

```

1  """Gas detection for Raspberry Pi using ADS1x15 and MQ-2 sensors."""
2
3  # pylint: disable=C0103
4  # pylint: disable=R0201
5
6  import math
7  import time
8
9  import board
10 import busio
11
12 from adafruit_ads1x15.analog_in import AnalogIn
13 from adafruit_ads1x15 import ads1015
14 from adafruit_ads1x15 import ads1115
15
16 # Convertors
17 ADS1015 = ads1015.ADS1015
18 ADS1115 = ads1115.ADS1115
19
20 # Pins
21 P0 = 0
22 P1 = 1
23 P2 = 2
24 P3 = 3
25
26 # Address of convertor
27 ADDRESS = 0x48
28
29 class GasDetection:
30     """Gas detection class."""
31
32     # Load resistance on the board in kilo ohms
33     LOAD_RESISTANCE = 5
34
35     # Clean air factor from the chart in the datasheet
36     CLEAN_AIR_FACTOR = 9.6
37
38     # Identification of gases
39     CO_GAS = 0
40     H2_GAS = 1
41     CH4_GAS = 2
42     LPG_GAS = 3
43     PROPANE_GAS = 4
44     ALCOHOL_GAS = 5
45     SMOKE_GAS = 6
46
47     # Calculated logarithm for the left point and
48     # slope from the curve, using ten logarithm
49     # and the two-point form
50     #
51     # More details about calculating:
52     # https://tutorials-raspberrypi.com/configure-and-read-out-the-raspberry-pi-gas-sensor-mq-2/
53     CO_CURVE = [2.30775, 0.71569, -0.33539]
54     H2_CURVE = [2.30776, 0.71895, -0.33539]
55     CH4_CURVE = [2.30987, 0.48693, -0.37459]
56     LPG_CURVE = [2.30481, 0.20588, -0.46621]
57     PROPANE_CURVE = [2.30366, 0.23203, -0.46202]
58     ALCOHOL_CURVE = [2.30704, 0.45752, -0.37398]
59     SMOKE_CURVE = [2.30724, 0.53268, -0.44082]
60
61     # Number of samples and time between them in milliseconds for calibration
62     CALIBRATION_SAMPLE_NUMBER = 50
63     CALIBRATION_SAMPLE_INTERVAL = 500
64
65     # Number of samples and time between them in milliseconds for reading
66     READ_SAMPLE_NUMBER = 5
67     READ_SAMPLE_INTERVAL = 50
68
69     # Analog to digital channel
70     channel = None
71
72     # Ro value of the sensor
73     ro = None
74
75     def __init__(self, convertor=ADS1115, pin=P0, address=ADDRESS, ro=None):

```

```

77 Initialize the class.
78
79 Input:
80 -- convertor -- Convertor to use. Must be one of ADS1x15. Default is ADS1115.
81 -- pin -- Pin of ADC convertor to use. Must be one of supported pins. Default is P0.
82 -- address -- Address of ADC convertor. Must be one of I2C addresses. Default is 0x48
83 -- ro -- Ro value of the sensor. Must be valid ro value. Default is to calibrate it.
84 """
85
86 i2c = busio.I2C(board.SCL, board.SDA)
87 adc = convertor(i2c=i2c, address=address)
88
89 self.channel = AnalogIn(adc, pin)
90
91 if ro:
92     self.ro = ro
93 else:
94     self.ro = self.calibrate()
95
96 def __read(self, number=None, interval=None):
97     """
98     Read sensor resistance from ADC voltage.
99
100     This function uses :func:`gas_detection.GasDetection.__calculate_resistance` to
101     caculate the sensor resistance (Rs). The resistance changes as the sensor is in the
102     different concentration of the target gas.
103
104     Input:
105     -- number -- Number of samples. Default is 5.
106     -- interval -- Time between samples in milliseconds. Default is 50.
107
108     Output:
109     -- Sensor resistance.
110     """
111
112     number = number if number else self.READ_SAMPLE_NUMBER
113     interval = interval if interval else self.READ_SAMPLE_INTERVAL
114
115     rs = 0
116
117     for _ in range(number):
118         rs += self.__calculate_resistance(self.channel.voltage)
119         time.sleep(interval / 1000)
120
121     rs = rs / number
122
123     return rs
124
125 def __calculate_resistance(self, voltage, resistance=None):
126     """
127     Calculate sensor resistance from ADC voltage.
128
129     The sensor and the load resistor forms a voltage divider. Given the voltage
130     across the load resistor and its resistance, the resistance of the sensor
131     could be derived.
132
133     Input:
134     -- voltage -- Voltage from ADC convertor.
135     -- resistance -- Load resistance on the board in kilo ohms. Default is 5.
136
137     Output:
138     -- Calculated sensor resistance.
139     """
140
141     resistance = resistance if resistance else self.LOAD_RESISTANCE
142
143     return float(resistance * (1023.0 - voltage) / float(voltage))
144
145 def __calculate_percentage(self, ratio, curve):
146     """
147     Calculate percentage from gas curve.
148
149     Calculate percentage from gas curve by using the slope and a point of the
150     line. The x (logarithmic value of ppm) of the line could be derived if y
151     ('ratio') is provided. As it is a logarithmic coordinate, power of 10 is
152     used to convert the result to non-logarithmic value.
153
154     Input:

```

```

155     -- ratio -- Rs divided by Ro.
156     -- curve -- The curve of the target gas.
157
158     Output:
159     -- Percentage of the target gas in ppm.
160     """
161
162     return math.pow(
163         10,
164         ((math.log(ratio) - curve[1]) / curve[2]) + curve[0]
165     )
166
167 def __calculate_gas_percentage(self, ratio, gas):
168     """
169     https://github.com/tutRPi/Raspberry-Pi-Gas-Sensor-MQ/blob/master/mq.py#L120
170
171     Get percentage of the target gas.
172
173     This function uses :func:`~gas_detection.GasDetection.__calculate_percentage` to
174     calculate percentage in ppm (parts per million) of the target gas by it's curve.
175
176     Input:
177     -- ratio -- Rs divided by Ro.
178     -- gas -- Identification of the target gas.
179
180     Output:
181     -- Percentage of the target gas in ppm.
182     """
183
184     if gas == self.CO_GAS:
185         ppm = self.__calculate_percentage(ratio, self.CO_CURVE)
186     elif gas == self.H2_GAS:
187         ppm = self.__calculate_percentage(ratio, self.H2_CURVE)
188     elif gas == self.CH4_GAS:
189         ppm = self.__calculate_percentage(ratio, self.CH4_CURVE)
190     elif gas == self.LPG_GAS:
191         ppm = self.__calculate_percentage(ratio, self.LPG_CURVE)
192     elif gas == self.PROPANE_GAS:
193         ppm = self.__calculate_percentage(ratio, self.PROPANE_CURVE)
194     elif gas == self.ALCOHOL_GAS:
195         ppm = self.__calculate_percentage(ratio, self.ALCOHOL_CURVE)
196     elif gas == self.SMOKE_GAS:
197         ppm = self.__calculate_percentage(ratio, self.SMOKE_CURVE)
198     else:
199         ppm = 0
200
201     return ppm
202
203 def calibrate(self, number=None, interval=None, factor=None):
204     """
205     Calibrate sensor.
206
207     This function assumes that the sensor is in clean air. It uses
208     :func:`~gas_detection.GasDetection.__calculate_resistance` to
209     caculate the sensor resistance (Rs) and divide it by clean
210     air factor.
211
212     Input:
213     -- number -- Number of samples. Default is 50.
214     -- interval -- Time between samples in milliseconds. Default is 500.
215     -- factor -- The clean air factor. Default is 9.6.
216
217     Output:
218     -- The ro value of sensor.
219     """
220
221     number = number if number else self.CALIBRATION_SAMPLE_NUMBER
222     interval = interval if interval else self.CALIBRATION_SAMPLE_INTERVAL
223     factor = factor if factor else self.CLEAN_AIR_FACTOR
224
225     rs = 0
226
227     for _ in range(number):
228         rs += self.__calculate_resistance(self.channel.voltage)
229         time.sleep(interval / 1000)
230
231     rs = rs / number

```

```

208 :func: '~gas_detection.GasDetection.__calculate_resistance' to
209 calculate the sensor resistance (Rs) and divide it by clean
210 air factor.
211
212 Input:
213 -- number -- Number of samples. Default is 50.
214 -- interval -- Time between samples in milliseconds. Default is 500.
215 -- factor -- The clean air factor. Default is 9.6.
216
217 Output:
218 -- The ro value of sensor.
219 """
220
221 number = number if number else self.CALIBRATION_SAMPLE_NUMBER
222 interval = interval if interval else self.CALIBRATION_SAMPLE_INTERVAL
223 factor = factor if factor else self.CLEAN_AIR_FACTOR
224
225 rs = 0
226
227 for _ in range(number):
228     rs += self.__calculate_resistance(self.channel.voltage)
229     time.sleep(interval / 1000)
230
231 rs = rs / number
232
233 return rs / factor
234
235 def percentage(self):
236     """
237     Get gas percentage of gases.
238
239     This function uses :func: '~gas_detection.GasDetection.__calculate_gas_percentage' to
240     the percentage of supported gases in ppm (parts per million).
241
242     Output:
243     -- Gas percentage of supported gases in ppm.
244     """
245
246     resistance = self.__read()
247     ppm = {}
248
249     ppm[self.CO_GAS] = self.__calculate_gas_percentage(
250         resistance / self.ro, self.CO_GAS
251     )
252
253     ppm[self.H2_GAS] = self.__calculate_gas_percentage(
254         resistance / self.ro, self.H2_GAS
255     )
256
257     ppm[self.CH4_GAS] = self.__calculate_gas_percentage(
258         resistance / self.ro, self.CH4_GAS
259     )
260
261     ppm[self.LPG_GAS] = self.__calculate_gas_percentage(
262         resistance / self.ro, self.LPG_GAS
263     )
264
265     ppm[self.PROPANE_GAS] = self.__calculate_gas_percentage(
266         resistance / self.ro, self.PROPANE_GAS
267     )
268
269     ppm[self.ALCOHOL_GAS] = self.__calculate_gas_percentage(
270         resistance / self.ro, self.ALCOHOL_GAS
271     )
272
273     ppm[self.SMOKE_GAS] = self.__calculate_gas_percentage(
274         resistance / self.ro, self.SMOKE_GAS
275     )
276
277 return ppm

```