

ESD Essentials Term Paper Evaluation

Generated by Doxygen 1.14.0

1 Namespace Index	1
1.1 Package List	1
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Namespace Documentation	7
4.1 test1 Namespace Reference	7
4.1.1 Function Documentation	8
4.1.1.1 read_csv_data()	8
4.1.1.2 read_ultrasonic()	8
4.1.2 Variable Documentation	8
4.1.2.1 app	8
4.1.2.2 DHT_PIN	9
4.1.2.3 DHT_SENSOR	9
4.1.2.4 file_lock	9
4.1.2.5 filename	9
4.1.2.6 GAS_DO_PIN	9
4.1.2.7 mode	9
4.1.2.8 newline	10
4.1.2.9 root	10
4.1.2.10 ULTRASONIC_ECHO	10
4.1.2.11 ULTRASONIC_TRIG	10
4.1.2.12 writer	10
5 Class Documentation	11
5.1 test1.SensorApp Class Reference	11
5.1.1 Detailed Description	12
5.1.2 Constructor & Destructor Documentation	12
5.1.2.1 __init__()	12
5.1.3 Member Function Documentation	12
5.1.3.1 on_close()	12
5.1.3.2 sensor_loop()	13
5.1.3.3 start_sensor_thread()	13
5.1.3.4 update_gas_label()	13
5.1.3.5 update_plots()	13
5.1.3.6 update_temp_label()	14
5.1.4 Member Data Documentation	14
5.1.4.1 axs	14
5.1.4.2 canvas	14
5.1.4.3 fig	14

5.1.4.4	gas_slider	14
5.1.4.5	gas_threshold	14
5.1.4.6	gas_value_label	14
5.1.4.7	gas_values	15
5.1.4.8	levels	15
5.1.4.9	on_close	15
5.1.4.10	root	15
5.1.4.11	sensor_thread	15
5.1.4.12	status_label	15
5.1.4.13	temp_slider	15
5.1.4.14	temp_threshold	16
5.1.4.15	temp_value_label	16
5.1.4.16	temps	16
5.1.4.17	times	16
5.1.4.18	update_plots	16
6	File Documentation	17
6.1	test1.py File Reference	17
6.2	test1.py	18

Chapter 1

Namespace Index

1.1 Package List

Here are the packages with brief descriptions (if available):

test1	7
-----------------------	-------	-------------------

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

test1.SensorApp	A GUI application for real-time visualization and monitoring of sensor data	11
---------------------------------	---------------------------------------------------------------------------------------	--------------------

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

test1.py	17
--------------------------	-------	----

Chapter 4

Namespace Documentation

4.1 test1 Namespace Reference

Classes

- class **SensorApp**
A GUI application for real-time visualization and monitoring of sensor data.

Functions

- **read_ultrasonic ()**
Reads distance (in cm) from the HC-SR04 ultrasonic sensor.
- **read_csv_data ()**
Reads data from the CSV log file.

Variables

- int **DHT_PIN** = 17
DHT11 data pin.
- int **GAS_DO_PIN** = 27
MQ Gas Sensor digital output pin.
- int **ULTRASONIC_TRIG** = 23
Ultrasonic sensor TRIG pin.
- int **ULTRASONIC_ECHO** = 24
Ultrasonic sensor ECHO pin.
- **DHT_SENSOR** = Adafruit_DHT.DHT11
Sensor type for DHT.
- str **filename** = "essentials_log.csv"
CSV filename for data logging.
- **file_lock** = threading.Lock()
Thread lock to synchronize file access.
- **mode**
- **newline**
- **writer** = csv.writer(file)
- **root** = tk.Tk()
- **app** = SensorApp(root)

4.1.1 Function Documentation

4.1.1.1 `read_csv_data()`

```
test1.read_csv_data ()
```

Reads data from the CSV log file.

Returns

Four lists containing timestamps, temperature, gas, and level data.

Definition at line 117 of file [test1.py](#).

4.1.1.2 `read_ultrasonic()`

```
test1.read_ultrasonic ()
```

Reads distance (in cm) from the HC-SR04 ultrasonic sensor.

Returns

Distance in centimeters (float), or None if timeout occurs.

Definition at line 88 of file [test1.py](#).

Here is the caller graph for this function:



4.1.2 Variable Documentation

4.1.2.1 `app`

```
test1.app = SensorApp(root)
```

Definition at line 324 of file [test1.py](#).

4.1.2.2 DHT_PIN

```
int test1.DHT_PIN = 17
```

DHT11 data pin.

Definition at line [43](#) of file [test1.py](#).

4.1.2.3 DHT_SENSOR

```
test1.DHT_SENSOR = Adafruit_DHT.DHT11
```

Sensor type for DHT.

Definition at line [51](#) of file [test1.py](#).

4.1.2.4 file_lock

```
test1.file_lock = threading.Lock()
```

Thread lock to synchronize file access.

Definition at line [67](#) of file [test1.py](#).

4.1.2.5 filename

```
test1.filename = "essentials_log.csv"
```

CSV filename for data logging.

Definition at line [64](#) of file [test1.py](#).

4.1.2.6 GAS_DO_PIN

```
int test1.GAS_DO_PIN = 27
```

MQ Gas Sensor digital output pin.

Definition at line [45](#) of file [test1.py](#).

4.1.2.7 mode

```
test1.mode
```

Definition at line [72](#) of file [test1.py](#).

4.1.2.8 **newline**

```
test1.newline
```

Definition at line [78](#) of file [test1.py](#).

4.1.2.9 **root**

```
test1.root = tk.Tk()
```

Definition at line [323](#) of file [test1.py](#).

4.1.2.10 **ULTRASONIC_ECHO**

```
int test1.ULTRASONIC_ECHO = 24
```

Ultrasonic sensor ECHO pin.

Definition at line [49](#) of file [test1.py](#).

4.1.2.11 **ULTRASONIC_TRIG**

```
int test1.ULTRASONIC_TRIG = 23
```

Ultrasonic sensor TRIG pin.

Definition at line [47](#) of file [test1.py](#).

4.1.2.12 **writer**

```
test1.writer = csv.writer(file)
```

Definition at line [79](#) of file [test1.py](#).

Chapter 5

Class Documentation

5.1 test1.SensorApp Class Reference

A GUI application for real-time visualization and monitoring of sensor data.

Public Member Functions

- [`__init__`](#) (self, `root`)
Constructor for initializing the GUI layout and logic.
- [`update_temp_label`](#) (self, event=None)
Updates temperature slider label.
- [`update_gas_label`](#) (self, event=None)
Updates gas slider label.
- [`start_sensor_thread`](#) (self)
Starts a background thread for continuous sensor data acquisition.
- [`sensor_loop`](#) (self)
Sensor data acquisition loop (runs on a background thread).
- [`update_plots`](#) (self)
Updates live Matplotlib plots on the GUI.
- [`on_close`](#) (self)
Handles cleanup when closing the GUI.

Public Attributes

- `root` = `root`
- `temp_threshold` = `tk.DoubleVar(value=50)`
- `gas_threshold` = `tk.IntVar(value=1)`
- list `times` = []
- list `temps` = []
- list `gas_values` = []
- list `levels` = []
- `fig`
- `axs` = `plt.subplots(3, 1, figsize=(8, 6), sharex=True)`
- `canvas` = `FigureCanvasTkAgg(self.fig, master=root)`

- `temp_slider` = `ttk.Scale(root, from_=0, to=100, orient='horizontal', variable=self.temp_threshold, command=self.update_temp_label)`
- `temp_value_label` = `ttk.Label(root, text=f'{self.temp_threshold.get():.1f}')`
- `gas_slider` = `ttk.Scale(root, from_=0, to=1, orient='horizontal', variable=self.gas_threshold, command=self.update_gas_label)`
- `gas_value_label` = `ttk.Label(root, text=f'{int(self.gas_threshold.get())}')`
- `status_label` = `ttk.Label(root, text='', font=("Arial", 14))`
- `on_close`
- `sensor_thread` = `threading.Thread(target=self.sensor_loop, daemon=True)`
- `update_plots`

5.1.1 Detailed Description

A GUI application for real-time visualization and monitoring of sensor data.

This class creates a Tkinter window embedding live-updating Matplotlib plots for temperature, gas concentration, and water level readings.

It also continuously logs sensor data into a CSV file using a background thread.

Responsibilities:

- Acquire and log sensor data.
- Dynamically plot readings in real-time.
- Provide fault/anomaly detection.
- Allow runtime adjustment of sensor thresholds.

Definition at line 153 of file `test1.py`.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 `__init__()`

```
test1.SensorApp.__init__ (
    self,
    root)
```

Constructor for initializing the GUI layout and logic.

Parameters

<code>root</code>	The main Tkinter window.
-------------------	--------------------------

Definition at line 156 of file `test1.py`.

5.1.3 Member Function Documentation

5.1.3.1 on_close()

```
test1.SensorApp.on_close (
    self)
```

Handles cleanup when closing the GUI.

Definition at line 312 of file [test1.py](#).

5.1.3.2 sensor_loop()

```
test1.SensorApp.sensor_loop (
    self)
```

Sensor data acquisition loop (runs on a background thread).

Reads DHT11, MQ gas sensor, and ultrasonic distance data periodically. Logs results to CSV and updates internal memory.

Definition at line 216 of file [test1.py](#).

Here is the call graph for this function:



5.1.3.3 start_sensor_thread()

```
test1.SensorApp.start_sensor_thread (
    self)
```

Starts a background thread for continuous sensor data acquisition.

Definition at line 208 of file [test1.py](#).

5.1.3.4 update_gas_label()

```
test1.SensorApp.update_gas_label (
    self,
    event = None)
```

Updates gas slider label.

Definition at line 204 of file [test1.py](#).

5.1.3.5 update_plots()

```
test1.SensorApp.update_plots (
    self)
```

Updates live Matplotlib plots on the GUI.

Definition at line [262](#) of file [test1.py](#).

5.1.3.6 update_temp_label()

```
test1.SensorApp.update_temp_label (
    self,
    event = None)
```

Updates temperature slider label.

Definition at line [200](#) of file [test1.py](#).

5.1.4 Member Data Documentation

5.1.4.1 axs

```
test1.SensorApp.axs = plt.subplots(3, 1, figsize=(8, 6), sharex=True)
```

Definition at line [171](#) of file [test1.py](#).

5.1.4.2 canvas

```
test1.SensorApp.canvas = FigureCanvasTkAgg(self.fig, master=root)
```

Definition at line [174](#) of file [test1.py](#).

5.1.4.3 fig

```
test1.SensorApp.fig
```

Definition at line [171](#) of file [test1.py](#).

5.1.4.4 gas_slider

```
test1.SensorApp.gas_slider = ttk.Scale(root, from_=0, to=1, orient='horizontal', variable=self.gas_threshold, command=self.update_gas_label)
```

Definition at line [185](#) of file [test1.py](#).

5.1.4.5 **gas_threshold**

```
test1.SensorApp.gas_threshold = tk.IntVar(value=1)
```

Definition at line 162 of file [test1.py](#).

5.1.4.6 **gas_value_label**

```
test1.SensorApp.gas_value_label = ttk.Label(root, text=f"{int(self.gas_threshold.get())}")
```

Definition at line 187 of file [test1.py](#).

5.1.4.7 **gas_values**

```
test1.SensorApp.gas_values = []
```

Definition at line 167 of file [test1.py](#).

5.1.4.8 **levels**

```
test1.SensorApp.levels = []
```

Definition at line 168 of file [test1.py](#).

5.1.4.9 **on_close**

```
test1.SensorApp.on_close
```

Definition at line 197 of file [test1.py](#).

5.1.4.10 **root**

```
test1.SensorApp.root = root
```

Definition at line 157 of file [test1.py](#).

5.1.4.11 **sensor_thread**

```
test1.SensorApp.sensor_thread = threading.Thread(target=self.sensor_loop, daemon=True)
```

Definition at line 209 of file [test1.py](#).

5.1.4.12 **status_label**

```
test1.SensorApp.status_label = ttk.Label(root, text="", font=("Arial", 14))
```

Definition at line 190 of file [test1.py](#).

5.1.4.13 **temp_slider**

```
test1.SensorApp.temp_slider = ttk.Scale(root, from_=0, to=100, orient='horizontal', variable=self.←  
temp_threshold, command=self.update_temp_label)
```

Definition at line 179 of file [test1.py](#).

5.1.4.14 **temp_threshold**

```
test1.SensorApp.temp_threshold = tk.DoubleVar(value=50)
```

Definition at line 161 of file [test1.py](#).

5.1.4.15 **temp_value_label**

```
test1.SensorApp.temp_value_label = ttk.Label(root, text=f"{self.temp_threshold.get():.1f}")
```

Definition at line 181 of file [test1.py](#).

5.1.4.16 **temps**

```
test1.SensorApp.temps = []
```

Definition at line 166 of file [test1.py](#).

5.1.4.17 **times**

```
test1.SensorApp.times = []
```

Definition at line 165 of file [test1.py](#).

5.1.4.18 **update_plots**

```
test1.SensorApp.update_plots
```

Definition at line 309 of file [test1.py](#).

The documentation for this class was generated from the following file:

- [test1.py](#)

Chapter 6

File Documentation

6.1 test1.py File Reference

Classes

- class `test1.SensorApp`
A GUI application for real-time visualization and monitoring of sensor data.

Namespaces

- namespace `test1`

Functions

- `test1.read_ultrasonic ()`
Reads distance (in cm) from the HC-SR04 ultrasonic sensor.
- `test1.read_csv_data ()`
Reads data from the CSV log file.

Variables

- int `test1.DHT_PIN = 17`
DHT11 data pin.
- int `test1.GAS_DO_PIN = 27`
MQ Gas Sensor digital output pin.
- int `test1.ULTRASONIC_TRIG = 23`
Ultrasonic sensor TRIG pin.
- int `test1.ULTRASONIC_ECHO = 24`
Ultrasonic sensor ECHO pin.
- `test1.DHT_SENSOR = Adafruit_DHT.DHT11`
Sensor type for DHT.
- str `test1.filename = "essentials_log.csv"`
CSV filename for data logging.
- `test1.file_lock = threading.Lock()`
Thread lock to synchronize file access.
- `test1.mode`
- `test1.newline`
- `test1.writer = csv.writer(file)`
- `test1.root = tk.Tk()`
- `test1.app = SensorApp(root)`

6.2 test1.py

[Go to the documentation of this file.](#)

```

00001
00025
00026 import tkinter as tk
00027 from tkinter import ttk
00028 from tkinter import messagebox
00029 import RPi.GPIO as GPIO
00030 import Adafruit_DHT
00031 import time
00032 import csv
00033 from datetime import datetime
00034 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
00035 import matplotlib.pyplot as plt
00036 import threading
00037
00038 # =====
00039 #           GPIO AND SENSOR CONFIGURATION
00040 # =====
00041
00042
00043 DHT_PIN = 17
00044
00045 GAS_DO_PIN = 27
00046
00047 ULTRASONIC_TRIG = 23
00048
00049 ULTRASONIC_ECHO = 24
00050
00051 DHT_SENSOR = Adafruit_DHT.DHT11
00052
00053 # GPIO mode and pin setup
00054 GPIO.setmode(GPIO.BCM)
00055 GPIO.setup(GAS_DO_PIN, GPIO.IN)
00056 GPIO.setup(ULTRASONIC_TRIG, GPIO.OUT)
00057 GPIO.setup(ULTRASONIC_ECHO, GPIO.IN)
00058
00059 # =====
00060 #           CSV FILE INITIALIZATION
00061 # =====
00062
00063
00064 filename = "essentials_log.csv"
00065
00066
00067 file_lock = threading.Lock()
00068
00069 # @brief Initialize CSV file with header if empty
00070 with file_lock:
00071     try:
00072         with open(filename, mode='r') as f:
00073             if f.read(1):
00074                 pass
00075             else:
00076                 raise FileNotFoundError
00077     except (FileNotFoundException, IOError):
00078         with open(filename, mode='w', newline='') as file:
00079             writer = csv.writer(file)
00080             writer.writerow(["Timestamp", "TEMP", "PPM", "LEVEL", "Anomaly"])
00081
00082 # =====
00083 #           SENSOR READING FUNCTIONS
00084 # =====
00085
00086
00088 def read_ultrasonic():
00089     GPIO.output(ULTRASONIC_TRIG, False)
00090     time.sleep(0.1)
00091
00092     GPIO.output(ULTRASONIC_TRIG, True)
00093     time.sleep(0.00001)
00094     GPIO.output(ULTRASONIC_TRIG, False)
00095
00096     pulse_start = time.time()
00097     timeout = pulse_start + 0.04
00098     while GPIO.input(ULTRASONIC_ECHO) == 0:
00099         pulse_start = time.time()
00100     if pulse_start > timeout:
00101         return None
00102
00103     pulse_end = time.time()
00104     timeout = pulse_end + 0.04
00105     while GPIO.input(ULTRASONIC_ECHO) == 1:
00106         pulse_end = time.time()

```

```

00107     if pulse_end > timeout:
00108         return None
00109
00110     pulse_duration = pulse_end - pulse_start
00111     distance_cm = pulse_duration * 17150
00112     return round(distance_cm, 2)
00113
00114
00115
00116 def read_csv_data():
00117     times, temps, gas_values, levels = [], [], [], []
00118     with file_lock:
00119         try:
00120             with open(filename, mode='r') as file:
00121                 reader = csv.DictReader(file)
00122                 for row in reader:
00123                     try:
00124                         times.append(datetime.strptime(row["Timestamp"], "%Y-%m-%d %H:%M:%S"))
00125                         temps.append(float(row["TEMP"])) if row["TEMP"] != "N/A" else None
00126                         gas_values.append(int(row["PPM"]))
00127                         levels.append(float(row["LEVEL"])) if row["LEVEL"] != "N/A" else None
00128                     except Exception:
00129                         continue
00130                     except FileNotFoundError:
00131                         pass
00132     return times, temps, gas_values, levels
00133
00134
00135
00136 # =====
00137 #           MAIN APPLICATION CLASS
00138 # =====
00139
00140
00141
00142 class SensorApp:
00143
00144     def __init__(self, root):
00145         self.root = root
00146         self.root.title("Sensor Monitoring and Visualization")
00147
00148         # Threshold variables
00149         self.temp_threshold = tk.DoubleVar(value=50)
00150         self.gas_threshold = tk.IntVar(value=1)
00151
00152         # Data containers for live updates
00153         self.times = []
00154         self.temps = []
00155         self.gas_values = []
00156         self.levels = []
00157
00158         # ----- GUI Configuration -----
00159         self.fig, self.axs = plt.subplots(3, 1, figsize=(8, 6), sharex=True)
00160         self.fig.tight_layout(pad=3)
00161
00162         self.canvas = FigureCanvasTkAgg(self.fig, master=root)
00163         self.canvas.get_tk_widget().grid(row=0, column=0, columnspan=6, padx=10, pady=10)
00164
00165         # Sliders and labels for thresholds
00166         ttk.Label(root, text="Temperature Threshold (°C)").grid(row=1, column=0, sticky="w", padx=10)
00167         self.temp_slider = ttk.Scale(root, from_=0, to=100, orient='horizontal',
00168                                     variable=self.temp_threshold, command=self.update_temp_label)
00169         self.temp_slider.grid(row=1, column=1, sticky="ew", padx=(0, 5))
00170         self.temp_value_label = ttk.Label(root, text=f"{self.temp_threshold.get():.1f}")
00171         self.temp_value_label.grid(row=1, column=2, sticky="w")
00172
00173         ttk.Label(root, text="Gas Threshold (PPM, 0 or 1)").grid(row=1, column=3, sticky="w", padx=10)
00174         self.gas_slider = ttk.Scale(root, from_=0, to=1, orient='horizontal',
00175                                     variable=self.gas_threshold, command=self.update_gas_label)
00176         self.gas_slider.grid(row=1, column=4, sticky="ew", padx=(0, 5))
00177         self.gas_value_label = ttk.Label(root, text=f"{int(self.gas_threshold.get())}")
00178         self.gas_value_label.grid(row=1, column=5, sticky="w")
00179
00180         self.status_label = ttk.Label(root, text="", font=("Arial", 14))
00181         self.status_label.grid(row=2, column=0, columnspan=6, pady=10)
00182
00183         # Start live updates
00184         self.update_plots()
00185         self.start_sensor_thread()
00186
00187         root.protocol("WM_DELETE_WINDOW", self.on_close)
00188
00189
00190     def update_temp_label(self, event=None):
00191         self.temp_value_label.config(text=f"{self.temp_threshold.get():.1f}")
00192
00193     def update_gas_label(self, event=None):
00194
00195
00196
00197
00198
00199
00200
00201
00202
00203
00204

```

```

00205         self.gas_value_label.config(text=f"{int(round(self.gas_threshold.get()))}"))
00206
00207
00208     def start_sensor_thread(self):
00209         self.sensor_thread = threading.Thread(target=self.sensor_loop, daemon=True)
00210         self.sensor_thread.start()
00211
00212
00213     def sensor_loop(self):
00214         while True:
00215             timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
00216
00217             humidity, temperature = Adafruit_DHT.read(DHT_SENSOR, DHT_PIN)
00218             temp_val = temperature if temperature is not None else "N/A"
00219
00220             gas_state = GPIO.input(GAS_DO_PIN)
00221             gas_detected = (gas_state == 0)
00222             ppm_val = 1 if gas_detected else 0
00223
00224             level = read_ultrasonic()
00225             level_val = level if level is not None else "N/A"
00226
00227             # Simple anomaly detection logic
00228             anomaly = "No"
00229             if temp_val == "N/A" or level_val == "N/A":
00230                 anomaly = "Yes"
00231             else:
00232                 if not (0 <= temp_val <= 50) or not (0 <= level_val <= 400):
00233                     anomaly = "Yes"
00234
00235             # Thread-safe CSV writing
00236             with file_lock:
00237                 with open(filename, mode='a', newline="") as file:
00238                     writer = csv.writer(file)
00239                     writer.writerow([timestamp, temp_val, ppm_val, level_val, anomaly])
00240
00241             # Data update
00242             dt = datetime.strptime(timestamp, "%Y-%m-%d %H:%M:%S")
00243             self.times.append(dt)
00244             self.temps.append(float(temp_val) if temp_val != "N/A" else None)
00245             self.gas_values.append(ppm_val)
00246             self.levels.append(float(level_val) if level_val != "N/A" else None)
00247
00248             # Maintain buffer size
00249             max_len = 100
00250             if len(self.times) > max_len:
00251                 self.times = self.times[-max_len:]
00252                 self.temps = self.temps[-max_len:]
00253                 self.gas_values = self.gas_values[-max_len:]
00254                 self.levels = self.levels[-max_len:]
00255
00256             time.sleep(0.5)
00257
00258
00259
00260
00261
00262     def update_plots(self):
00263         for ax in self.axs:
00264             ax.clear()
00265
00266             # --- Temperature ---
00267             if self.times and any(t is not None for t in self.temps):
00268                 temps_clean = [t if t is not None else float('nan') for t in self.temps]
00269                 self.axs[0].plot(self.times, temps_clean, 'r-', label='Temperature (°C)')
00270                 self.axs[0].axhline(self.temp_threshold.get(), color='r', linestyle='--', label='Temp Threshold')
00271                 self.axs[0].set_ylabel("Temperature (°C)")
00272                 self.axs[0].legend(loc='upper right')
00273                 self.axs[0].grid(True)
00274
00275             # --- Gas PPM ---
00276             if self.times:
00277                 self.axs[1].step(self.times, self.gas_values, 'g-', label='Gas PPM')
00278                 self.axs[1].axhline(self.gas_threshold.get(), color='g', linestyle='--', label='Gas Threshold')
00279                 self.axs[1].set_ylabel("Gas PPM")
00280                 self.axs[1].set_ylim(-0.1, 1.1)
00281                 self.axs[1].legend(loc='upper right')
00282                 self.axs[1].grid(True)
00283
00284             # --- Ultrasonic Level ---
00285             if self.times and any(l is not None for l in self.levels):
00286                 levels_clean = [l if l is not None else float('nan') for l in self.levels]
00287                 self.axs[2].plot(self.times, levels_clean, 'b-', label='Level (cm)')
00288                 self.axs[2].set_ylabel("Level (cm)")
00289                 self.axs[2].set_xlabel("Time")
00290                 self.axs[2].legend(loc='upper right')
00291                 self.axs[2].grid(True)
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319
00320
00321
00322
00323
00324
00325
00326
00327
00328
00329
00330
00331
00332
00333
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343
00344
00345
00346
00347
00348
00349
00350
00351
00352
00353
00354
00355
00356
00357
00358
00359
00360
00361
00362
00363
00364
00365
00366
00367
00368
00369
00370
00371
00372
00373
00374
00375
00376
00377
00378
00379
00380
00381
00382
00383
00384
00385
00386
00387
00388
00389
00390
00391
00392
00393
00394
00395
00396
00397
00398
00399
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428
00429
00430
00431
00432
00433
00434
00435
00436
00437
00438
00439
00440
00441
00442
00443
00444
00445
00446
00447
00448
00449
00450
00451
00452
00453
00454
00455
00456
00457
00458
00459
00460
00461
00462
00463
00464
00465
00466
00467
00468
00469
00470
00471
00472
00473
00474
00475
00476
00477
00478
00479
00480
00481
00482
00483
00484
00485
00486
00487
00488
00489
00490
00491
00492
00493
00494
00495
00496
00497
00498
00499
00500
00501
00502
00503
00504
00505
00506
00507
00508
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573
00574
00575
00576
00577
00578
00579
00580
00581
00582
00583
00584
00585
00586
00587
00588
00589
00590
00591
00592
00593
00594
00595
00596
00597
00598
00599
00600
00601
00602
00603
00604
00605
00606
00607
00608
00609
00610
00611
00612
00613
00614
00615
00616
00617
00618
00619
00620
00621
00622
00623
00624
00625
00626
00627
00628
00629
00630
00631
00632
00633
00634
00635
00636
00637
00638
00639
00640
00641
00642
00643
00644
00645
00646
00647
00648
00649
00650
00651
00652
00653
00654
00655
00656
00657
00658
00659
00660
00661
00662
00663
00664
00665
00666
00667
00668
00669
00670
00671
00672
00673
00674
00675
00676
00677
00678
00679
00680
00681
00682
00683
00684
00685
00686
00687
00688
00689
00690
00691
00692
00693
00694
00695
00696
00697
00698
00699
00700
00701
00702
00703
00704
00705
00706
00707
00708
00709
00710
00711
00712
00713
00714
00715
00716
00717
00718
00719
00720
00721
00722
00723
00724
00725
00726
00727
00728
00729
00730
00731
00732
00733
00734
00735
00736
00737
00738
00739
00740
00741
00742
00743
00744
00745
00746
00747
00748
00749
00750
00751
00752
00753
00754
00755
00756
00757
00758
00759
00760
00761
00762
00763
00764
00765
00766
00767
00768
00769
00770
00771
00772
00773
00774
00775
00776
00777
00778
00779
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01
```

```
00293     self.fig.autofmt_xdate()
00294     self.canvas.draw()
00295
00296     # Fault detection and GUI update
00297     fault_msg = ""
00298     if self.temps and self.gas_values and self.levels:
00299         last_temp = self.temps[-1]
00300         last_gas = self.gas_values[-1]
00301         last_level = self.levels[-1]
00302
00303         if last_temp is None or last_level is None:
00304             fault_msg = "Critical fault: Sensor data missing!"
00305         elif last_temp > self.temp_threshold.get() or last_gas > self.gas_threshold.get() or not
00306             (0 <= last_level <= 400):
00307                 fault_msg = "Critical fault detected!"
00308
00309     self.status_label.config(text=fault_msg, foreground='red' if fault_msg else 'green')
00310     self.root.after(1000, self.update_plots)
00311
00312     def on_close(self):
00313         GPIO.cleanup()
00314         self.root.destroy()
00315
00316
00317 # =====
00318 #           MAIN PROGRAM ENTRY POINT
00319 # =====
00320
00321
00322 if __name__ == "__main__":
00323     root = tk.Tk()
00324     app = SensorApp(root)
00325     root.mainloop()
```

