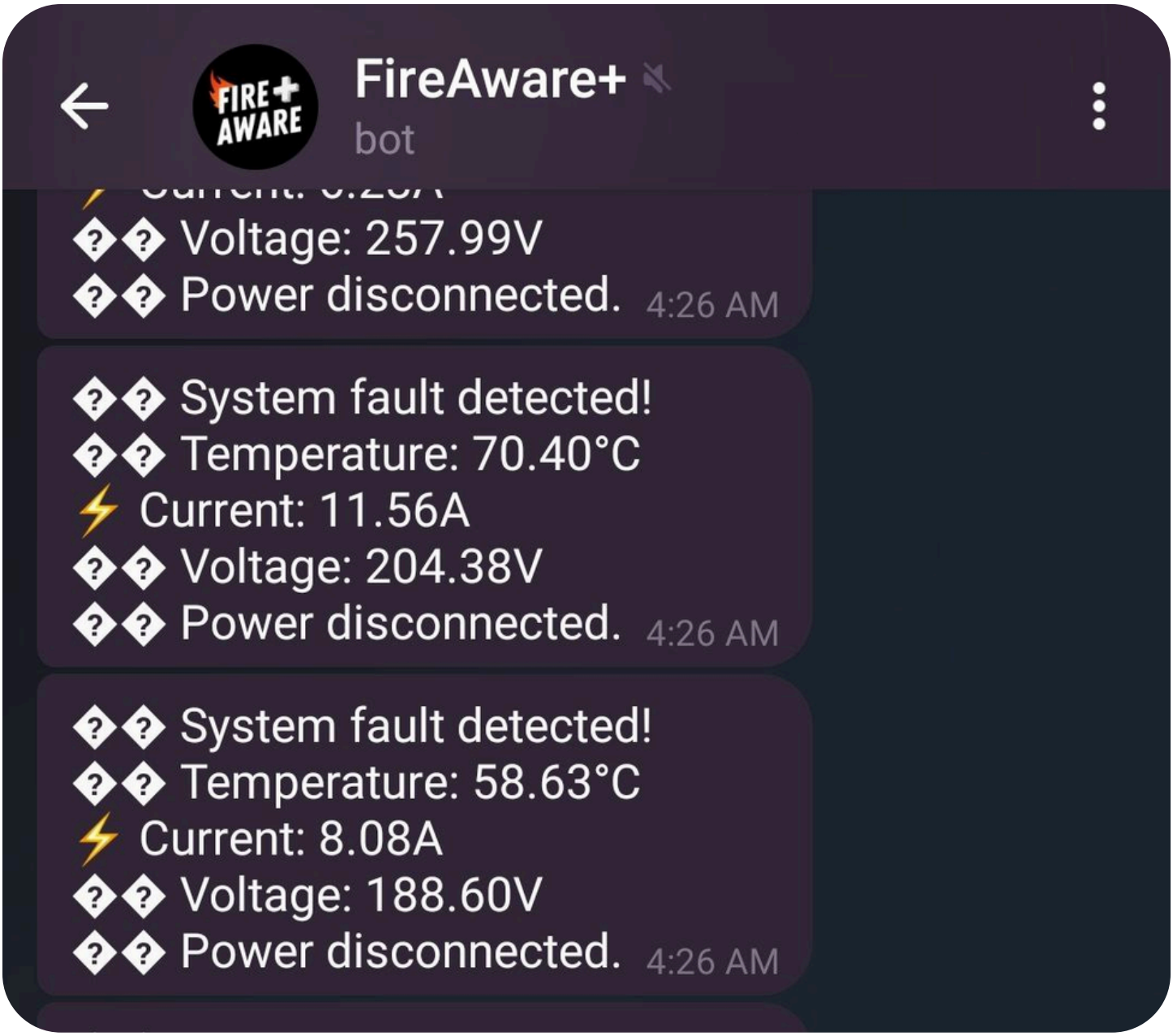
**Matlab Simulation Output**

```
🚀 Simulation started...
📨 Notification sent to Telegram!
🕒 1 | 🔥 56.64°C | ⚡ 9.44A | 🔋 214.58V | Relay: 0 | Buzzer: 1
📨 Notification sent to Telegram!
🕒 2 | 🔥 12.92°C | ⚡ 1.57A | 🔋 267.66V | Relay: 0 | Buzzer: 1
📨 Notification sent to Telegram!
🕒 3 | 🔥 61.09°C | ⚡ 16.99A | 🔋 235.47V | Relay: 0 | Buzzer: 1
🕒 4 | 🔥 22.27°C | ⚡ 7.64A | 🔋 199.87V | Relay: 1 | Buzzer: 0
📨 Notification sent to Telegram!
🕒 5 | 🔥 29.01°C | ⚡ 4.14A | 🔋 171.26V | Relay: 0 | Buzzer: 1
🕒 6 | 🔥 42.18°C | ⚡ 11.31A | 🔋 219.34V | Relay: 1 | Buzzer: 0
🕒 7 | 🔥 27.28°C | ⚡ 14.34A | 🔋 200.15V | Relay: 1 | Buzzer: 0
🕒 8 | 🔥 40.13°C | ⚡ 13.76A | 🔋 216.55V | Relay: 1 | Buzzer: 0
📨 Notification sent to Telegram!
```

## The Simulation with Simulink

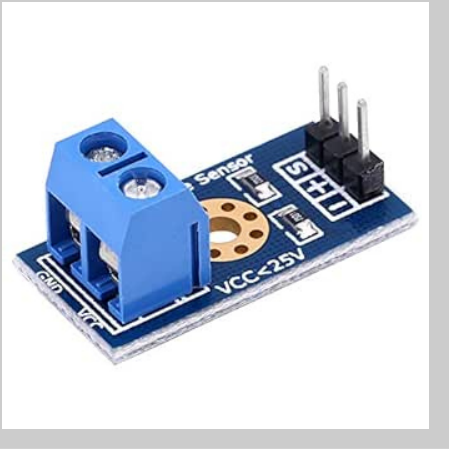



## Telegram Output (End User Aspect)



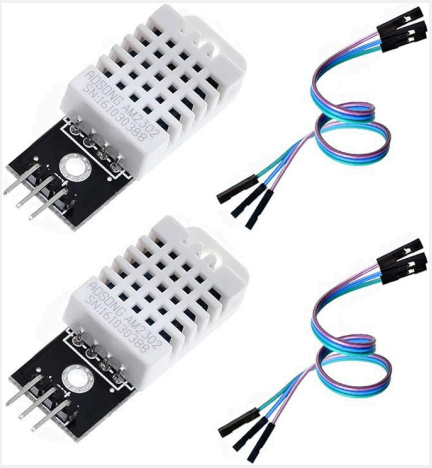

**Hardware Needed**

| No. | Component | Description |
|---|---|---|
| 1 | Temperature Sensor (DHT22 or LM35) | Measures ambient temperature to detect overheating in electrical circuits. |
| 2 | Current Sensor (ACS712) | Measures electrical current to detect overloads or electrical faults. |
| 3 | Voltage Sensor | Measures electrical voltage to monitor and analyze system stability. |
| 4 | Microcontroller (ESP32 or Arduino) | Processes data and analyzes system readings. |
| 5 | Relay Module | Automatically disconnects power when a fault is detected. |
| 6 | Wireless Communication Module (Wi-Fi or GSM) | Sends alerts to the user via the internet. |
| 7 | Power Supply Unit | Supplies power to the system components. |
| 8 | Display Screen (LCD or OLED) | Displays system status information. |
| 9 | Buzzer | Produces an audible alarm when a fault is detected. |
| 10 | System Enclosure | Protects electronic components from external damage. |
| 11 | Wires and Connectors | Electrical connections between components. |


Sensor
VCC-<25V


SRD-05VDC-SL-C


MAJESTY
GAMING
850W BRONZE
POWER SUPPLY




GND VCC SCL SDA


U1
D1
R1
C1
C2
GND OUTVCC
U1
D1
R1
C1
C2
GND OUTVCC