REAL TIME ENVIRONMENTAL SENSING IN AIR QUALITY

MEMBERS:

Ashwin Sudhakaran

Balaji NM
hhuh
Dillan RV

Hafeezur Rahiman

AIM:

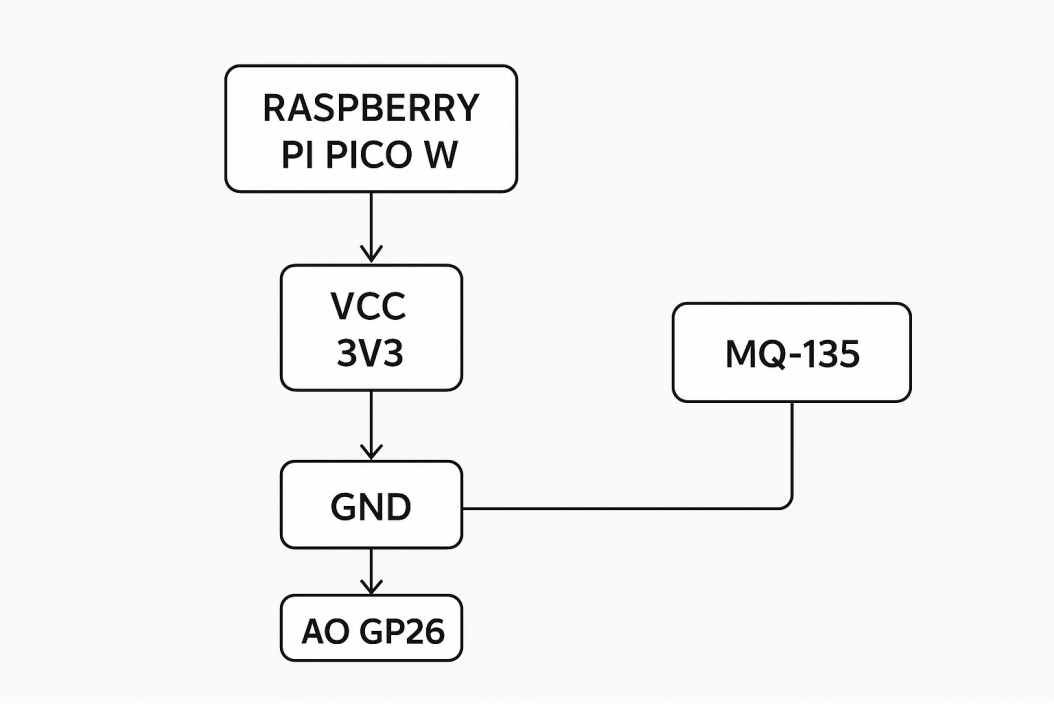
The aim of REAL TIME ENVIRONMENTAL SENSING IN AIR QUALITY is to continuously monitor air quality and environmental parameters , providing timely alerts to ensure health , safety for a cleaner environment.

COMPONENTS REQUIRED:

- **MQ-135** - Air quality sensor (detects CO_2, NH_3, NOx, alcohol, benzene,smoke)
- **Raspberry Pi pico**-The **Raspberry Pi Pico** is used as the main controller to collect sensor data, process it into AQI, and send it to displays, alerts, or cloud platforms in real time.
- **Bread board-**The breadboard is used to easily connect and test sensors, microcontroller, and modules without soldering during the prototyping stage of the air quality sensing system.
- **OLED display**-The OLED display is used to show real-time air quality readings (PM2.5, CO_2, AQI, etc.) and status alerts directly on the device for quick monitoring.
- **Buzzer**-The buzzer is used to give an audible alert when air quality levels cross the safe limit, warning users immediately of hazardous conditions.

Component	Pin	Connects To
MQ-135 Sensor	VCC	3V3
	GND	GND
	AO	GP26
	DO (optional)	GP15
Breadboard	Power Rails (+)	3V3
	Power Rails (–)	GND
OLED SSD1306	VCC	3V3
	GND	GND
	SCL	GP5
	b.b1	GP4

FLOW CHART:



CODE

```
# main.py -- Raspberry Pi Pico W + MQ-5 Gas Sensor + SSD1306 OLED + Buzzer
# ----------------------------------------------------------
# Requirements:
#   - MicroPython firmware on Pico W
#   - ssd1306.py library copied to Pico (in /lib folder)
#
# Wiring:
#   MQ-5:
#     VCC -> 5V (VBUS pin 40 or VSYS pin 39)
#     GND -> GND
#     A0  -> GP26 (ADC0, pin 31)  [⚠️ MUST use resistor divider to max 3.3V]
#
#   OLED (SSD1306 I²C, 128x64):
#     VCC -> 3.3V (pin 36)
#     GND -> GND
#     SDA -> GP0 (pin 1)
#     SCL -> GP1 (pin 2)
#
#   Buzzer:
#     + -> GP15 (pin 21)
#     - -> GND
# ----------------------------------------------------------
```

```
import time
from machine import Pin, ADC, I2C
from ssd1306 import SSD1306_I2C

# ---------- CONFIG ----------
ADC_PIN = 26        # GP26 (ADC0)
BUZZER_PIN = 15     # GP15
I2C_SDA = 0         # GP0
I2C_SCL = 1         # GP1
OLED_WIDTH = 128
OLED_HEIGHT = 64
CALIB_FILE = "r0.txt"

SAMPLES = 50
SAMPLE_INTERVAL = 0.2  # sec
PUBLISH_INTERVAL = 5   # sec
ALARM_RATIO_THRESHOLD = 0.5  # Rs/R0 threshold

# ---------- Initialize hardware ----------
adc = ADC(ADC_PIN)
buzzer = Pin(BUZZER_PIN, Pin.OUT)
i2c = I2C(0, sda=Pin(I2C_SDA), scl=Pin(I2C_SCL))
oled = SSD1306_I2C(OLED_WIDTH, OLED_HEIGHT, i2c)

# ---------- Helper functions ----------
def read_adc_voltage():
    raw = adc.read_u16()
    voltage = raw * 3.3 / 65535
    return raw, voltage

def get_sensor_resistance(voltage, vcc=5.0, rl=10.0):
    if voltage <= 0:
        return None
    try:
        Rs = rl * (vcc - voltage) / voltage
        return Rs
    except:
        return None

def save_r0(r0):
    with open(CALIB_FILE, "w") as f:
        f.write(str(r0))

def load_r0():
    try:
        with open(CALIB_FILE, "r") as f:
            return float(f.read().strip())
    except:
        return None
```

```
# Approximate LPG curve constants
A_LPG = 0.45
B_LPG = -0.38

def approx_ppm_from_ratio(rs_r0, a=A_LPG, b=B_LPG):
    if rs_r0 <= 0:
        return None
    try:
        ppm = (rs_r0 / a) ** (1.0 / b)
        return ppm
    except:
        return None

# ---------- Calibration ----------
def calibrate_r0(samples=50, sample_interval=0.2, vcc=5.0, rl=10.0):
    print("Calibrating R0: place sensor in clean air (no gas)...")
    time.sleep(2)
    rs_vals = []
    for i in range(samples):
        raw, v = read_adc_voltage()
        Rs = get_sensor_resistance(v, vcc=vcc, rl=rl)
        if Rs is not None:
            rs_vals.append(Rs)
        time.sleep(sample_interval)
    if not rs_vals:
        raise Exception("Calibration failed")
    avg_rs = sum(rs_vals) / len(rs_vals)
    R0 = avg_rs / 9.8  # Clean air factor from datasheet
    save_r0(R0)
    print("Calibration done. R0 =", R0)
    return R0

# ---------- Main loop ----------
def main():
    # Load or calibrate R0
    R0 = load_r0()
    if R0 is None:
        try:
            R0 = calibrate_r0(SAMPLES, SAMPLE_INTERVAL)
        except Exception as e:
            print("Calibration failed:", e)
            return

    oled.fill(0)
    oled.text("MQ-5 Gas Monitor", 0, 0)
    oled.text("Initializing...", 0, 16)
    oled.show()
    time.sleep(2)

    while True:
        rs_vals = []
```

```
        for _ in range(8):
            raw, v = read_adc_voltage()
            Rs = get_sensor_resistance(v, vcc=5.0, rl=10.0)
            if Rs is not None:
                rs_vals.append(Rs)
            time.sleep(0.05)

        if not rs_vals:
            time.sleep(PUBLISH_INTERVAL)
            continue

        Rs_avg = sum(rs_vals) / len(rs_vals)
        ratio = Rs_avg / R0
        ppm = approx_ppm_from_ratio(ratio)

        # Alarm
        alarm = False
        if ratio <= ALARM_RATIO_THRESHOLD:
            alarm = True
            for _ in range(3):
                buzzer.value(1)
                time.sleep(0.12)
                buzzer.value(0)
                time.sleep(0.08)

        # Format display lines
        line1 = "Rs:{:.1f} R0:{:.1f}".format(Rs_avg, R0)
        line2 = "Ratio:{:.2f}".format(ratio)
        line3 = "LPG~{:.0f}ppm".format(ppm) if ppm else "No PPM data"
        line4 = "!!! ALARM !!!" if alarm else ""

        # Print to REPL
        print(line1)
        print(line2)
        print(line3, line4)
        print("-" * 30)

        # Show on OLED
        oled.fill(0)
        oled.text(line1, 0, 0)
        oled.text(line2, 0, 16)
        oled.text(line3, 0, 32)
        if alarm:
            oled.text(line4, 0, 48)
        oled.show()

        time.sleep(PUBLISH_INTERVAL)

# ---------- Run ----------
if __name__ == "__main__":
    try:
```

```
    main()
except KeyboardInterrupt:
    print("Stopped by user")
```

,