Group 6

CARBON EMISSION MONITORING SYSTEM

Project Report

Innovation Lab
Autumn 2023

Prof. Srinibas Karmakar

ABSTRACT:

Measuring the concentration of pollutants in remote places and high altitudes is a crucial but challenging task, we must be able to understand and quantify the impact that various combustion and industrial pollutants have on the atmosphere as a whole. It is precisely for this that we develop a multiple-sensor integrated portable airborne module that detects and measures the concentrations of various pollutants in the air.

The initial design included a helium balloon to which the integrated sensor module was to be attached, and its altitude was to be controlled by reducing the helium content in the balloon using a valve. However, due to budget and logistical constraints, the helium balloon idea was dropped. We next settled on a drone-based sensor module which included MQ135 and MQ7 sensors that measure the CO2 and CO concentrations respectively. We also included a BMP280 sensor to obtain the altitude of our module based on the atmospheric pressure. The nRF24L01 transceiver module was used to transmit the data from the airborne module to the receiver on the ground.

The module ran smoothly in the project demo and we obtained reasonable data. The test was conducted within the campus of IIT Kharagpur and we found out that the surface level CO2 concentration was much lesser than the global average concentration, which is an indicator of the good air quality within the campus, which we may perhaps attribute to the abundance of trees within. Over approximately 26 meters of ascent and descent, we found the CO2 concentration to increase with height to levels very close to the global average value of 421 ppm. However, the carbon monoxide levels remained constant at 4 ppm.

We can perhaps obtain more interesting data if we are to test our model in more polluted environments, such as near a factory or near busy traffic. The maximum height of the ascent was also limited due to budget and safety concerns, which limited the data we could collect. This module apart from mapping the variation of concentration of pollutants with altitude can also be used to measure the pollutant concentrations in remote and dangerous locations such as over the tall smoke chimneys in factories or over volcanoes.

INTRODUCTION AND OBJECTIVES:

Carbon emission has become a significant problem in recent years and is responsible for global warming and the resulting climate changes. Hence it only makes sense that we should monitor carbon emissions regularly and act accordingly. Also, according to the research done, carbon emission monitoring gas balloons are a very cost-effective way of monitoring upper atmospheric carbon emission data.

Our project aims to develop an affordable carbon monitoring device that will provide an effective means of continuously monitoring carbon emissions, enabling industries to ensure optimal emission control and environmental compliance. Following the design and making of the model, we aim to plot a concentration vs altitude plot for both CO2 and CO gases. Through this innovation, we aim to offer a cost-effective solution that enhances the monitoring and management of industrial carbon emissions, contributing to sustainability efforts and regulatory adherence.

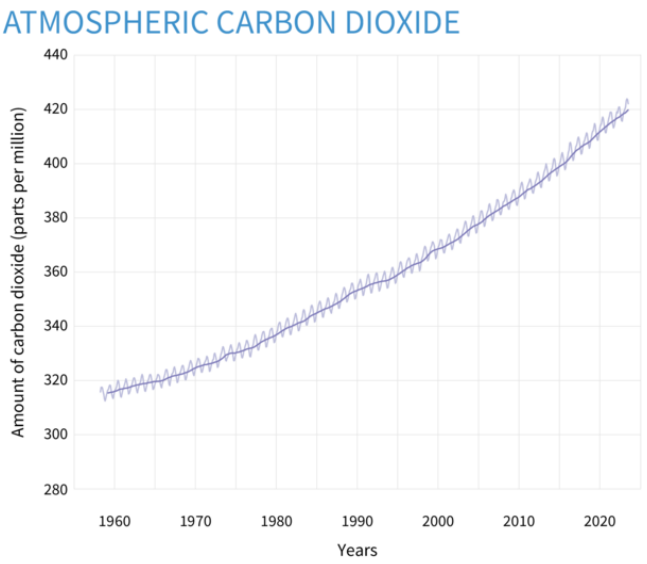

Figure 1. Graph showing variation in concentration of atmospheric CO_2 over the years

The applications of our project include:

- Environmental Monitoring in Different Environments: The carbon emission monitoring system can be used in different environments. This could include urban areas, industrial sites, forests, and more.
- Real-time Monitoring: Balloons/Drones equipped with carbon emission monitoring devices can provide real-time data, enabling rapid response to sudden changes or anomalies in emission levels. Drones can be especially useful during events like chemical spills, fires, or accidents.
- Public Awareness: Visualising carbon emissions through interactive maps and reports can raise public awareness about environmental issues. This can lead to greater public support for emission reduction initiatives and sustainable practices.
- Verification of Emission Reduction Claims: Industries and companies that claim to have reduced their carbon emissions can have their claims verified through carbon monitoring, promoting transparency and accountability.

BACKGROUND AND THEORY:

MQ-Series Sensors

Both MQ7 and MQ135 use a thin film of tin oxide (SnO2) as their sensing element. These are n-type semiconductors i.e., the majority of charge carriers are electrons. The presence of an oxidising or reducing gas changes the resistance of the film by changing the surface charge carrier concentration via altering the amount of oxygen adsorbed on the surface. This change in charge carrier concentration is typically affected by the adsorption of oxygen on the film surface. Due to their opposite effects on charge carrier concentration, a reducing or oxidizing gas will have opposite effects on the resistance of the sensors. Since reducing gases such as CO2 and CO increase the number of electrons on the surface, they lower the resistance in n-type

semiconductors present in these sensors. Thus the voltage output of the sensor increases with increasing concentration of the corresponding reducing gas.

MQ135

MQ-135 Gas Sensor is an air quality sensor for detecting a wide range of gases, including NH3, NOx, alcohol, benzene, smoke, and CO2. In this project, we are using an MQ-135 sensor module to measure CO2 concentration in PPM. The analog data is taken by ADC (A0, A1, A2, etc pins) of Arduino, according to the voltage levels. The circuit diagram for the sensor is given below:

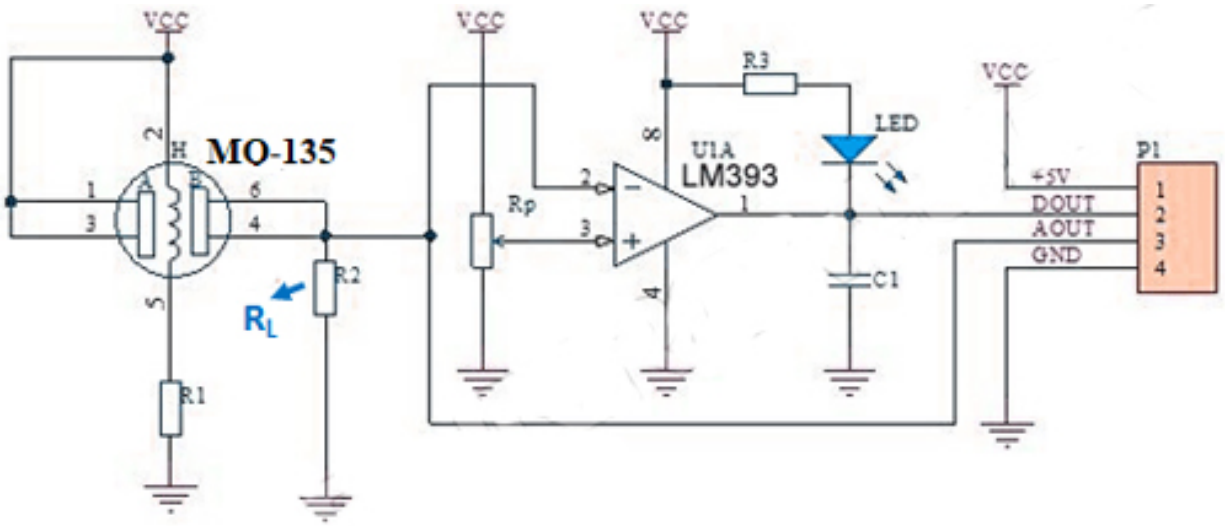

Figure 2. MQ135 sensor circuit diagram

The sensor resistance is in series with the load resistor R_L (or R_2). The value of R_L is indicated on the sensor board as 1K Ohm.

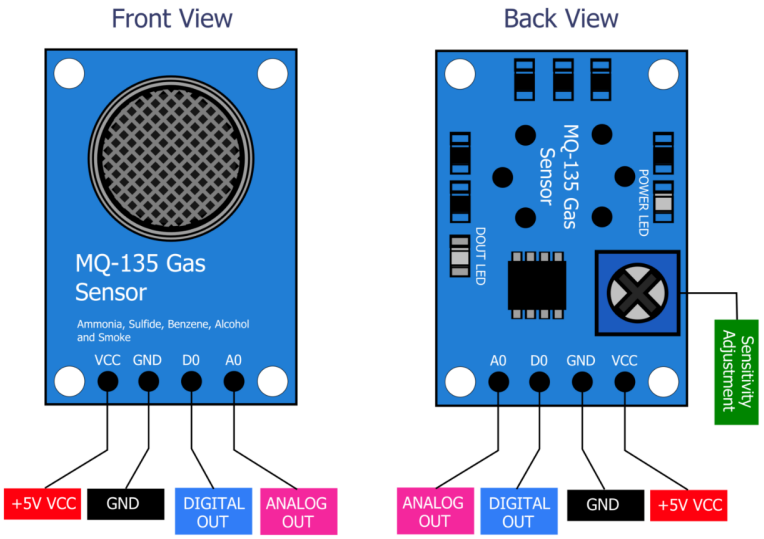

Figure 3. MQ135 pin diagram

Calibration:

The calibration curve for CO_2 is given below:

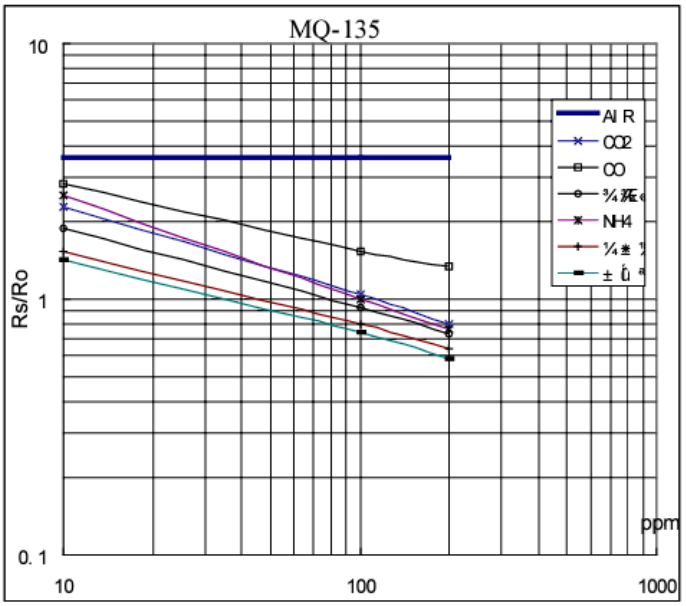

Figure 4. Sensitivity characteristics of the MQ-135 for several gases

The equation of the calibration curve (as per the plot in the datasheet) is given by:

$$ppm = 116.602\ (R_s/R_o)^{-2.769}$$

Ro is the sensor resistance in clean air. The value of Ro can be measured from the code given below. First, the MQ135 library is downloaded to import the necessary functions. In addition to this, the load resistance value is changed in the MQ135.h file present in the library to 1K Ohm.

```
#include "MQ135.h"
void setup (){
Serial.begin (9600);
}
void loop() {
MQ135 gasSensor = MQ135(A0); // Attach sensor to pin A0
float rzero = gasSensor.getRZero();
Serial.println (rzero);
delay(1000);
}
```

The Ro value obtained is input in the MQ135.h file. The clean air concentration of CO_2 is specified to be 390 ppm as per the local environment data.
The following line of code can now be used to obtain the ppm value for CO_2.

```
float ppm = gasSensor.getPPM();
```

MQ7

The MQ-7 carbon monoxide gas sensor is specially designed to be sensitive to carbon monoxide (CO) gas. The circuit and connections are identical to the MQ135 sensor.

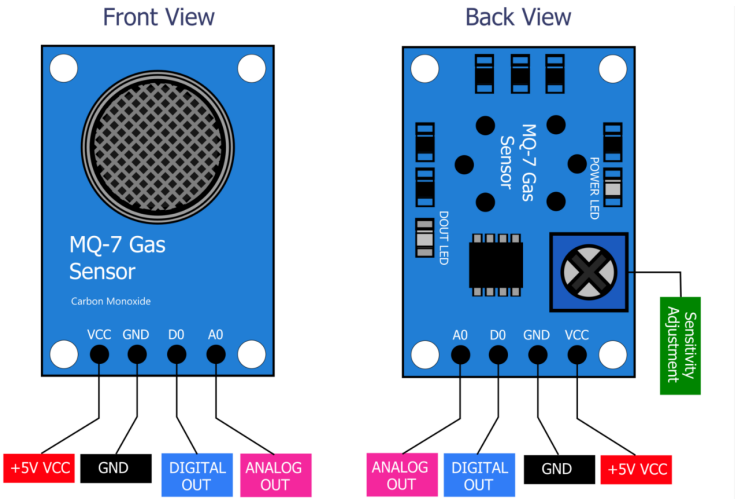

Figure 5. MQ7 pin diagram

Calibration

The sensitivity characteristic curve for CO is given below:

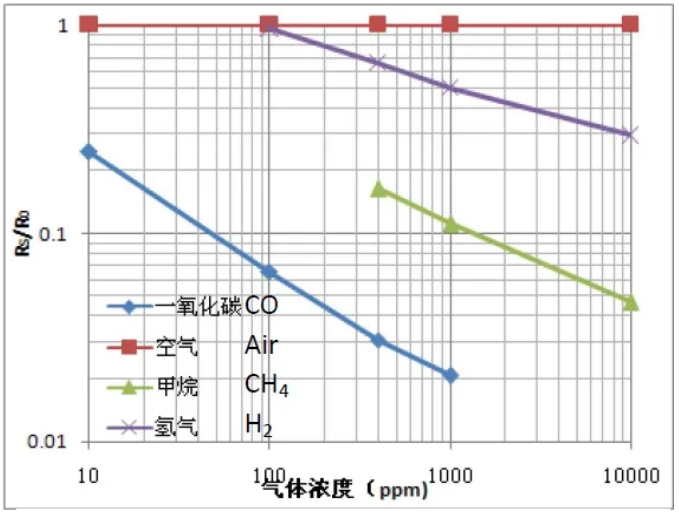

Figure 6. Sensitivity characteristics of the MQ-7 for several gases

The Calibration equation is given by:

$$ppm = 1538.46\ (R_s/R_o)^{-1.709}$$

Ro is the sensor resistance in clean air and can be measured using the code given below. The value of R2 is indicated on the sensor as 1K Ohm.

```
float sensor_volt;
float RS_gas;
float R0;
int R2 = 1000;

void setup() {
 Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  sensor_volt=(float)sensorValue/1023*5.0;
  RS_gas = ((5.0 * R2)/sensor_volt) - R2;
  R0 = RS_gas / 1;
  Serial.print("R0: ");
  Serial.println(R0);
}
```

Once R0 is determined, we can then use the following equation to display the ppm concentration of C0
Rs =(5 - sensor voltage)*R2/sensor voltage
The value of Rs/Ro can be used in the calibration equation to obtain the ppm value.

BMP280 - Pressure sensor:-

BMP280 is a sensor that can measure both barometric Pressure and temperature. This sensor is Cheap and can be used as an altimeter as Pressure changes with altitude. The temperature range and the operating range of this sensor are the same: -40 to +85 °C. That means the temperature of the sensor should be within that range for stable operation. It has both I2C and SPI interfaces. The environmental pressure is subject to many short-term changes, caused e.g. by the slamming of a door or window, or wind blowing into the sensor. To suppress these disturbances in the output data without causing additional interface traffic and processor workload, the BMP280 features an internal IIR filter. It effectively reduces the bandwidth of the output signals.

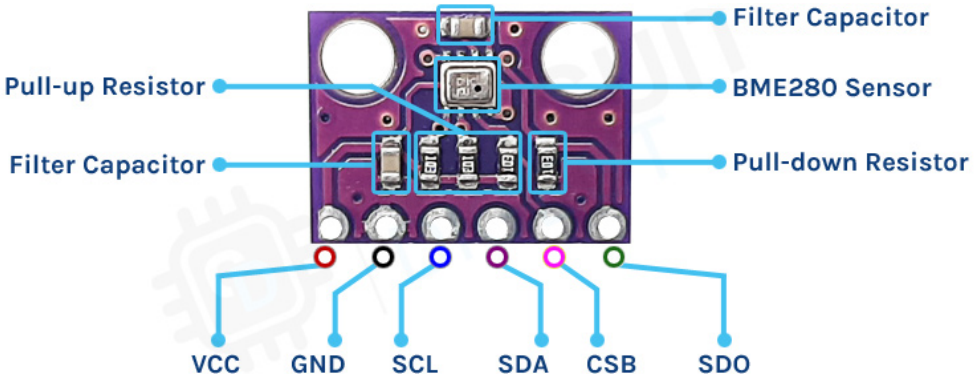

Figure 7. BMP280 sensor components and pinouts

The BMP280 Digital Pressure Sensor module has 6 pins VCC, GND, SCL, SDA, CSB, and SDO. All the pins of this sensor module are digital, except VCC and Ground.

- VCC is the power supply pin of the BMP280 that can be connected to 3.3V or 5V of the supply. Note that the analog output will vary depending on the provided supply voltage.
- Ground is the ground pin of the Sensor and it should be connected to the ground pin of the Arduino.
- SCL stands for Serial Clock the master device pulses this pin at regular intervals to generate a clock signal for communication.
- SDA stands for Serial Data, through this pin data exchange happens between two devices.
- CSB is the chip select pin of the module, if you are communicating with the device with SPI you can use this pin to communicate to select one if multiple devices are connected in the same bus.
- SDO is the Serial Data out pin of the module. An output signal on a device where data is sent out to another SPI device.

SCL, SDA, CSB, and SDO pins are 5V tolerant. As you can see in the above image this board has two filter capacitors, both of which are 100nF capacitors. One capacitor is used to filter the analog power line and another one is used to power the digital power line. Other than that we

have three pullup resistors for SCL, SDA, and SDO and one pulldown Resistor for SDI, with all these parts the BMP280 sensor is made.
In our Project, we are I2C Interface with Arduino. So we only require 4 pins: VCC, Ground, SCL, and SDA.
The Connection is shown below:

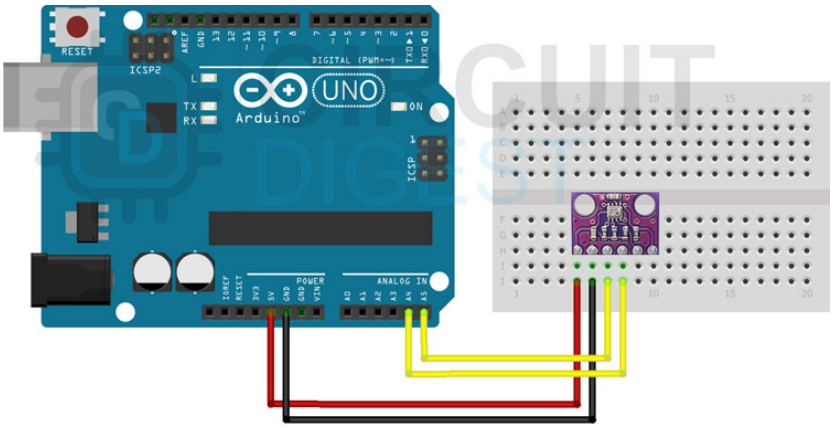

Figure 8. BMP280 connection to Arduino Uno

We have connected the power pins of the Arduino board to the power pins of the BMP280 module and we have connected the SCL and SDA pins of the module to the SCL and SDA pins of the Arduino board(i.e A5 and A4 respectively).

nRF24L01:

The nRF24L01 is a single-chip transceiver(half duplex) with an embedded baseband protocol engine (Enhanced ShockBurst™), designed for ultra-low-power wireless applications. It uses the 2.4 GHz band and it can operate with baud rates from 250 kbps up to 2 Mbps. The module's operating voltage ranges from 1.9 to 3.9V. The nRF24L01 is configured and operated through a Serial Peripheral Interface (SPI). Our model has an RFX2401C chip which includes **PA** (Power Amplifier) and **LNA** (Low-Noise Amplifier). This amplifies the NRF24L01 signal and enables a better transmission range of up to 1000 metres in open space. The module can use 125 different channels which gives a possibility to have a network of 125 independently working modems in one place. Each channel can have up to 6 addresses, or each unit can communicate with up to 6 other units at the same time.

Figure 9. nRF24L01 pin diagram

- GND is the ground pin. It has a square marking to distinguish it from the other pins.

- VCC supplies power to the module. It can range from 1.9 to 3.9 volts. You can connect it to your Arduino's 3.3V output. With the help of an adapter module, we can connect the VCC pin to Arduino's 5V output.
- CE (Chip Enable) is an active-high pin. When enabled, the nRF24L01 will either transmit or receive, depending on the mode.
- CSN (Chip Select Not) is an active-low pin that is typically held HIGH. When this pin goes low, the nRF24L01 begins listening for data on its SPI port and processes it accordingly.
- SCK (Serial Clock) accepts clock pulses from the SPI bus master.
- MOSI (Master Out Slave In) is the SPI input for the nRF24L01.
- MISO (Master In Slave Out) is the SPI output of the nRF24L01.

In the setup section of the transmitter, we need to initialise the radio object and use the radio.openWritingPipe() function. We set the address of the receiver to which we will send data. On the other side, at the receiver, using the radio.setReadingPipe() function we set the same address and in that way, we enable the communication between the two modules. To send multiple values at the same time, we created a struct and stored 8 values(float) in it.

We have the radio.stopListening() function which sets the module as a transmitter, and on the other side, we have the radio.startListening() function which sets the module as a receiver. Using the radio.write() function we will send the message to the receiver. Using the radion.read() function we read and store the data on the receiver end.

NEO-6m GPS Module:

The Global Positioning System (GPS) is a satellite-based navigation system that consists of 24 orbiting satellites, each of which makes two circuits around the Earth every 24 hours. These satellites transmit three bits of information – the satellite's number, its position in space, and the time the information is sent. These signals are picked up by the GPS receiver, which uses this information to calculate the distance between it and the GPS satellites. With signals from three or more satellites, a GPS receiver can triangulate its location on the ground (i.e., longitude and latitude) from the known position of the satellites. With four or more satellites, a GPS receiver can determine a 3D position (i.e., latitude, longitude, and elevation).
The NEO-6M GPS module is a GPS receiver that can locate all locations on Earth as it can track approximately 22 satellites. It consists of a high-performance U-blox 6 positioning engine. Measuring 16 x 12.2 x 2.4 mm, its compact architecture along with its low power consumption makes it a good choice for IoT projects. Overall it is a good cost-effective GPS receiver.

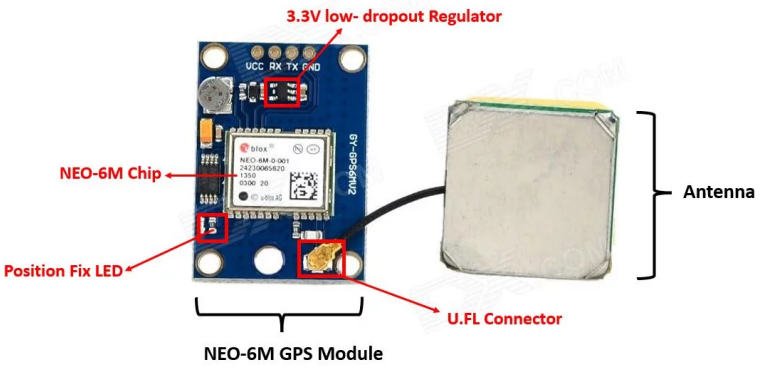

Figure 10. Neo-6M GPS module components

INITIAL MODEL:

The initial model was designed such that a carbon emission monitoring module would be taken up into the air tied to a valved helium balloon, whose ascend and descend can be controlled by a valve that releases the helium from it. For this, we wrote a function in Python that takes the balloon's volume, payload, and the desired altitude as input and gives back the time taken to reach the particular altitude, and velocity at that altitude, to open the valve in time. We wrote this code so as to obtain a ballpark amount of time for which it is reasonable that the valve has not opened and the balloon continues its ascent. So if the valve has not opened even after this estimated time in a scenario where the code for the valve malfunctions we can manually make the valve open by overriding the valve code. The code is as follows:

```
import numpy as np
import scipy
from scipy.integrate import solve_ivp
import math

# Constants
g = 9.81
# rho_helium = 0.1786  # kg/m³
V_ballooni =float(input("enter the volume of helium in m³"))
payload=float(input("enter the payload in kg"))
# Desired altitude
target_height = float(input("enter the target height in metres")) # m
velocityf=[]

r3=3/(4* np.pi)
r=r3**(1/3)
A_balloon = np.pi * (r ** 2)  # m²
rho_0 = 1.225  # kg/m³ (surface air density)
scale_height = 8000  # m
Cd = 0.47  # Drag coefficient (for a spherical balloon)
# You may adjust Cd based on the shape and size of the balloon.

# Differential equation with drag
def balloon_ascent_with_drag(t, y):
    h, v = y

    # Constants
    P0 = 1013.25  # hPa (standard pressure at sea level)
    L = 0.0065  # K/m (standard temperature lapse rate)
    T0 = 298  # K (standard temperature at sea level)
    T=T0-0.00649*h
    M = 0.029  # kg/mol (molar mass of dry air)
    R = 8.314  # J/(mol·K) (universal gas constant)


    # Calculate pressure using the barometric formula
    P = P0 * (1 - (L * h) / T0) ** ((g * M) / (R * L))

    rho_air = rho_0 * np.exp(-h / scale_height)
    rho_helium=rho_air*4/29
    V_balloon = V_ballooni *P0*T/(T0*P)
    m_balloon=V_balloon*rho_helium
    drag_force = 0.5 * Cd * rho_air * A_balloon * v**2
    dh_dt = v
    dv_dt = ((rho_air - rho_helium) * V_balloon * g -
drag_force-payload*g) / m_balloon
```

```
    velocityf.append(v)
    return [dh_dt, dv_dt]
# Initial conditions
initial_state = [0, 0]  # [h(0), v(0)]


# Time span
t_span = (0, 500)  # Adjust the upper limit as needed

# Solve the differential equation with drag
sol = solve_ivp(balloon_ascent_with_drag, t_span, initial_state,
dense_output=True)

# Find the time when the desired height is reached
heights = sol.y[0, :]
times = sol.t
for i, h in enumerate(heights):
    if h >= target_height:
        ascent_time = times[i]
        break
print("Velocity at target height without loss of helium = ",velocityf[i])
print(f"Time to reach {target_height} meters with drag:
{ascent_time:.2f} seconds")
```

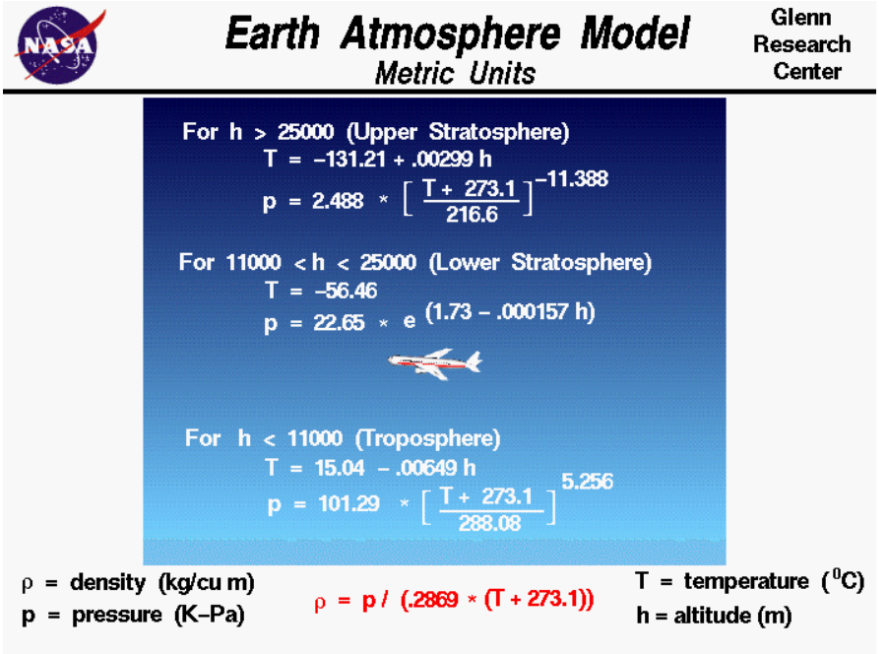

Figure 11. Earth Atmosphere Model (Source: NASA)

For the code variation in atmospheric conditions such as changes in density, temperature, and pressure were computed using the equations from NASA's atmospheric model. For simplification, a slacked balloon model was assumed. In a slacked balloon pressure outside the balloon is equal to the pressure inside the balloon. The coefficient of drag of the balloon was assumed to be that of a sphere,0.47. The value of time and velocity were found by continuous iteration using numerical methods(Initial Value Problem method).

The schematic for the initial model is given below.

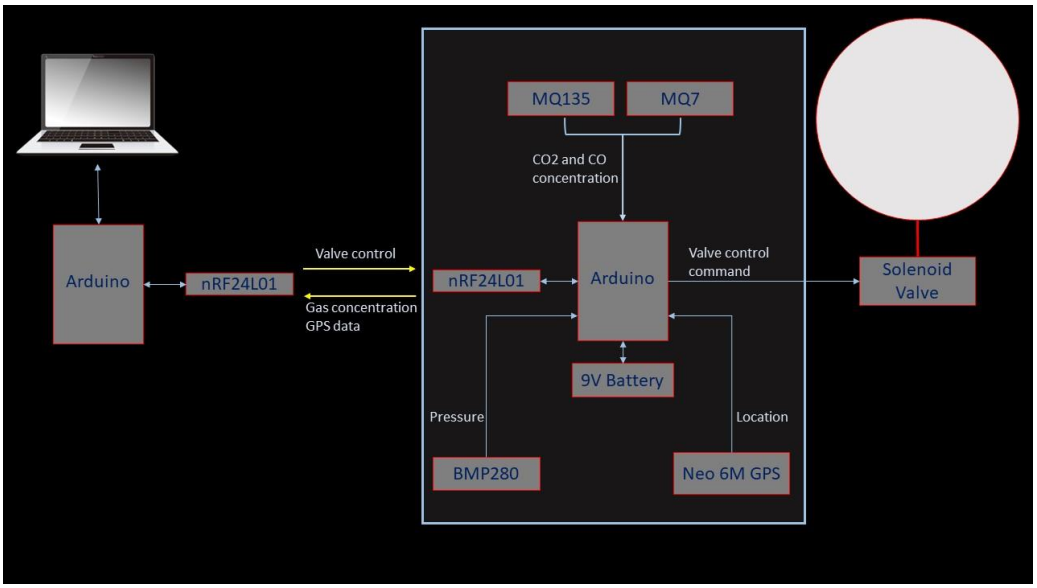

Figure 12. Schematic for initial model

The MQ sensors are used for the detection and measurement of the pollutant concentrations, the nRF24L01 was used for the wireless transmission of data between the airborne module and the ground station (Laptop). The BMP280 for obtaining altitude based on the atmospheric pressure readings, and a solenoid valve to control the amount of helium gas within the balloon. We also included a GPS module so that we can easily locate and retrieve the balloon and module after it lands.
This initial model was estimated to cost over Rs 10000.

FINAL MODEL:

Due to the unavailability of helium, we were forced to look for alternatives for lifting the carbon emission monitoring module to different altitudes. The cheapest and easiest way to lift the module was with a drone. We borrowed a drone and tied the carbon monitoring module to the drone using a long rope. The long rope was used so that the downwind from the drone's propeller wouldn't interact with the MQ sensors' readings.
The schematic of the final model is given below.

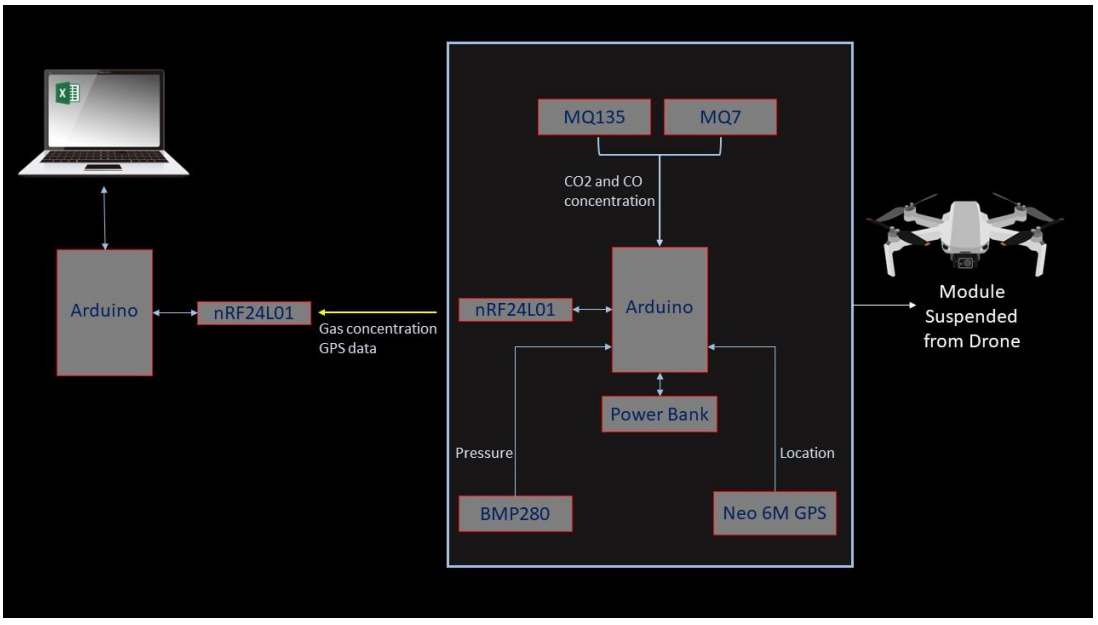

Figure 13. Schematic for the final model

Except for the lack of a solenoid valve, the rest of the components used in this model are the same as those used in the initial model. Also since we have control over the ascent descent and landing of the drone the GPS module became obsolete, but was included nonetheless.
The final cost was estimated to be Rs. 4342.68.

The following is the Arduino code for the transmitter end i.e., for the airborne module:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <Adafruit_BMP280.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
#include "MQ135.h"

int RXPin = 2;
int TXPin = 3;

MQ135 gasSensor = MQ135(A1);
float RS_gas = 0;
float ratio = 0;
float sensorValue = 0;
float sensor_volt = 0;
float R0 = 3400;

int GPSBaud = 9600;

TinyGPSPlus gps;
Adafruit_BMP280 bmp;
RF24 radio(7, 8); // CE, CSN
SoftwareSerial gpsSerial(RXPin, TXPin);

const byte address[6] = "00001";
struct Data {
  float T_bmp = 0;
  float P_bmp = 0;
  float h_bmp = 0;
  float lat = 0;
  float lon = 0;
  float h_gps=0;
  float MQ7_CO_ppm=0;
  float MQ135_CO2_ppm=0;
};

Data X;


void setup() {
  Serial.begin(9600);
  gpsSerial.begin(GPSBaud);
  while ( !Serial ) delay(100);   // wait for native usb
  Serial.println(F("BMP280 test"));
  unsigned status;
  //status = bmp.begin(BMP280_ADDRESS_ALT, BMP280_CHIPID);
  status = bmp.begin(0x76);
  if (!status) {
    Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "
                     "try a different address!"));
    Serial.print("SensorID was: 0x"); Serial.println(bmp.sensorID(),16);
    Serial.print("        ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
    Serial.print("   ID of 0x56-0x58 represents a BMP 280,\n");
    Serial.print("        ID of 0x60 represents a BME 280.\n");
    Serial.print("        ID of 0x61 represents a BME 680.\n");
```

```
    while (1) delay(10);
  }

  /* Default settings from datasheet(BMP280). */
  bmp.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Operating Mode. */
                  Adafruit_BMP280::SAMPLING_X2,     /* Temp. oversampling */
                  Adafruit_BMP280::SAMPLING_X16,    /* Pressure oversampling */
                  Adafruit_BMP280::FILTER_X16,      /* Filtering. */
                  Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */

  radio.begin();
  radio.openWritingPipe(address);
  radio.setPALevel(RF24_PA_MIN);
  radio.stopListening();
}

void loop() {
  while (gpsSerial.available() > 0)
  {
    if (gps.encode(gpsSerial.read()))
    {
      X.lat=gps.location.lat();
      X.lon=gps.location.lng();
      X.h_gps=gps.altitude.meters();
      break;
    }
  }
  // If 5000 milliseconds pass and there are no characters coming in
  // over the software serial port, show a "No GPS detected" error
  if (millis() > 5000 && gps.charsProcessed() < 10)
  {
    Serial.println("No GPS detected");
  }

  sensorValue = analogRead(A2);
  sensor_volt = sensorValue/1023*5.0;
  RS_gas = (5.0-sensor_volt)/sensor_volt;
  ratio = RS_gas/R0;
  float x = 1538.46 * ratio;
  X.MQ7_CO_ppm = pow(x,-1.709);
  X.MQ135_CO2_ppm = gasSensor.getPPM();
  X.T_bmp=bmp.readTemperature();
  X.P_bmp=bmp.readPressure();
  X.h_bmp=bmp.readAltitude(1009.87);
  //Transmitting the required data
  radio.write(&X,sizeof(Data));
  delay(100);
}
```

The following is the code at the receiver end i.e., for the Arduino connected to the laptop:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(7, 8); // CE, CSN

const byte address[6] = "00001";

struct Data
{
  float T_bmp;
  float P_bmp;
```

```
  float h_bmp;
  float lat;
  float lon;
  float h_gps;
  float MQ7_CO_ppm;
  float MQ135_CO2_ppm;
};

Data x;

void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(0, address);
  radio.setPALevel(RF24_PA_MIN);
  radio.startListening();
}


void loop() {
  if (radio.available()) {
  //Receiving the required data
  radio.read(&x,sizeof(Data));
  Serial.print(x.T_bmp);
  Serial.print(",");
  Serial.print(x.P_bmp);
  Serial.print(",");
  Serial.print(x.h_bmp);
  Serial.print(",");
  Serial.print(x.h_gps);
  Serial.print(",");
  Serial.print(x.lat);
  Serial.print(",");
  Serial.print(x.lon);
  Serial.print(",");
  Serial.print(x.MQ135_CO2_ppm);
  Serial.print(",");
  Serial.println(x.MQ7_CO_ppm);
  }
  //Serial Monitor Output Format:
  //T_bmp,P_bmp,h_bmp,h_gps,lat,lon,MQ135_CO2_ppm,MQ7_CO_ppm
}
```

Challenges Faced:

The final cost was estimated to be Rs. 4342.68 and this is considerably cheaper than the initial model. The initial model's expense was more than 10000 as Helium was very expensive. So we had to switch to a Drone based model. The next problem we faced was that nRF24L01 modules operate at 2.4 GHz ISM (Industrial, Scientific, and Medical) band, which is a crowded frequency range. Interference from other wireless devices operating in the same band, such as Wi-Fi routers, Bluetooth devices, and microwaves, could have disrupted communication because we had faced a lot of problems getting the transceiver working properly. Providing a stable and adequate power supply was a Challenge since voltage fluctuations or inadequate power could lead to unstable communication. Most of the components (nRF24L01, NEO-6M GPS, MQ135, MQ7, BMP280) were connected to the transmitter side. Each of these components has its own power requirements. So connecting all these to a single Arduino could lead to power loss which would eventually lead to data loss. This was minimized during testing by increasing the sampling Frequency. Also, 9V batteries could not be used to power the module as the charge drained too fast and had to be replaced with a power bank.

OBSERVATION AND CONCLUSIONS:

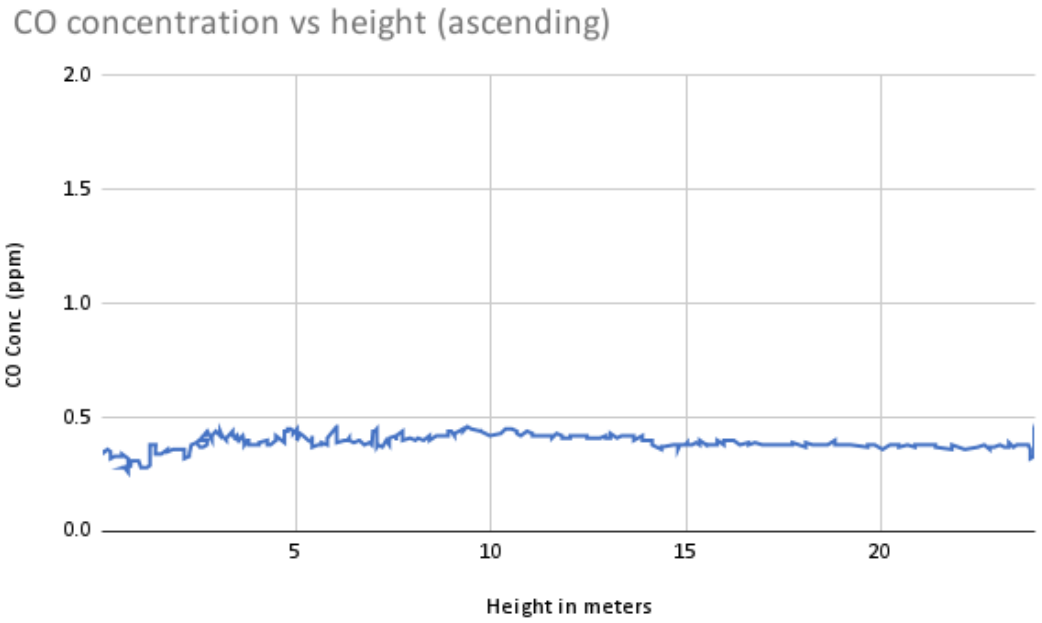

Figure 14. Graph showing the variation in CO concentration with altitude(while ascending)

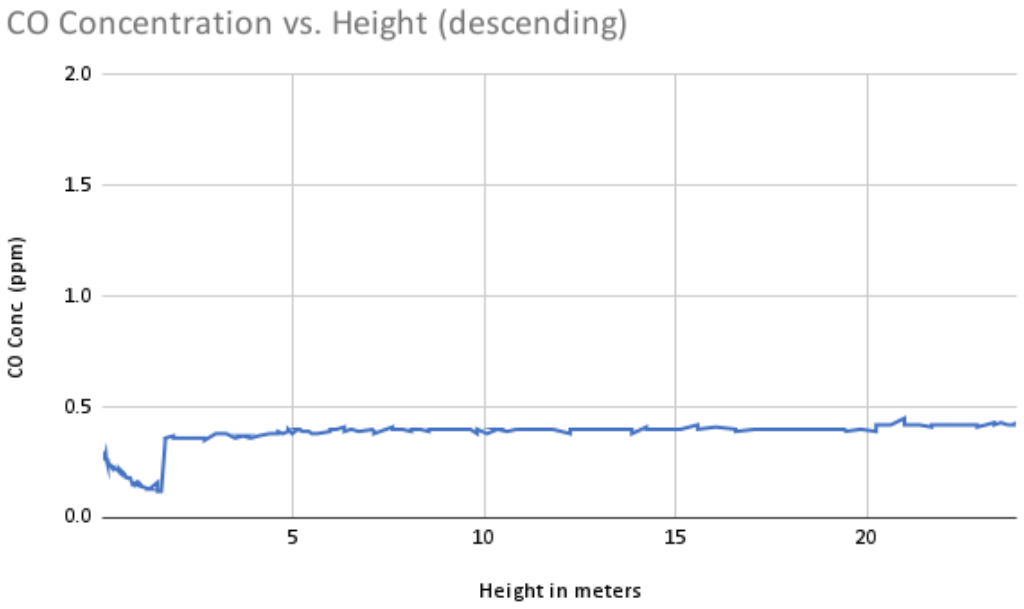

Figure 15. Graph showing the variation in CO concentration with altitude (while descending)

Carbon monoxide concentration was found to be almost constant with altitude at around 0.4 ppm. Carbon monoxide is a long-lived gas in the atmosphere and can travel across regions due to atmospheric circulation patterns. This mixing creates the impression of a relatively constant concentration with altitude. The slight irregularity at the end of the descend plot might have been due to the impact that the module had on the ground when the drone was about to land and need not be observed.

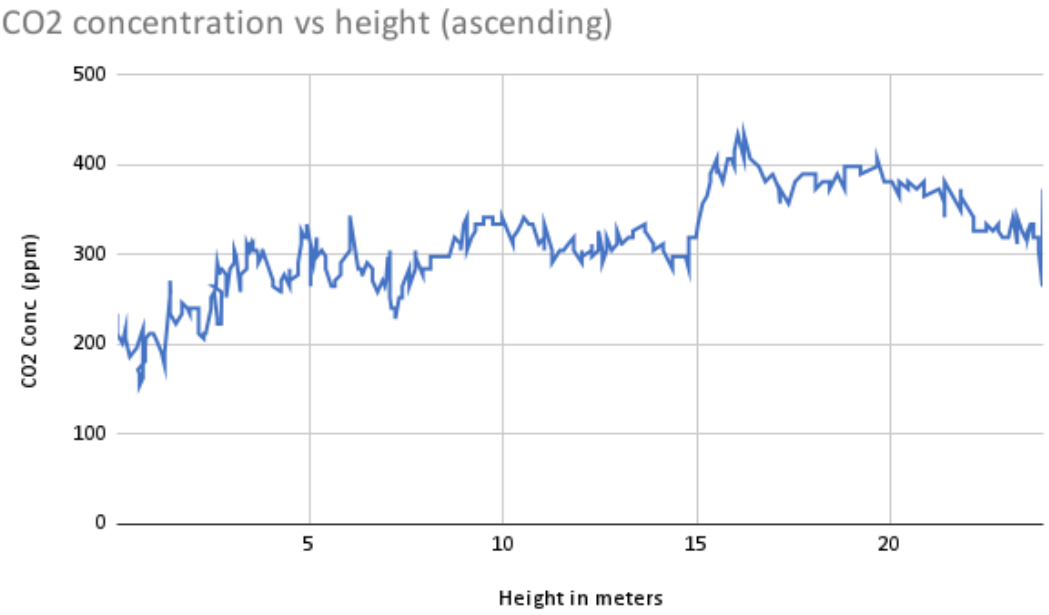

Figure 16.Graph showing variation in CO_2 concentration with altitude(while ascending)

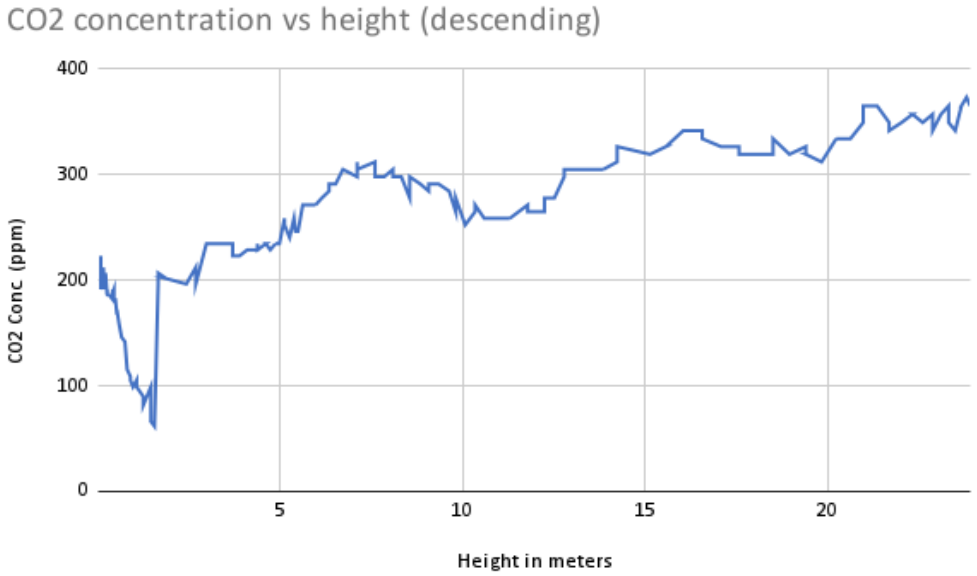

Figure 17. Graph showing variation in CO_2 concentration with altitude(while descending)

Carbon dioxide concentration was found to be increasing with altitude; we can be confident in the reliability of this trend as we obtained mutually consistent results in the ascent and descent phases. Carbon dioxide concentration close to the ground was found to be 300 ppm whereas that at 20 metres was found to be 400 ppm which is very close to the average concentration of Carbon dioxide in the atmosphere(421 ppm). The reason for lower concentrations close to the ground can be attributed to the cleaner atmosphere on the campus and the abundance of trees.

The sensors were also tested by breathing into it and by burning plastics close to it.
Snapshots of performing the above-mentioned tests are given below(Figure 18 and Figure 20). The left-hand side shows one of the members breathing into the MQ135 sensor (Figure 18) and burning plastic close to the MQ sensors (Figure 20). The right-hand side shows the logging of data into the serial monitor which is recorded using the Coolterm software.

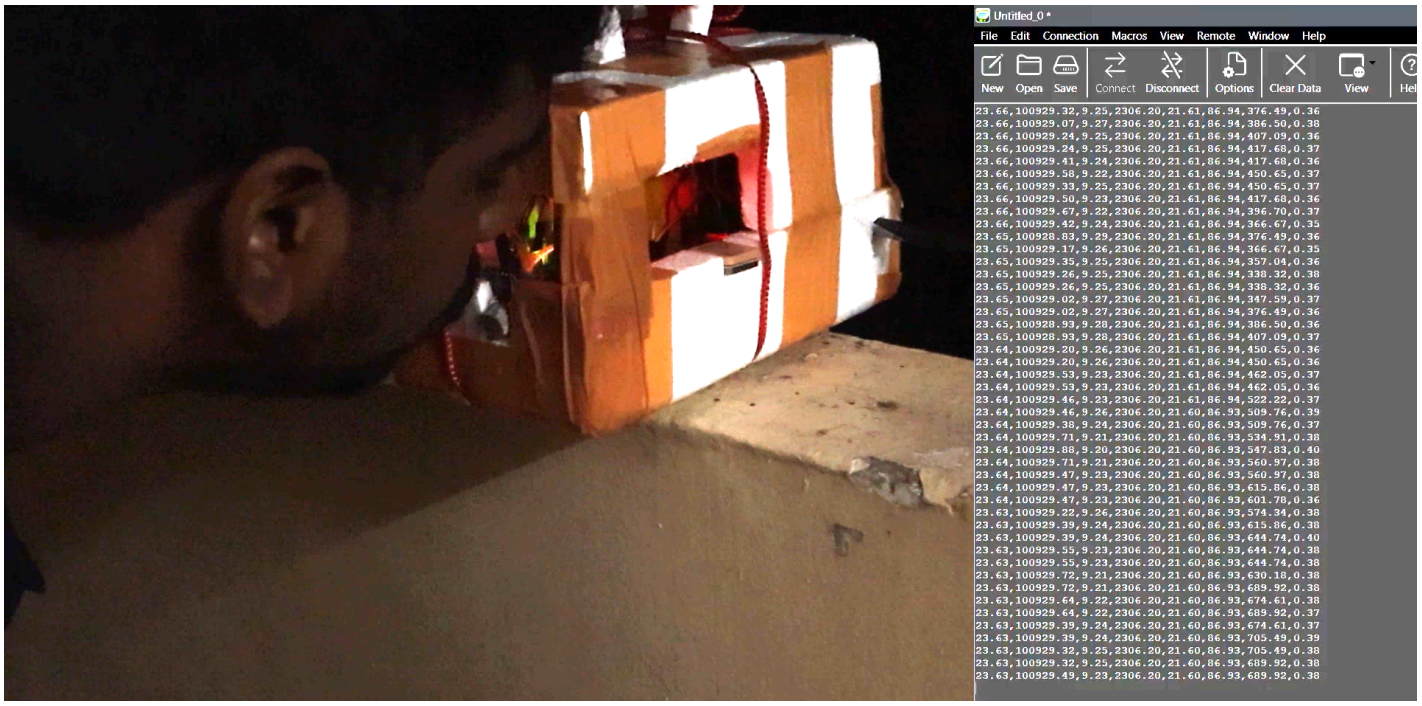

Figure 18. Snapshot of testing the module by breathing into MQ Sensors

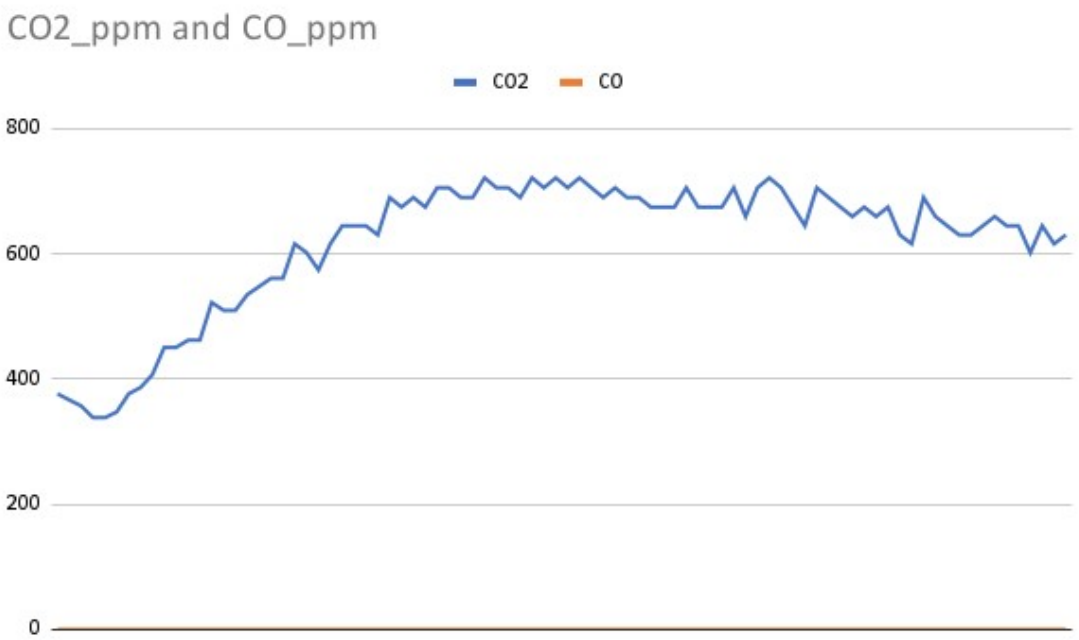

Figure 19. Plot of CO2 and CO concentration while breathing into the MQ sensors

The carbon dioxide concentration increased to around 720 ppm during the process. There were only negligible changes in carbon monoxide concentrations.

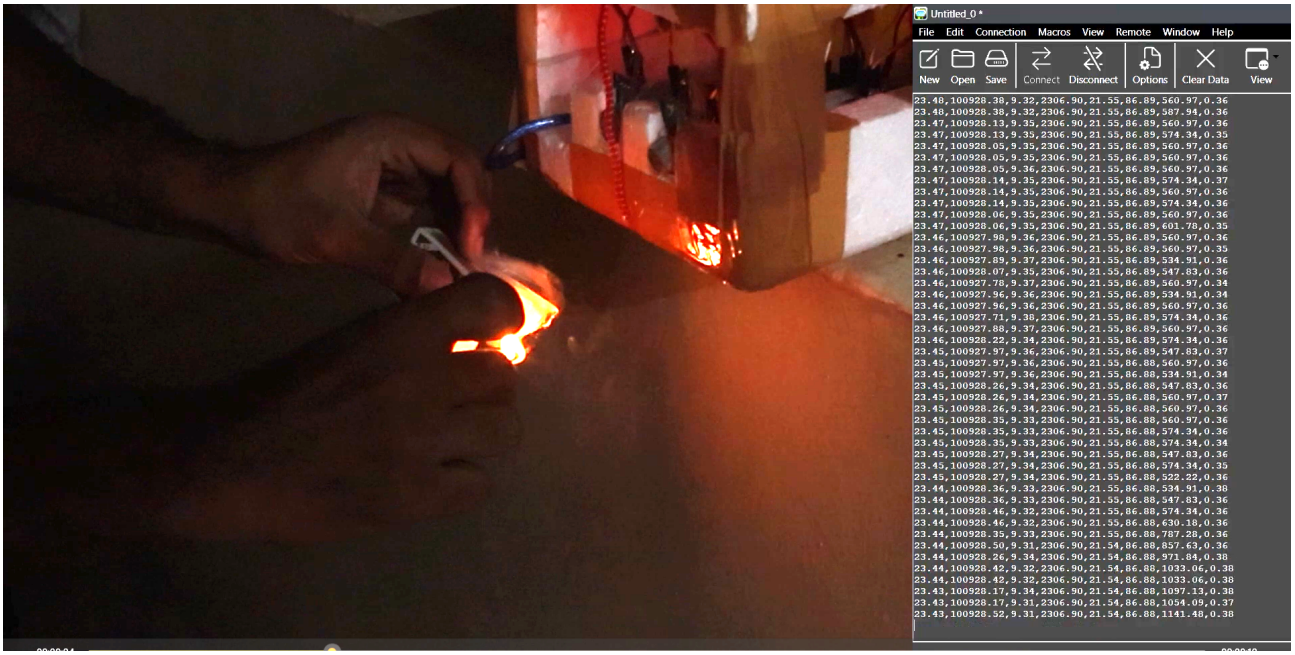

Figure 20.Snapshot of testing the module by burning plastic close to the MQ sensors

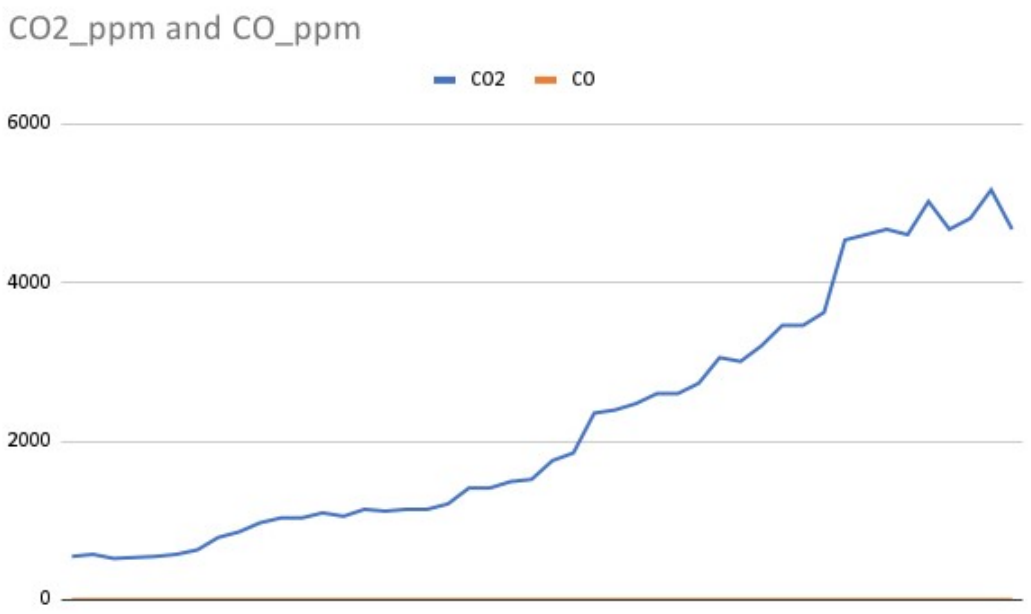

Figure 21.Plot of CO2 and CO concentration while burning plastic close to MQ sensors

The carbon dioxide concentration increased to around 5000 ppm during the process. There were only negligible changes in carbon monoxide concentrations.

DISCUSSION:

The module's successful demonstration within the IIT Kharagpur campus yielded notable findings. The lower-than-average surface-level CO2 concentrations suggested that the campus, rich in greenery, maintained a commendable air quality. As the module ascended, the observed increase in CO2 concentrations indicated a potential correlation between elevation and pollutant levels. Interestingly, carbon monoxide levels remained constant throughout the ascent and descent phases, suggesting stable emissions from local sources within the campus.

The limitations on ascent height impacted the scope of data collection. Nevertheless, the insights gained from the controlled environment of the campus laid a foundation for future investigations in more polluted settings. To unlock the full potential of the module, testing in areas with higher concentrations of pollutants, such as near industrial zones or busy traffic, is crucial for a comprehensive understanding of pollutant distribution patterns.

The carbon emission monitoring module exhibits promising potential for diverse applications. One avenue for future exploration is to enable greater ascent height, thereby expanding the range of data collection. This adjustment would provide a more comprehensive understanding of how pollutant concentrations vary with altitude, offering insights into the vertical distribution of pollutants in different environments. Moreover, the drone-based module opens up opportunities for monitoring pollutants in remote and hazardous locations. Deploying the module over tall smoke chimneys in factories or above volcanoes could yield critical data for environmental monitoring, regulatory compliance, and disaster management. Future iterations of the module may also consider incorporating advanced sensors or additional modules for a more comprehensive analysis of air quality parameters.

In the context of remote sensing, the module could be adapted for autonomous monitoring missions, allowing for the continuous assessment of air quality in areas that are challenging or unsafe for human access. This adaptability could make the module an invaluable tool for environmental researchers, regulatory bodies, and emergency response teams.

In conclusion, the carbon emission monitoring module, despite the shift in its design approach, proved effective in capturing valuable data on pollutant concentrations and altitude within the IIT Kharagpur campus. The findings suggest a potential correlation between air quality and elevation, highlighting the importance of further testing in diverse environments. While the limited ascent height constrained the data collection range, the module's successful demonstration established a solid foundation for future enhancements and applications.

Group Members and Individual Contributions:

Aswin D Menon (21AE10044):

- Assembled the model.
- Performed literature review and did some of the procurement of materials.
- Recorded and edited module testing videos.
- Performed soldering wherever needed.
- Performed the compilation of weekly presentations.
- Worked on the BMP280 pressure sensor, MQ sensors, and GPS Sensors in its coding and connections.

Rahul Sunil (21AE10050):

- Researched the required components and worked on their procurement.
- Worked on calibration of the MQ Sensors.
- Helped with testing of MQ Sensors and Neo-6M GPS Module.
- Assisted in troubleshooting of code.
- Used the CoolTerm software to log data from the serial monitor to a text file and converted it to CSV format during the final project demonstration.

Adithyan U S (21AE10049):

- Researched and worked on the nRF24L01 transceiver, its code, and testing.
- Helped in the procurement of helium.
- Helped in assembling the model.
- Wrote the final Arduino code.
- Worked on testing and troubleshooting of the Arduino code.

Anfal S (21AE10003):

- Worked on the BMP280 Pressure Sensor, its code and testing.
- Worked on the nRF24L01 Transceiver Module Testing.
- Worked on the procurement of Helium.
- Assisted in the assembly of the model.
- Worked on testing and troubleshooting of the Arduino code.

Sagar K P (21AE30031):

- Worked on the Python code for time of ascent estimation.
- Worked on the procurement of Helium.
- Wrote code and operated the CoolTerm software that collected data from the Arduino serial monitor into an Excel file.
- Plotted and analysed the data collected.
- Did troubleshooting and editing of the Arduino code.

Arjun Biju (21AE10007):

- Worked on the Neo 6m geo GPS module code and connections.
- Worked on the code of time of ascent and descent of the helium model-based balloon.
- Helped with the assembly of components.

Allwin Johnjo Prince (21AE10002):

- Worked on the code involving the creation of an atmospheric model, time of ascent, and final velocity calculation of the helium model-based balloon.
- Worked on the code of the Neo 6m geo GPS module.
- Procurement of suitable sensors and helium for the model.
- Data collection and analysis of values obtained from sensors.

Safa N (21AE10033):

- Worked and Researched on MQ-7 and MQ-135 gas sensor models.
- Worked on the codes, testing, and connections of MQ-7 and MQ-135 gas sensors..
- Procurement of the materials required for the model, like helium, required sensors, etc.
- Collected data and analysed values obtained from the sensor.

REFERENCES:

[1] Measuring CO2 Concentration in Air using Arduino and MQ-135 Sensor. (n.d.). https://circuitdigest.com/microcontroller-projects/interfacing-mq135-gas-sensor-with-arduino-to-measure-co2-levels-in-ppm

[2] Pelayo, R., & Pelayo, R. (2023, October 13). How to Use the MQ-7 Carbon Monoxide Sensor | Microcontroller Tutorials. Microcontroller Tutorials | Microcontroller Tutorials and Resources. https://www.teachmemicro.com/use-mq-7-carbon-monoxide-sensor/

[3] Engineers, L. M. (2022, June 26). Interface ublox NEO-6M GPS Module with Arduino. Last Minute Engineers. https://lastminuteengineers.com/neo6m-gps-arduino-tutorial/

[4] Fisher, Alec. "Characterization of MQ-series gas sensor behavior." (2013).

[5] How Does the BMP280 Digital Pressure Sensor Work and how to Interface it with Arduino? (n.d.). https://circuitdigest.com/microcontroller-projects/interfacing-bmp280-sensor-with-arduino

[6]D. (2022, February 27). nRF24L01 – How It Works, Arduino Interface, Circuits, Codes. How to Mechatronics. https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/

[7] https://www.grc.nasa.gov/WWW/K-12/airplane/atmosmet.html

[8]*Pressure Sensor BMP280*. (n.d.). Bosch Sensortec. https://www.bosch-sensortec.com/products/environmental-sensors/pressure-sensors/bmp280/

[9]In-Depth: How nRF24L01 Wireless Module Works & Interface with Arduino. (2018, July 8). Last Minute Engineers. https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/

[10]Simple nRF24L01+ 2.4GHz transceiver demo. (2016, August 28). Arduino Forum. https://forum.arduino.cc/t/simple-nrf24l01-2-4ghz-transceiver-demo/405123

APPENDIX:

[1] Drive link to demonstration videos and pictures of the model: https://drive.google.com/drive/folders/1b_QKvpWWvA8KWqXjem_FMhuLIKy3jdYw?usp=sharing

[2] Link to the excel sheet containing the data and plots: https://docs.google.com/spreadsheets/d/1X-jiQ0da_QNjeM8plZi-cQG62TrI1m9H/edit#gid=1821682068

[3] Link to excel sheet containing data and plots of CO2 and CO concentration during breathing and combustion tests on the module. https://docs.google.com/spreadsheets/d/1gPZ6csv_MRtxPw4Kdqohhf_livHI4_vJbeGhfbBV_H4/edit?usp=sharing