



**BACHELOR'S DEGREE PROGRAMME**

**Electrical Engineering**

---

**Enhancing Accuracy and Efficiency of a  
Digital Nose System with Sensor  
Technology for Early Detection of Changes  
in the Forest**

**SUBMITTED AS A BACHELOR THESIS**

**to obtain the academic degree of**

**Bachelor of Science in Engineering (BSc)**

**by**

**Leo Bilješko**

**Wels, August 2024**

---

Thesis supervisor

Georg Roman Schneider M.Sc.

---

## SWORN DECLARATION

I hereby declare that I prepared this work independently and without help from third parties, that I did not use sources other than the ones referenced and that I have indicated passages taken from those sources.

This thesis was not previously submitted in identical or similar form to any other examination board, nor was it published.

This printed thesis is identical with the electronic version submitted.



.....  
Leo Bilješko

Wels, August 2024

## **ABSTRACT**

The detection of forest health has become significant for maintaining the forest environment, especially now in the time of increasing ecological stressors. The objective of this project is to design an electronic nose (e-nose) using metal-oxide (MOx) gas sensors to be able to distinguish between healthy and stressed trees by detecting unique volatile organic components (VOCs).

The project involved the development and implementation of a gas sensor array, combining multiple MOx sensors, to detect VOCs. Taking advantage of the Arduino microcontroller, data was able to be received from gas sensors, while Python was utilized for data analysis. Data analysis involved machine learning methods, such as Linear Discriminant Analysis (LDA) and Principal Component Analysis (PCA), for classification and dimensionality reduction of the sensor data. Python also came in handy for the creation of graphical user interfaces using libraries like Tkinter and Matplotlib.

The capacity of the e-noses to differentiate between healthy and sick trees was demonstrated in the initial results, where it showed a reasonable level of accuracy. Initially, PCA provided good separation, however, with an increased number of target gases, the separation accuracy deteriorated. The LDA provided a clear separation between two classes, with slight overlaps. A printed circuit board (PCB) was also designed to create a compact and efficient e-nose.

The e-nose was further assessed for different substances that may be present in stressed trees. Although it has shown the good separability of some substances, others overlapped. The great sensitivity of the MOx sensor comes with a cost of selectivity for different gases. Future research will focus on detecting these specific substances contained in the tree's odor using a neural network, enhancing the electronic nose's ability to detect a wider range of compounds.

## KURZFASSUNG

Die Erkennung der Waldgesundheit ist von großer Bedeutung für die Erhaltung der Waldumgebung, insbesondere in Zeiten zunehmender ökologischer Belastungen. Ziel dieses Projekts ist es, eine elektronische Nase (e-nose) mit Metalloxid (MOx) Gassensoren zu entwickeln, die zwischen gesunden und gestreßten Bäumen unterscheiden kann, indem sie einzigartige flüchtige organische Verbindungen (VOCs) erkennt.

Das Projekt umfaßte die Entwicklung und Implementierung eines Gassensorarrays, das mehrere MOx-Sensoren kombiniert, um VOCs zu erkennen. Unter Nutzung des Arduino-Mikrocontrollers konnten Daten von den Gassensoren empfangen werden, während Python für die Datenanalyse verwendet wurde. Die Datenanalyse umfaßte Methoden des maschinellen Lernens wie Lineare Diskriminanzanalyse (LDA) und Hauptkomponentenanalyse (PCA) zur Klassifikation und Dimensionsreduktion der Sensordaten. Python war auch nützlich bei der Erstellung grafischer Benutzeroberflächen mit Bibliotheken wie Tkinter und Matplotlib.

Die Fähigkeit der e-nose, zwischen gesunden und kranken Bäumen zu unterscheiden, wurde in den ersten Ergebnissen demonstriert, die ein angemessenes Maß an Genauigkeit zeigten. Anfangs bot die PCA eine gute Trennung, jedoch verschlechterte sich die Trennungsgenauigkeit bei einer erhöhten Anzahl von Zielgasen. Die LDA ermöglichte eine klare Trennung zwischen zwei Klassen, mit geringfügigen Überlappungen. Eine Leiterplatte (PCB) wurde ebenfalls entworfen, um eine kompakte und effiziente e-nose zu schaffen.

Die e-nose wurde weiter auf verschiedene Substanzen getestet, die in gestreßten Bäumen vorhanden sein könnten. Obwohl sie eine gute Trennbarkeit einiger Substanzen gezeigt hat, überlappten andere. Die hohe Empfindlichkeit des MOx-Sensors geht auf Kosten der Selektivität für verschiedene Gase. Zukünftige Forschung wird sich darauf konzentrieren, diese spezifischen Substanzen im Baumgeruch mit Hilfe eines neuronalen Netzwerks zu erkennen und die Fähigkeit der elektronischen Nase zur Erkennung einer breiteren Palette von Verbindungen zu verbessern.

# CONTENTS

<b>Sworn declaration .....</b>	<b>II</b>
<b>Abstract.....</b>	<b>III</b>
<b>Kurzfassung.....</b>	<b>IV</b>
<b>1 Introduction.....</b>	<b>1</b>
1.1 Background and Motivation .....	1
1.2 Objectives .....	2
1.3 Approach .....	2
1.4 Expected Results .....	3
<b>2 Theory.....</b>	<b>4</b>
2.1 Electronic Nose .....	4
2.2 Metal-oxide (MOx) Gas Sensors.....	4
2.2.1 Resistance Response.....	5
2.2.2 Resistance Measurement .....	7
2.3 Machine Learning Algorithms .....	8
2.3.1 Supervised Learning.....	9
2.3.2 Unsupervised Learning.....	9
2.3.3 Principal Component Analysis.....	9
2.3.4 Linear Discriminant Analysis.....	10
2.4 Microcontrollers .....	11
2.5 Python.....	13
<b>3 Methodology.....</b>	<b>14</b>
3.1 Experimental Setup .....	14
3.2 Circuit Design.....	18
3.3 PCB Design .....	20
3.4 Data Acquisition and Processing.....	22
3.4.1 Data Acquisition With Arduino Due.....	22
3.4.2 Data Analysis Using Python.....	25
3.4.3 User Interface (UI) .....	30
<b>4 Results and Discussion .....</b>	<b>35</b>

4.1	Diginose.....	35
4.1.1	Resistance Measuring.....	35
4.1.2	Linear Discriminant Analysis.....	36
4.2	Smell inspector: .....	38
<b>5</b>	<b>Conclusion .....</b>	<b>40</b>
<b>6</b>	<b>Outlook .....</b>	<b>41</b>
<b>7</b>	<b>List of figures .....</b>	<b>42</b>
<b>8</b>	<b>List of tables .....</b>	<b>44</b>
<b>9</b>	<b>Bibliography .....</b>	<b>45</b>
<b>10</b>	<b>Appendix.....</b>	<b>48</b>
10.1	Appendix I: Arduino Code .....	48

# 1 INTRODUCTION

This chapter gives a concise outline of the thesis. Sections 1.1 and 1.2 describe the major aim and objectives, whereas sections 1.3 and 1.4 deal with the techniques and expected results. This thesis was developed using software tools, such as Visual Studio Code, KiCad, and the Arduino IDE.

## 1.1 BACKGROUND AND MOTIVATION

In recent years, the detection of forest health has become critical for maintaining the forest environment and agricultural richness. Trees, like other living species, produce volatile organic compounds (VOCs) as part of their normal everyday activities. When stress is initiated, such as from diseases or environmental changes like drought and deluge, the concentration of the VOCs that trees produce can be increased significantly. Recent studies of electronic nose technology have unlocked new alternative approaches to monitoring forest health through the detection of odor fingerprints.

Recent research has emphasized the potential of electronic nose technology as a novel method of monitoring forest health by detecting these odor signatures. An electronic nose, sometimes known as an e-nose, uses an array of gas sensors to identify complex odors, much like the human olfactory system [11]. This approach has been successfully used in a variety of applications, including food quality control and medical diagnostics, and is now being investigated for environmental monitoring [24][25].

Conventionally, Gas Chromatography-Mass Spectrometry (GC-MS) has been a standard for the detection of VOCs. Despite the fact that it offers high accuracy and sensitivity, GC-MS requires higher complexity and skills to operate. Due to these challenges, a real-time observation of a tree's health would be complex. Metal-oxide (MOx) sensors offer a great, accessible, and cost-effective alternative due to their wide range of sensitivity to various gases. The change of the gas or the concentration of the gas is detected by the metal-oxide layer's resistance, which makes it suitable for environmental monitoring.

MOx gas sensor technology has been increasing in recent years in many applications, including air quality control, food quality monitoring, and medical applications, featuring its effectiveness in many fields [6].

## **1.2 OBJECTIVES**

The main goal of this study is to develop a stable, cost-effective e-nose for early detection of tree health using MOx gas sensor technologies and machine learning techniques. The objective is to design an e-nose system that can determine sick or healthy tree using MOx sensors and data analysis algorithms. This approach allows real-time monitoring of changes in surrounding environments, providing an additional option to the more complex and more common GC-MS. Furthermore, the project aims to provide stability in the environment by providing a simple solution for early detection of stress present in forests.

## **1.3 APPROACH**

In this work, an electronic nose system was created specifically for monitoring the health of a tree utilizing MOx sensors. A system was created to collect and analyze the scents produced by stressed and unstressed environments. This requires collecting volatile organic compounds (VOCs) generated by trees and then analyzing these chemicals to identify stress patterns.

Machine learning techniques such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) have been used to identify various odors from trees and generate a fingerprint for their identification. PCA decreases the complexity of sensor data while retaining the most relevant information, whereas LDA improves classification, making it easier to distinguish between healthy and stressed trees.

The system also includes a creation of a custom PCB to incorporate all gas sensors into an array, therefore optimizing their performance and stability. Additionally, a graphical user interface (GUI) has been created to enable real-time monitoring and visualization of any changes in gas around the sensors, providing a user with real-time feedback [12].

#### **1.4 EXPECTED RESULTS**

The purpose of this project is to create and successfully apply an electronic nose system that combines gas sensor technology with machine learning methods. This technology is expected to produce a reliable, cost-effective early detection tool for tree health that can be monitored in real time. This method, which provides a reliable alternative to the more common and expensive GC-MS, has the potential for wide usage in environmental and agricultural applications.

Finally, the implementation of this approach has the possibility to significantly improve forest health management by allowing for early detection and reaction to diseases. This approach can help alleviate the consequences of disease, pests, and environmental change, hence enhancing the sustainability and resilience of forest ecosystems. Furthermore, the knowledge gathered from this technology could be applied to other agricultural techniques, assisting larger efforts in environmental conservation and sustainable agriculture.

## 2 THEORY

In this chapter, the theoretical base and background which are needed for the overall thesis implementation are given. It contemplates the electronic nose with a focus on gas sensors, machine learning, microcontrollers, and Python.

### 2.1 ELECTRONIC NOSE

An electronic nose is a sensor device designed to detect various odors. It aims to emulate the human olfactory system utilizing an array of gas sensors, sensitive to a wide range of gases [13]. Data acquisition and pattern recognition algorithms are also important components of an electronic nose design.

Electronic noses work on the assumption that different chemicals produce unique volatile compounds that may be detected and recognized by sensors. A basic workflow for detecting scents with the electronic nose is as follows:

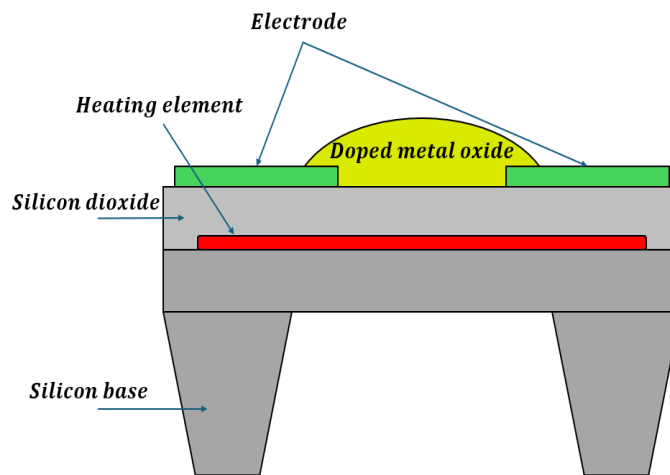
1. Sample collection: The odor's volatile compounds are delivered to the sensor array.
2. Detection: An array of sensors reacts to changes in the medium.
3. Data processing: Raw sensor array data is normalized, filtered, and feature identified.
4. Pattern recognition algorithms: The generated data is then evaluated using pattern recognition methods such as principal component analysis, artificial neural network, or linear discriminant analysis to provide a fingerprint that can be identified.

Metal-oxide semiconductor gas sensors, or shorter MO<sub>x</sub> sensors, are the most popular type of gas sensor, but other options include photoionization sensors, electrochemical sensors, and infrared sensors [14].

### 2.2 METAL-OXIDE (MOX) GAS SENSORS

MO<sub>x</sub> sensors, which are resistive sensors made from metal-oxide semiconductors, have been accepted as a promising possibility to develop low-cost, highly efficient sensors with fast response and recovery times. Due to their simple mechanism for measuring resistance and their high sensitivity to different gases, these devices are often used to detect volatile compounds (VOCs) at ppm and sub-ppm levels. These sensors have applications in the

agriculture and forestry industries to detect plant infections caused by fungus, bacteria, and viruses, as well as damage caused by insects or mechanical means. The schematic representation of a MOx sensor is given in Fig. 1. The schematics show the main components of the sensor, including the metal-oxide sensing layer, the substrate with integrated electrodes, and the heating element, which helps to maintain stable operating conditions [8].



*Fig. 1: Schematic representation of MOx sensor [24]*

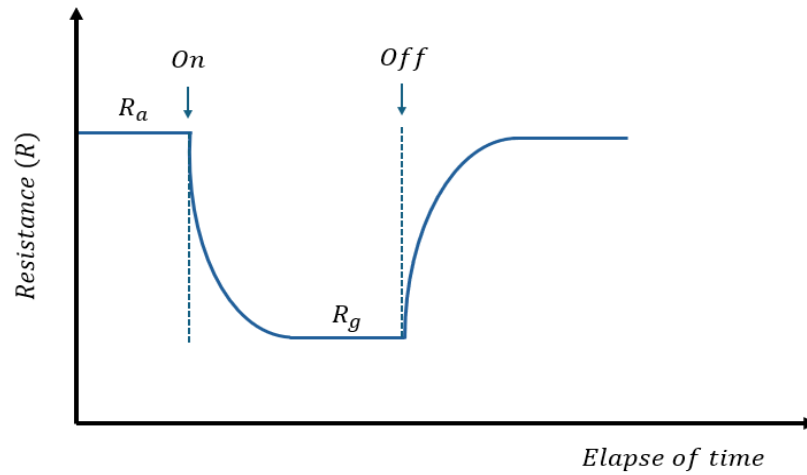
The working principle of the MOx device is based on the changes in the electrical resistance of a metal oxide semiconductor when it encounters gases at elevated temperatures (150-500 °C) [9]. The sensing material is placed on a substrate with integrated electrodes for measuring electrical resistance and a vapor resistor for heating the sensing material [9]. The MOx sensors are susceptible to humidity, so the addition of a heater allows it to operate at controlled conditions and temperatures [8].

In the case of n-type semiconductors, resistance decreases with exposure to reducing gases in the air ( $H_2$ , CO, NO, alcohols, propane, etc.), while it increases with exposure to oxidative gases (NO, ozone,  $N_2O$ , etc.).

### **2.2.1 RESISTANCE RESPONSE**

The behavior of the sensor's resistance when switching between base value and gas exposure is shown in Fig. 2. During non-exposure to a reducing gas, resistance remains

at its base level. During the on-period, gas is detected by a sensor, and its resistance reduces to a stationary value ( $R_b$ ), while it goes back to its base value after the gas exposure ends.



*Fig. 2: Response of a sensor to switching on and off a reducing gas*

Empirically, gas response is defined as the ratio  $R_g/R_a$  (normalized resistance). The characteristic plot shows multiple important aspects regarding the gas sensor's performance. The sensitivity of a sensor is defined by the steepness of the curve, where a steeper curve means higher sensitivity [5]. It also shows its selectivity and its ability to distinguish between different gases. The typical response of the gas sensor is a decrease of  $R_g/R_a$  in reducing gas, as illustrated in Fig. 3.

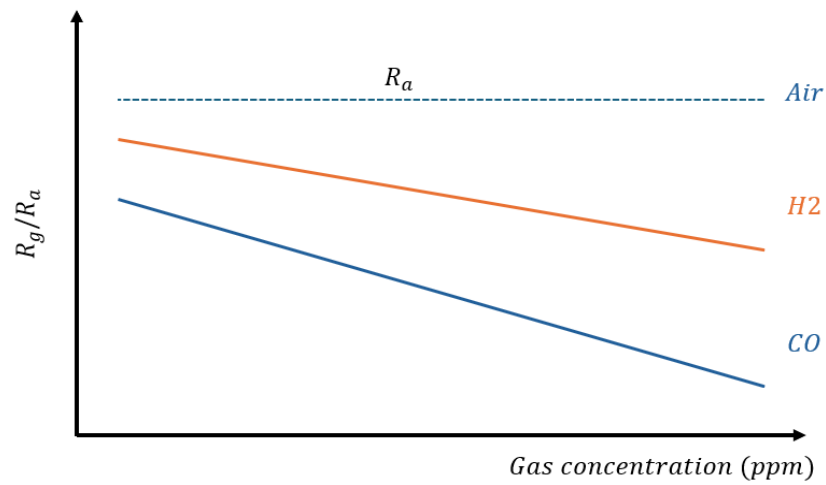


Fig. 3: Linear correlation between resistance and gas concentration

### 2.2.2 RESISTANCE MEASUREMENT

As previously stated, gas sensors' responses are measured as changes in resistance. The measurement of the sensitive layer resistance ( $R_S$ ) is commonly measured using a voltage divider principle with a load resistor ( $R_L$ ). This approach is favored because it is simple and effective in converting resistance changes into quantifiable voltage signals.

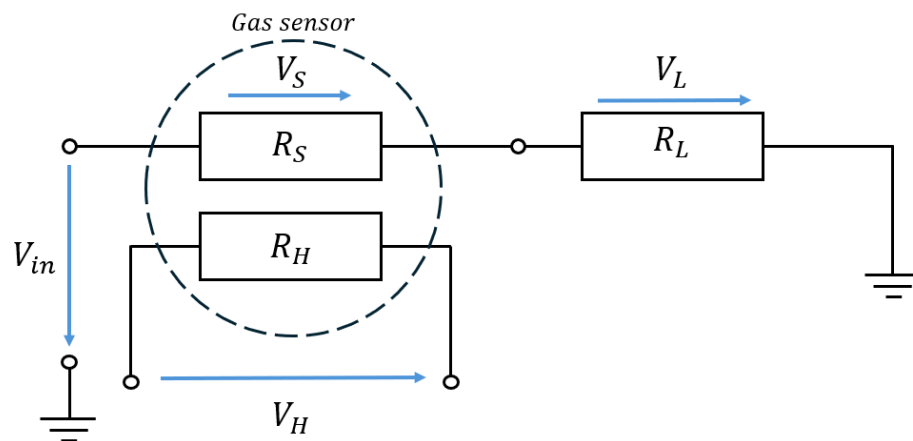


Fig. 4: Equivalent circuit of a gas sensor

A simplified equivalent circuit of a gas sensor is represented in Fig. 4. A circuit is composed of heater resistance  $R_H$ , sensitive layer resistance  $R_S$ , and load resistance  $R_L$ .

A voltage divider is applied to two serially linked resistors, in this example, the sensitive layer resistance  $R_S$  and the load resistor  $R_L$ . When a constant voltage is delivered across resistances, a voltage drop across the load resistor can be utilized to determine the resistance of the sensitive layer [14].

The voltage divider formula is provided as follows:

$$V_L = V_{in} \left( \frac{R_L}{R_L + R_S} \right) \quad (1.1)$$

In this configuration,  $V_L$  is the measurable output voltage that changes when there is a change in  $R_S$ . As the concentration of gas changes, so does  $R_S$  which in return influences  $V_L$ .

By rearranging the formula, the obtained  $R_S$  is represented as:

$$R_S = R_L \left( \frac{V_{in}}{V_L} - 1 \right) \quad (1.2)$$

By measuring  $R_L$  and knowing  $V_{in}$  and  $R_L$ , the resistance  $R_S$  can be estimated appropriately.

The selection of the load resistor  $R_L$  is critical for the gas sensor's performance. It must be selected such that the load voltage  $V_L$  is within a measurable range for the expected changes in  $R_S$ . A value of  $R_L$  also influences the sensitivity of the voltage that is being monitored. A well-chosen load resistor can help detect minor changes in gas concentration, increasing the overall sensitivity of the gas sensor system.

### 2.3 MACHINE LEARNING ALGORITHMS

Machine learning can be described as a set of algorithms and statistical methods that can automatically recognize patterns in data and then use the unknown patterns to accurately predict upcoming data or perform other kinds of decision-making under uncertainty [15].

Usually, machine learning is categorized into two main types: predictive or supervised learning and descriptive or unsupervised learning [2].

### **2.3.1 SUPERVISED LEARNING**

In a supervised approach, the goal is to learn mapping from inputs to outputs so that it can predict the output of new, unseen inputs [16]. An algorithm is provided with a labeled set of input-output pairs, known as the training set [17]. An algorithm utilizes a training set to discover and learn a connection between the inputs and the outputs, giving it the ability to predict new data [2].

Supervised learning has two objectives to accomplish:

- Classification: The output variable is a class (e.g., a healthy or unhealthy tree).
- Regression: To predict the future value of the output variable [18].

### **2.3.2 UNSUPERVISED LEARNING**

In unsupervised learning, the goal is to find hidden patterns in the data provided. It is less defined than supervised learning since it is not given what kinds of patterns to look for and what the desired output is for each input. The advantage that unsupervised learning provides over supervised learning is that it does not require human intervention to manually label the data, which speeds up the data processing [2].

Unsupervised learning goals to achieve:

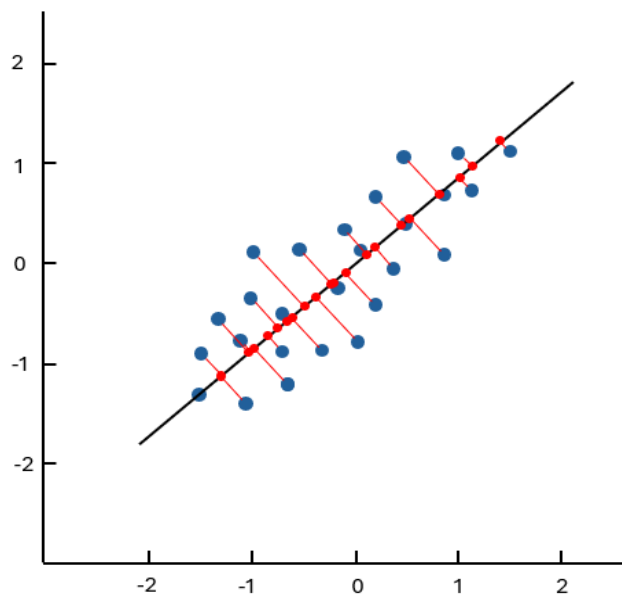
- Clustering: grouping the data that has similar features.
- Dimensionality reduction: reducing the number of input features while keeping as much information as possible [19].

### **2.3.3 PRINCIPAL COMPONENT ANALYSIS**

One example of unsupervised learning is principal component analysis (PCA) [20]. PCA is commonly used as a method to reduce the dimensionality of large data sets [3]. The tradeoff for the reduction is accuracy, but the most significant data features are still retained. Smaller data sets are easier to analyze and visualize, which makes analyzing data points much faster [21].

PCA generates new variables by constructing linear combinations or mixtures of the initial variables, often known as principal components [2]. The concept behind this is that PCA attempts to compress as much information as possible in the first principal component (PC1), then the maximum remaining information in the second (PC2), and so on. The number of principal components depends on the number of input features [3].

The construction of the first principal component accounts for the largest possible variance in the data set. As shown below in Fig. 5, it is the line that goes through the origin, and it is the line in which the projection of the points (red dots) is the most spread out. In mathematical terms, it is the line that maximizes the variance (the average of the squared distance from the projected points to the origin) [3].



*Fig. 5: Construction of the principal component*

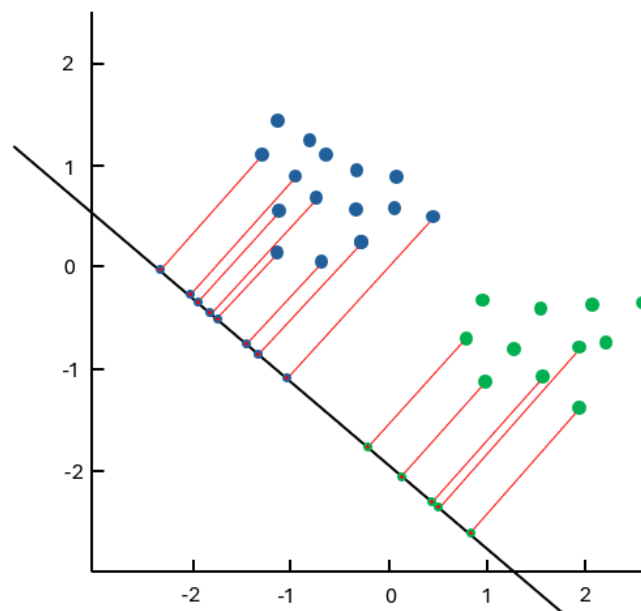
A second principal component is calculated in the same way; however, it must be orthogonal to the first component, and it must account for the next highest variance.

#### **2.3.4 LINEAR DISCRIMINANT ANALYSIS**

Linear Discriminant Analysis (LDA) is a supervised technique for dimensionality reduction that seeks to identify the linear combination of features that best distinguishes

the classes in a data set. The intent is to project data into lower-dimensional space while keeping the information that is most relevant for class discrimination [2].

LDA works by finding a new axis that maximizes the distance between the mean values of various classes while minimizing the spread of points within each class [10]. The LDA produces a set of linear combinations of the original features, comparable to PCA. However, with LDA, these linear discriminants are aiming to separate classes as far apart as possible, which is shown in Fig. 6.



*Fig. 6: Class separability in LDA*

## 2.4 MICROCONTROLLERS

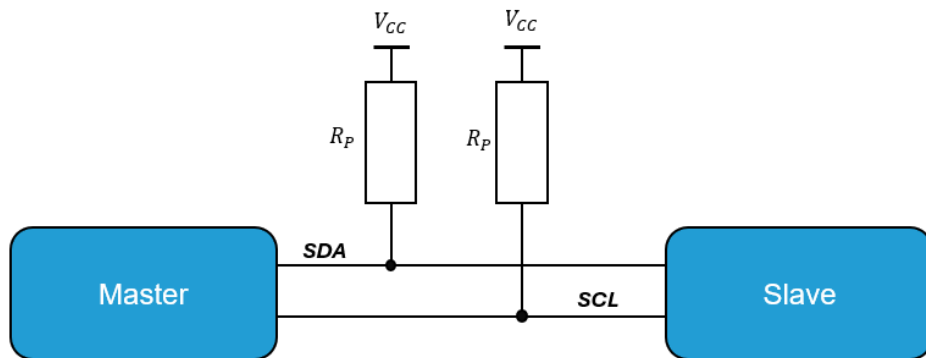
The electronic nose performs using microcontrollers. They are in charge of reading and processing data as well as communicating with other system components. The Arduino Due is one example of a microcontroller, and it was chosen for this project due to its simplicity, vast library choice, and the support Arduino has for its products. Arduino also includes an extensive range of sensors, among which are gas sensors.

A microcontroller is an integrated circuit designed to perform a certain function within embedded system. It consists of a central processing unit (CPU), memory, and input/output interfaces. Microcontrollers offer a wide range of applications in industry, homes, robotics, automation, etc.

Analog and digital pins allow gas sensor to be connected to the Arduino. The analog gas sensors provide a voltage output proportional to the concentration of the target gas and is converted to a digital value through analog-to-digital converter (ADC).

To communicate with the microcontroller, digital sensors use communication protocols such as Inter-Integrated Circuit (I2C) or Serial Peripheral Interface (SPI) [22]. In this project, the I2C protocol was used for digital sensor communication.

I2C communication provides a reliable communication between master (Arduino) and slaves (gas sensors). It allows multiple sensors to be connected to a single master with two wire connection: serial data (SDA) and serial clock (SCL). Each sensor on the bus has a specific address, which allows the microcontroller to communicate with each sensor without confusion. The representation of the I2C protocol is shown in Fig. 7.



*Fig. 7: Inter-Integrated Circuit communication protocol*

For the stable work, I2C communication lines SDA and SCL has to be connected to a pull up resistors, usually of a value 1-10k $\Omega$ .

## 2.5 PYTHON

Python is a high-level, object-oriented programming language known for its versatility and ease of use [4]. It is a popular language for many purposes, particularly scientific research, and data processing. Python was chosen as a tool in the electronic nose project to simplify data processing, machine learning algorithm implementation, and the development of a user interface for interfacing with the system.

A choice of python for this project was due to its wide range of libraries and a simple and easy adaptation to the environment. Libraries like NumPy and Pandas provided a large set of tools for multi-dimensional array data processing and data manipulation; Scikit-learn provided tools for machine learning algorithms such as classification and dimensionality reduction; Tkinter and Matplotlib for user interface and visualization of the data results. Additionally, Serial library which enabled a communication between Arduino and PC, was implemented.

Python's straightforward design, wide range of libraries and data analysis abilities make it a desirable choice for the analysis of the data sets provided by electronic nose.

## **3 METHODOLOGY**

This chapter focuses on the setups used for tree and substance measurements and how they differ. Furthermore, the most significant aspects of circuit and PCB design are highlighted. Finally, data collecting and analysis using Arduino and Python are described, including the user interface.

### **3.1 EXPERIMENTAL SETUP**

For the testing purposes, twelve pine trees were planted and split into three groups of four. Trees were placed in a Mars Hydro grow box to maintain stable conditions. This setup can be seen in Fig. 8. During the starting stages, all trees were treated in the same way and in ideal conditions. Every few weeks, the conditions were changed. The first group of trees remained in ideal conditions, while the other two groups were subjected to stress. One group of trees was depleted of water and remained in dry environments at about 29 °C. The other group was placed in a large drip tray with a large amount of water. An objective of these harsh conditions was to initiate stress in the trees and therefore make them produce more VOCs related to sick trees. Temperature, humidity, and CO<sub>2</sub> level were monitored during their growth to ensure a controlled environment.



*Fig. 8: Setup of trees in grow box*

Trees were also split into four groups for the specific time of the day. These groups were early morning, late morning, early afternoon, and late afternoon, to avoid any chance of daytime affecting data. This way, each tree was tested at approximately the same time of the day.

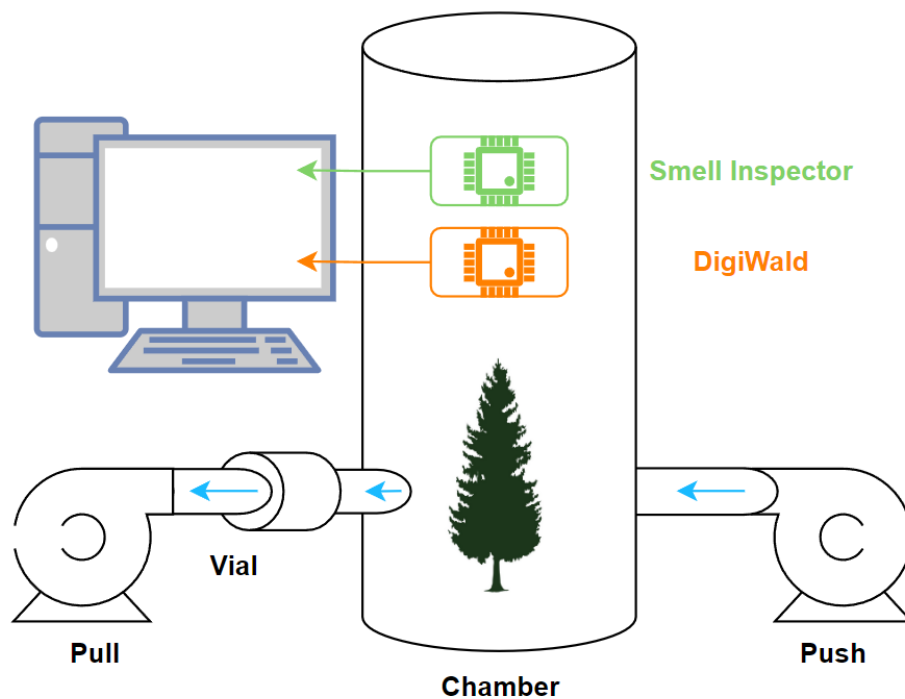
An experimental setup involved a measurement setup for a tree measuring and a substance measuring (see Fig. 9). On the diagram, one can see two electronic noses: Smell Inspector<sup>1</sup> and Digi-nose. A Smell Inspector is an electronic nose built by the company

---

<sup>1</sup> *Smell Inspector* is an electronic nose developed by Smart Nanotubes, used for olfactory measurement applications. More information can be found on their website <https://smart-nanotubes.com/produkt/smell-inspector-developer-kit/>

Smart Nanotubes<sup>2</sup> and serves as a reference to a Digi-nose to make sure that the results from Digi-nose are correct.

A measurement setup involved push and pull pumps for air circulation. For a controlled environment, test trees were placed inside the chamber with a volume of approximately 176 liters. On the output of the chamber, a vial is placed. Its purpose is to gather VOCs that can be later analyzed with the GC-MS method. For measurement, two electronic noses were placed inside the chamber, and their data was forwarded to the PC.



*Fig. 9: Measurement setup for tree measurements*

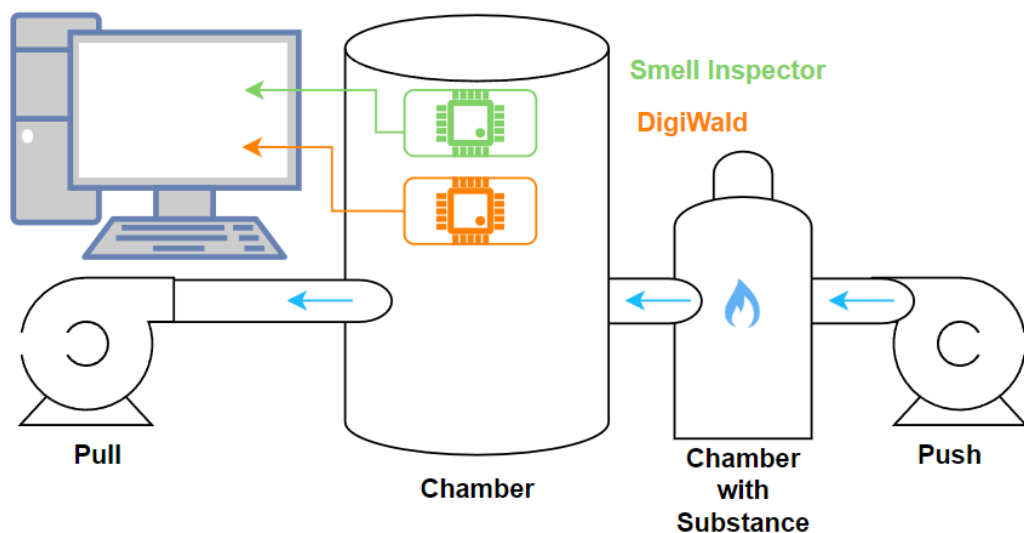
A measurement was divided in two parts. The first segment lasted around 15 minutes and was intended to assess pure air for the noses to establish a baseline. For this time, the chamber remained empty, with no trees inside. Following the initial clean air phase, a tree was placed within a chamber and sealed with no holes to keep the tree odor inside. This stage lasted around 40 minutes, long enough for odor to accumulate in the chamber

<sup>2</sup> *Smart Nanotubes* is a technology company specializing in developing electronic nose sensors and related technologies. Visit their official <https://smart-nanotubes.com/> for more details.

Temperature and humidity were also measured to see if they had an impact on the data. Because gas sensors rely on temperature and humidity, this information can be utilized to compensate for data errors and improve its reliability.

To measure substances, a slightly different configuration is used, but the fundamental principles remain. A vial is removed, and another chamber is inserted between the main four-liter chamber and the push pump. A target substance is placed within this chamber, and the contaminated air is propelled into another chamber containing electronic noses using a push pump. The initial baseline measurement lasted 15 minutes, while the substance measurement lasted about 10 minutes, which is significantly shorter than the tree measurement because the concentration of substance is higher, so gas sensors are more sensitive and longer measurements are unnecessary.

An objective of these measurements is to create a dataset of different substances for machine learning so that certain substances could be identified inside a tree's odor, such as in the GC-MS method. Some of these substances are d-limonene, alpha-pinene, 3-carene, 2-methyl, ethanol, and verbenol. These substances are not only produced by a tree but also from the insects that infest them, like bark beetles. This way it is possible to detect insect-infected trees also. A setup in Fig. 10 was used for substance measurement.



*Fig. 10: Measurement setup for substance measurements*

### 3.2 CIRCUIT DESIGN

Creating a circuit required the use of numerous components, including gas sensors, temperature and humidity sensors, multiplexors, and others. A circuit included a total of eighteen distinct gas sensors. The gas sensors utilized are indicated in Table 1. Sensors were supplied in sets of four or pairs for greater stability of the results.

*Table 1: Type of sensors*

Sensor	Sensitive to	Quantity
<i>MQ-3</i>	CO, Alcohol, Methane	4
<i>MQ-135</i>	CO, CO <sub>2</sub> , Alcohol, Methane	4
<i>UST GGS-1330</i>	CO, H <sub>2</sub> , Methane	2
<i>UST GGS-2330</i>	CO, Ethanol, Methane	2
<i>UST GGS-10330</i>	CO, H <sub>2</sub> , Butane	2
<i>Figaro TGS-2600</i>	CO, H <sub>2</sub> , Methane, Ethanol	2
<i>Figaro TGS-822</i>	CO, Ethanol, Acetone	2

Table 1 also indicates which compounds each sensor is sensitive to. According to prior study, the compounds produced by trees are usually alcohol-based, which influences the choice of gas sensors [1][7]. The total number of gases that sensors are sensitive to is unknown, but a table illustrates which ones they react to the best.

Sensors MQ-3 and MQ-135 came on a breakout board with an integrated circuit. These sensors are powered by a 5V power supply, and the analog output is connected directly to the output. Initially, these sensors featured load resistors of 1k $\Omega$  and 1.5k $\Omega$ , but eventually showed poor resolution and sensitivity to gasses. Load resistors of 10k $\Omega$  and 20k $\Omega$  were soldered instead, and better sensitivity has been achieved.

As the circuit used eight MQ sensors, multiplexor CD74HC4067 was used to reduce the number of pins required for Arduino connection. The sensor output was attached as an input to the multiplexor, which picked each channel and collected data from each sensor. As previously stated, MQ sensors were coupled to a 5V supply, and the output ranged

from 0 to 5V. Because Arduino Due operates at 3.3V, this voltage had to be reduced using a voltage divider. The multiplexor's output is coupled to a voltage divider made up of 910k $\Omega$  and 510k $\Omega$ , resulting in a range of 0-3V.

Other sensors, which did not use an integrated circuit, required load resistance to be established. The load resistance used for each sensor is shown in Table 2. As shown in Fig. 4, each sensor required a heater supply and a sensitive layer supply. The heater supply was 5V, as indicated in the datasheets, with the sensitive layer supply at 3.3V. Voltage across the load resistor is obtained as an output and sent to the Arduino for further processing.

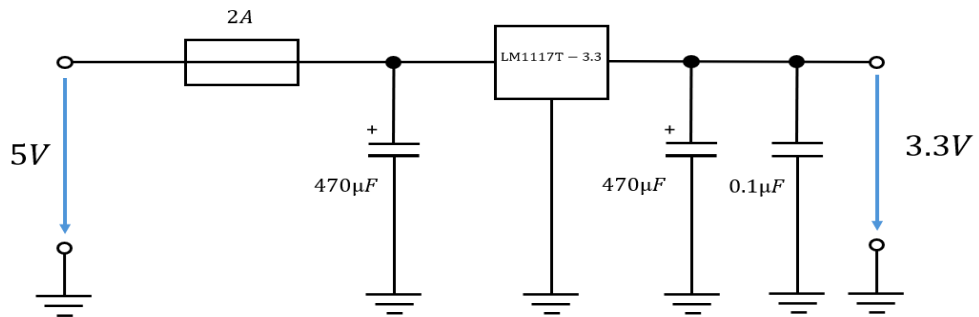
*Table 2: Load sensor values*

Sensor	Load resistance [k $\Omega$ ]
<i>UST GGS-1330</i>	47
<i>UST GGS-2330</i>	220
<i>UST GGS-10330</i>	220
<i>Figaro TGS-2600</i>	4.7
<i>Figaro TGS-822</i>	4.7

In addition to gas sensors, the circuit also monitored temperature, humidity, and TVOC levels. This was accomplished through the employment of ENS160 and BME280 sensors incorporated into a single circuit. They are designed to communicate via the I2C protocol. Pull-up resistors were already included in the circuit, therefore integrating them was unnecessary. The application in Arduino code will be discussed in the parts below.

Power was delivered using a power brick connected to the PCB using a barrel jack. This provided 5V supply to the circuit; however, 3.3V was achieved using a voltage regulator, LM-1117T-3.3, with additional electrolytic capacitors of 470 $\mu$ F, for the stable voltage output and a ceramic capacitor of 100nF for high frequency filtering. This design provided stable supply for the circuit. A 2A fuse has been added into the circuit to provide

protection. Additional protections are deemed unnecessary, as the overvoltage issue is not anticipated. An equivalent circuit is visualized in Fig. 11.

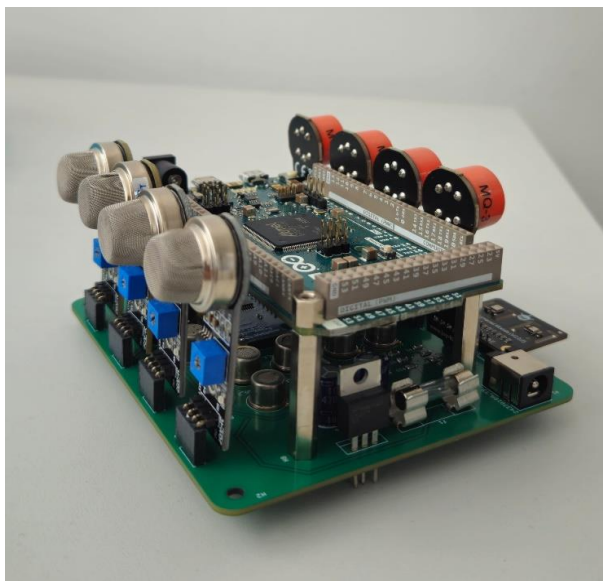


*Fig. 11: Equivalent circuit of 3.3V power supply*

### 3.3 PCB DESIGN

A breadboard was used to carry out the first circuit design. While the circuit was simple to use and modify, it was not steady or effective. As a result, PCBs were developed to be more reliable and smaller. A two-layer PCB was developed using KiCad, an open-source software.

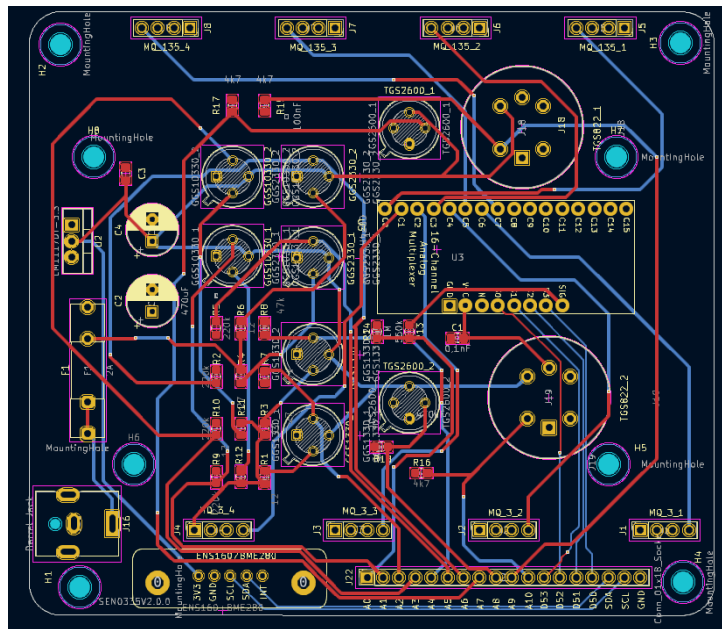
A PCB measuring  $10.8 \times 9.5$  cm had all of the circuit's components, as well as an additional Arduino Due set mounted on top. This method enabled a strong and compact circuit design. Fig. 12 shows a finished result.



*Fig. 12: Design of e-nose system*

As previously stated, the PCB was constructed with two layers: the top layer was a ground plate and the bottom layer was a 5V plate, making connections easier. To save space compared to THT components, resistors and capacitors were used in compliance with SMD standard 0805.

Fig. 13 depicts a PCB layout in KiCad. In a circuit, there are two sorts of traces: power and signal lines. Power traces were built to handle more current and so have a width of 0.53 mm, whilst signal traces have a width of 0.25 mm.



*Fig. 13: PCB-layout using KiCad*

All parts of PCB are shown in Fig. 14 containing sensors, power supply, multiplexor, load resistors and others.

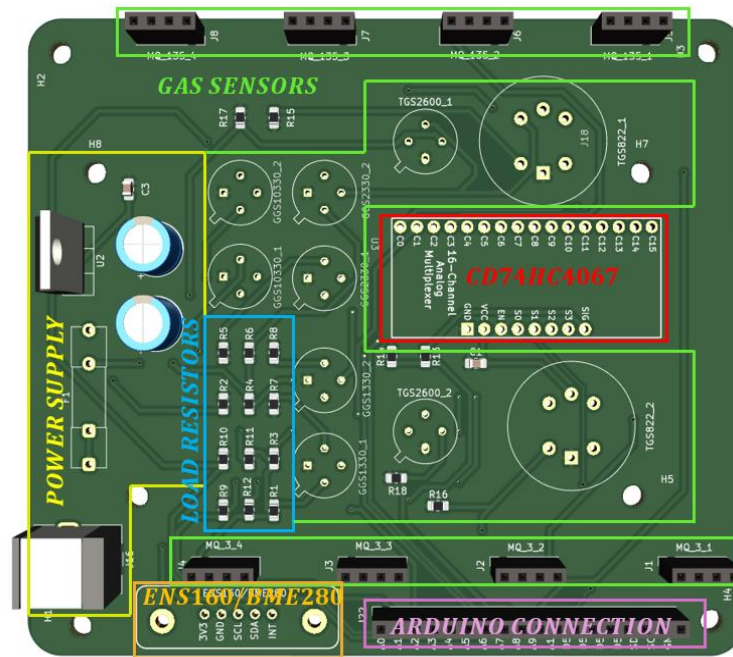


Fig. 14: Component layout of the PCB

### 3.4 DATA ACQUISITION AND PROCESSING

A general flow of data is shown in Fig. 15, where analog values from the gas sensor array (GSA) are collected by Arduino Due, processed by Python, and finally visualized by a graphical user interface (GUI).

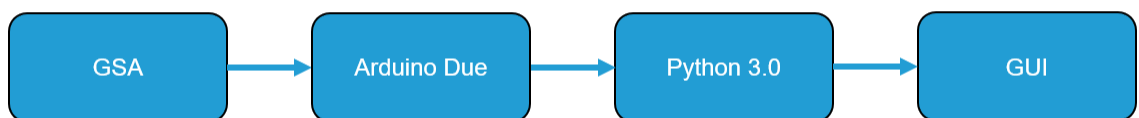


Fig. 15: Data flow

Further sections will focus on explaining each phase in greater detail.

#### 3.4.1 DATA ACQUISITION WITH ARDUINO DUE

Data was collected using the Arduino Due. The Arduino Due offered a sturdy and dependable data collection system. The code for reading data from sensors was written using the Arduino IDE software and the C++ programming language.

As with any computer code, the first step was to initialize variables, define sensor libraries, and designate input and output pins. The wire library created I2C communication with sensors ENS160 and BME280, using addresses 0x53 for ENS160 and 0x74 for BME280, as specified by datasheets. The implementation is demonstrated in the following code snippet (see Fig. 16).

```
//ENS160 -----  
DFRobot_ENS160_I2C ENS160(&Wire, /*I2CAddr*/ 0x53);  
  
// BME280 -----  
typedef DFRobot_BME280_IIC BME;  
BME bme(&Wire, 0x76);
```

*Fig. 16: Establishing I2C communication*

Following the initial stage, the setup function is developed. The setup function is demonstrated in the next code snippet. The setup function required establishing serial communication between Arduino and Python at a baud rate of 9600, enabling sensors, creating output pins for multiplexer control, and input pins for gas sensor data.

A function is built to regulate which sensor multiplexors receive input. A code will iterate through the combinations, collect data from sensors, and store it in variables that will be utilized to determine resistance (see Fig. 17).

```
void setup() {  
    Serial.begin(9600); // Begin serial communication.  
  
    // Start ENS160.  
    while( NO_ERR != ENS160.begin() ){  
        Serial.println("Communication with device failed, please check connection");  
        delay(3000);  
    }  
  
    ENS160.setPWRMode(ENS160_STANDARD_MODE);  
    ENS160.setTempAndHum(/*temperature=*/25.0, /*humidity=*/50.0);  
  
    // Start BME280.  
    bme.reset();  
  
    while(bme.begin() != BME::eStatusOK) {  
        Serial.println("BME280 begin failed");  
        printLastOperateStatus(bme.lastOperateStatus);  
        delay(2000);  
    }  
  
    // Select MUX selection pins as output.  
    pinMode(MUX_SELECT_PIN_S0, OUTPUT);  
    pinMode(MUX_SELECT_PIN_S1, OUTPUT);  
    pinMode(MUX_SELECT_PIN_S2, OUTPUT);  
    pinMode(MUX_SELECT_PIN_S3, OUTPUT);  
  
    // ADC setup  
    pinMode(A0, INPUT);  
    ...  
    pinMode(A10, INPUT);  
    analogReadResolution(12); //By default arduino uses 10 bit resolution.  
  
    delay(150);  
}
```

*Fig. 17: Setup function*

The final stage of data collecting is reading values from sensors. Sensors like the ENS160 and BME280 have built-in reading functions, whilst other gas sensors provide analog values. Pins ranging from A0 to A10 were utilized to read voltages across load resistors, and a sensor resistance value was derived using the voltage divider rule, as described in the theory section. The following sample shows an example of an analog read, in which the value read from the ADC is converted to voltage by multiplying it by 3.3 (supply voltage) and dividing by 4095, as 12-bit resolution is employed (see Fig. 18).

```
//Reading BME280 and ENS160 sensors
float t = bme.getTemperature();
float h = bme.getHumidity();

float CO2=ENS160.getECO2();
float TVOC=ENS160.getTVOC();

//Reading TGS2600
delay(150);
int sensor2600_1 = analogRead(A7);
float voltage2600_1 = sensor2600_1 * (3.3 / 4095);
float TGS2600_1 = (3.3/ voltage2600_1 - 1) *4700;
delay(150);
```

*Fig. 18: Reading values from sensors*

The similar method is followed for the remaining gas sensors. Data is stored in variables and then sent as a string to Python for additional processing. Data was delimited with a comma for simpler processing in Python, as shown in Fig. 19.

```
Time, Temperature, Humidity, CO2, TVOC, MQ3, MQ135, GGS1330, GGS2330, GGS10330, TGS2600, TGS822
```

*Fig. 19: String data*

A full Arduino code is provided in Appendix I section.

### 3.4.2 DATA ANALYSIS USING PYTHON

This section will explain how to store, process, and analyze data. Python will be used for further data analysis in conjunction with the editor tool Visual Studio Code. Python offered numerous useful libraries for data analysis and visualization. The libraries include matplotlib for graph plotting and visualization, sklearn for machine learning methods, pandas for datasets, and numpy for array manipulation.

Code snippets and examples will be used throughout this thesis to demonstrate major functions and methods. For the whole implementation and full codebase, please see the GitHub repository<sup>3</sup>.

---

<sup>3</sup> The full implementation and code can be found in the: <https://github.com/Leox47/DigiNose.git>

### 3.4.2.1 REAL-TIME MONITORING

First, a challenge of real-time data monitoring will be addressed. This arrangement enables the user to monitor gas sensor resistance, temperature, humidity, and TVOC levels in real time. As seen in the previous section, data from Arduino is transmitted via serial connection in the form of a string with a comma as a separator. The following code snippet demonstrates the `serial_connection()` function, which establishes a connection using the same COM port and baud rate as in Arduino. In addition, a new CSV file with a user-defined name is produced to hold all of the data (see Fig. 20).

```
def serial_connection(): # Connect to arduino
    global file_name, ser

    # Serial connection
    ser = serial.Serial('COM3', baudrate=9600, timeout=1)

    # Set file name
    file_name = f"{create_file.get()}.csv" #

    # Clear file if data inside
    f = open(file_name, "w+")
    f.close()

    # Function to write data to csv file
    def write_to_csv(df):

        df.to_csv(file_name, mode='a', header=False, index=False)
        read_data()

    # Read data from arduino and split it into columns
    def generate_data():

        while True:
            line = ser.readline().decode('utf-8').strip()

            if "failed to init chip, please check the chip connection" in line:
                print("Please check the chip connection!")

            elif line:
                splitted_line = line.split(',') # Splits string
                df = pd.DataFrame([splitted_line]) # Creates dataframe type
                write_to_csv(df) # Writes data to csv file in columns

        yield
```

Fig. 20: Arduino-Python communication

Because the data was given as a string separated by commas, it can be quickly divided into columns using the `split` command in the `generate_data()` function. After splitting, the

data is saved as a dataframe, which is structured similarly to an Excel spreadsheet. Finally, the function `write_to_csv(df)` saves the data to a CSV file.

After the data has been saved, it is read from the CSV file (see Fig. 21). Using the Pandas function. The variables store the data returned by `iloc()`. Since the gas resistance of different sensors can range from a few kilohms to a few megaohms, scaling is performed with `StandardScaler()`, and the resulting data is shown.

```
def read_data():
    # Reading data

    data_python = pd.read_csv(file_name, header=None)

    # Extracting data
    time_data = data_python.iloc[:, 0]
    co2=data_python.iloc[:,1]
    tvoc=data_python.iloc[:,2]
    temp = data_python.iloc[:, 4]
    humidity = data_python.iloc[:, 3]
    sensors=data_python.iloc[:, 5:12]

    # Scaling Data
    scaled_data = StandardScaler().fit_transform(sensors)

    MQ3=scaled_data[:, 0]
    ...
    TGS822=scaled_data[:, 6]

    # Plots data on ax1
    ax1.clear()
    ax1.plot(time_data / 60, MQ3,color='#0072BD',linewidth=2, label="MQ3")
    ...
    ax1.plot(time_data / 60, TGS822,color='pink',linewidth=2, label="TGS822")
```

*Fig. 21: Data processing in Python*

Similarly, temperature and humidity are read and shown, and CO2 and TVOC readings are updated.

The function `FuncAnimation()` from the Matplotlib library is used to display live data and plot it in real time. This function enables real-time data monitoring and continuous graph updates. Importing Data

The user interface also allows you to import, and plot measured data. Using `filedialog.askopenfilenames()`, the user can select a desired file and data will be displayed.

The same method of data reading and processing is demonstrated in the previous section Real Time Monitoring.

#### 3.4.2.2 PRINCIPAL COMPONENT ANALYSIS

A first approach of machine learning methods is PCA. In Python the sklearn library provides tools for PCA application. Before PCA analysis, data is scaled using a StandardScaler to ensure equal contribution to the analysis, PCA is initialized, and data is fitted and transformed. Fit function allows PCA to create a principal component, after which, using a transform function, data is projected on principal components. Creating a DataFrame allows one to easily visualize the transformed data. For example, a scatter plot can be generated using the first principal component (PC1) on the x-axis and the second principal component (PC2) on the y-axis, as seen if Fig. 22 [23].

```
def pca_analysis(time_data, temperature_data, humidity_data, sensor_data):  
  
    # Scaling data for PCA  
    scaled_data = StandardScaler().fit_transform(sensor_data)  
  
    # Perform PCA  
    pca = PCA(n_components=2)  
    pca.fit(scaled_data)  
    pca_data = pca.transform(scaled_data)  
  
    pca_df = pd.DataFrame(pca_data, columns=labels)
```

*Fig. 22: Principal Component Analysis in Python*

#### 3.4.2.3 LINEAR DISCRIMINANT ANALYSIS

The LDA package, like the PCA package, is part of the Sklear library. In addition to calling, LDA necessitates extra data processing. Primarily, only necessary data is retained. This means that temperature, humidity, CO2, and TVOC data are eliminated because they are not required for sensor data processing. Second, from each measurement file, the last data row is extracted for analysis and stored in a new CSV file. This is done for all measurements taken. LDA is a supervised approach, therefore labeling is also

required. The file name is used as a label and put to a CSV file, which is subsequently prepared for LDA applications.

Prior using LDA, the number of features and data points must be determined and defined as list  $y$ . Values in square brackets denote features (substances or trees), but the values they are multiplied by represent the amount of data for each feature. After then, the data is divided into two sets: training and testing. LDA is defined; data is fitted and transformed. As with PCA analysis, it is represented by a scatter plot and observed on a screen. Implementation can be seen in Fig. 23.

```
def LDA_function(csv_file_path, test_size_set):
    # Create an empty list to store the CSV data
    csv_data = []

    # Read the CSV file and store its contents in the list
    with open(csv_file_path, 'r') as file:
        reader = csv.reader(file)
        for row in reader:
            modified_row = row[1:]
            csv_data.append(modified_row)

    # Convert the numerical values to float
    numerical_rows = [[float(value) for value in row] for row in csv_data]

    # Normalize the numerical values using MinMaxScaler
    scaler = MinMaxScaler()
    normalized_rows = scaler.fit_transform(numerical_rows)

    y = [0] * 10 + [1] * 15 + [2] * 10 + [3] * 15 + [4] * 17 + [5] * 15 + [6] * 10 + [7] * 10 + [8] * 15

    # Splitting data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(normalized_rows, y,
        test_size=float(test_size_set)/100)

    # Initializing and training the LDA model
    lda = LinearDiscriminantAnalysis()
    lda.fit(X_train, y_train)

    # Transforming the data and plotting the transformed data
    X_train_transformed = lda.transform(X_train)

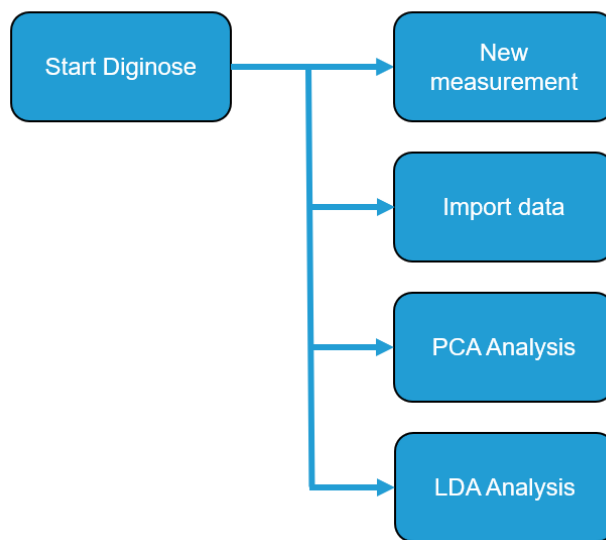
    labelss = ['3-Carene', 'Air', 'Alpha pinene', 'D-limonene', 'Ethanol', 'Everbanol',
        'Methyl', 'Mixture', 'Verbenol']
    colors = ['g', 'pink', 'b', 'orange', 'm', 'y', 'r', 'k', 'c']
    fig_1 = plt.figure(figsize=(12, 5), dpi=85, edgecolor='black')
    ax1 = fig_1.add_subplot(111)

    for i in range(len(labelss)):
        indices = np.where(np.array(y_train) == i)[0]
        ax1.scatter(X_train_transformed[indices, 0], X_train_transformed[indices, 1],
            label=labelss[i], color=colors[i])
```

Fig. 23 Linear Discriminant Analysis application in Python

### 3.4.3 USER INTERFACE (UI)

An objective of user interface (UI) is to make user's experience easy and intuitive, requiring minimum effort to receive desired outcome. A user interface for Diginose provided easier visualization of processed data and easier analysis of smells. In Fig. 24, features of user interface are presented. User can choose to start new measurement, import already measured data, perform PCA analysis and perform LDA analysis. Entire user interface was created by Tkinter library and its packages.

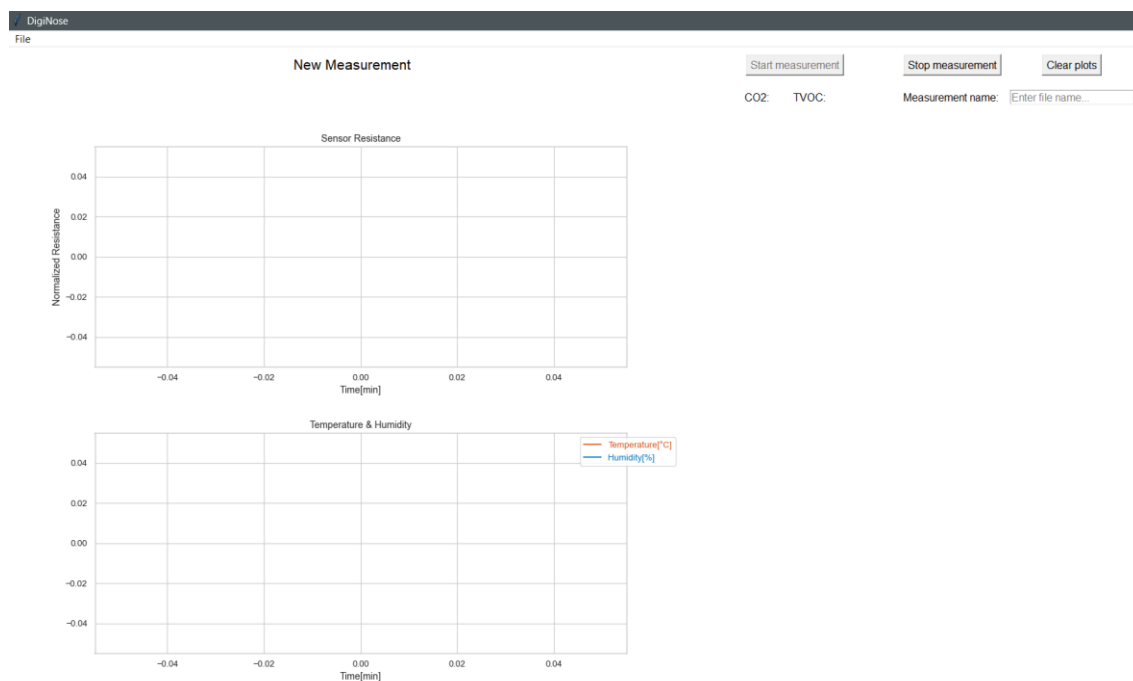


*Fig. 24: User interface features*

Each of the features will be discussed in the following sections.

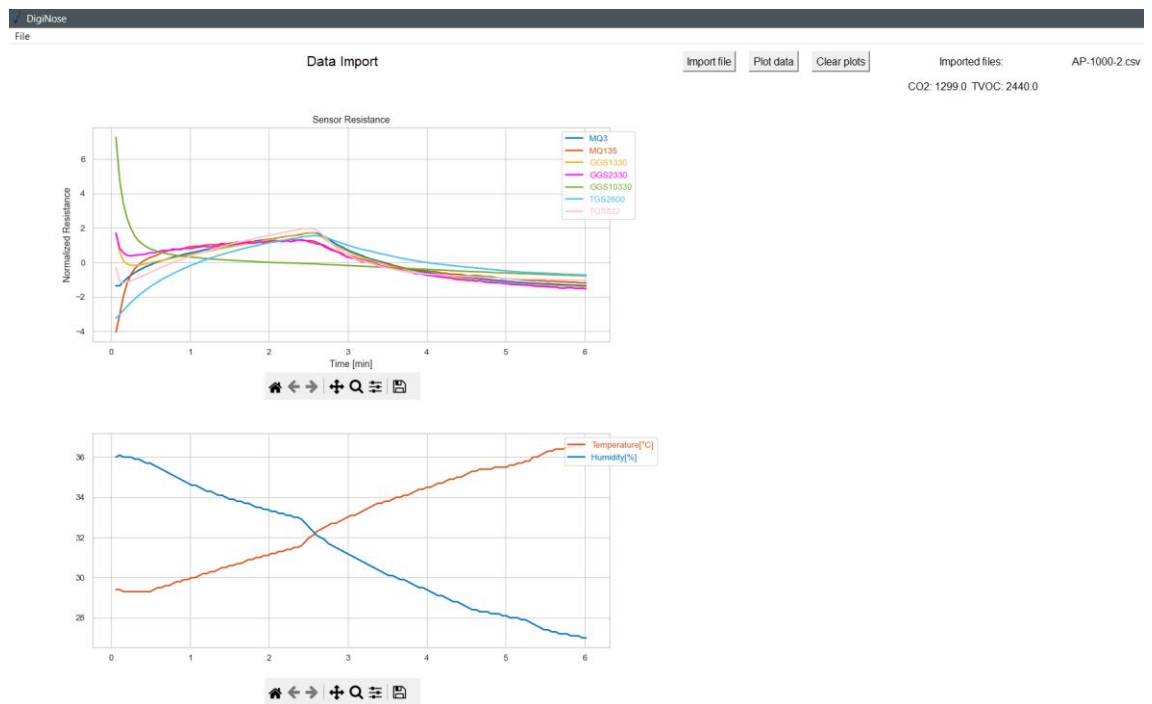
#### 3.4.3.1 NEW MEASUREMENT AND IMPORT DATA FEATURE

In Fig. 25, new measurement feature is shown. Starting a new measurement, user can detect real-time change of resistance, temperature, humidity, CO<sub>2</sub> and TVOC level. Once the user enters file name, measurement can be started and resistance of each of the sensors will be monitored.



*Fig. 25: New measurement feature*

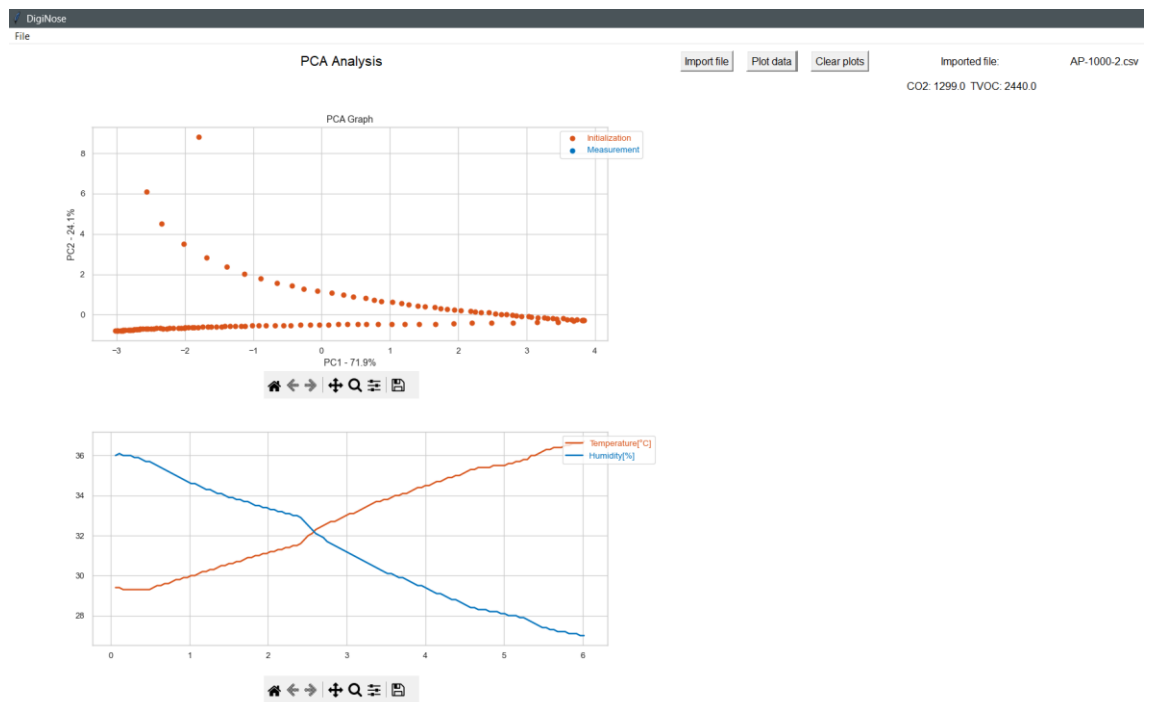
Another feature is importing already measured data. In following Fig. 26, user is able to import file that is already measured and plot it. This figure represents a measurement taken for Alpha-pinene at 1000ppm. On resistance plot, resistance of sensors is shown, and changes can be seen clearly. Temperature and humidity are also shown. On the right, last value of CO<sub>2</sub> and TVOC level is shown.



*Fig. 26: Import data feature*

#### 3.4.3.2 PCA ANALYSIS FEATURE

Since initial study was conducted by PCA, this feature is also implemented in user interface. Here, PCA is applied to one dataset and can show difference in air measurements and gas measurements. Graph also indicates how much of variance each principal component capture in percentage. PCA feature is displayed in Fig. 27.



*Fig. 27: PCA Visualization*

### 3.4.3.3 LDA ANALYSIS FEATURE

LDA analysis was also a feature implemented in user interface. Fig. 28 shows the implementation of LDA. User should provide a folder filled with measurements in form of CSV files. Once folder is selected, processing of data will start. When it is done, user can enter how much of data to assign to training and how much to testing. In this case 20% is chosen for testing and 80% for training. Finally, data can be plotted and data of LDA analysis is shown in graphs, one indicating training set and other testing set. Testing and training are also indicated on top right corner.

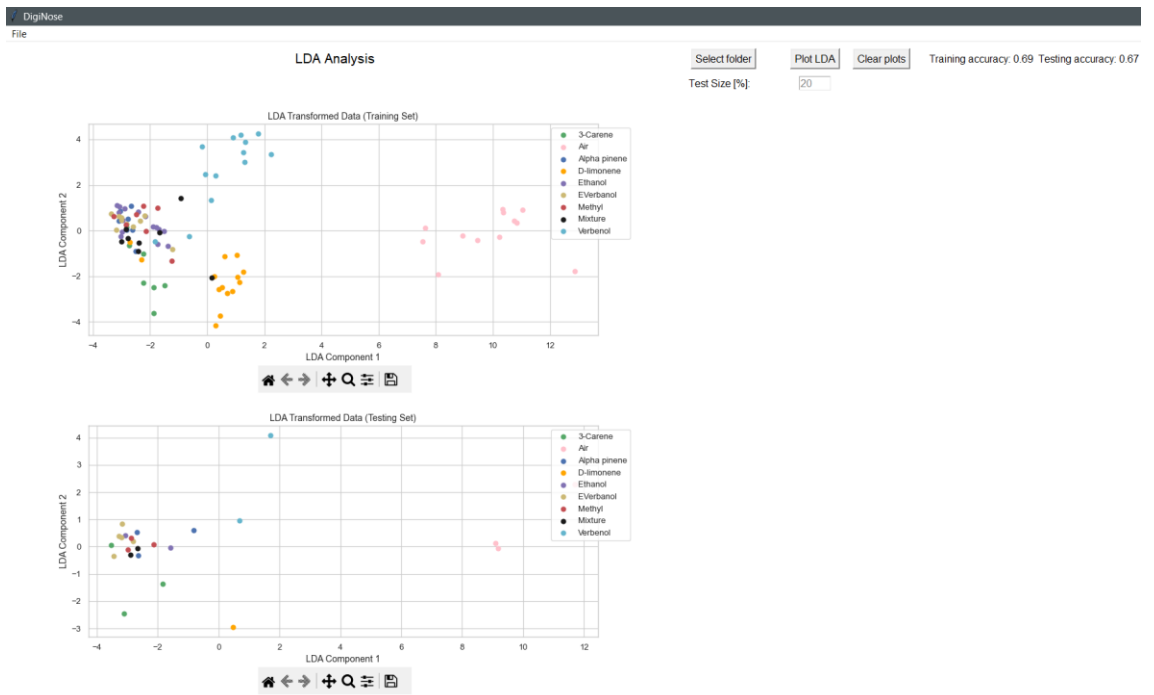


Fig. 28: LDA Visualization

## 4 RESULTS AND DISCUSSION

This chapter examines and analyzes the results yielded for two specific tests from Diginose and Smell Inspector. The similarities and differences between Smell Inspector and Diginose are explained in sections 4.1 and 4.2.

### 4.1 DIGINOSE

Diginose was utilized for measurements of substances and trees. In first section, resistance measurement is shown, and different behaviour of gas sensors is seen. In second section, LDA results are shown and discussed.

#### 4.1.1 RESISTANCE MEASURING

During measurement, resistance of gas sensors is monitored. As seen from Fig. 29, initial resistance is high and decreases as sensors heat up. Some sensors heat up faster than others and therefore their initial high resistance is not shown. After sensors reach stable resistance at around 2.5 minutes, a gas is introduced in their environment, in this case 3-carene. Resistance decreases even more since the conductivity of sensitive layer increases. Measurement is done until minute 6, when most of the gas has run out. Same measurement procedure is repeated for all substances and trees with slight difference in timing. Last value measurement is taken for LDA analysis.

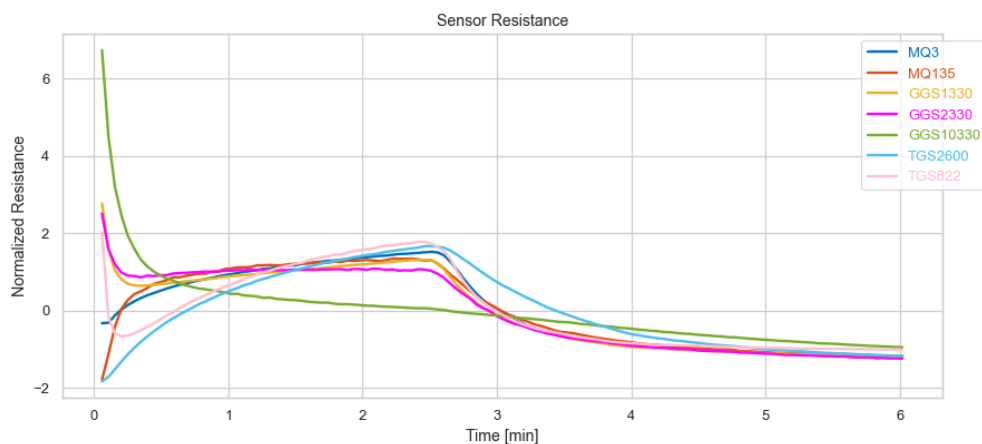


Fig. 29: Resistance measurement

### 4.1.2 LINEAR DISCRIMINANT ANALYSIS

Fig. 30 shows a significant overlap of sick and healthy trees. The cause could be that some trees are less stressed than others, or that healthy trees are not as healthy. A training set containing 80% of the data produced an accuracy score of 80%. The testing data set included 20% of all data and demonstrated improved separation, with an accuracy score of 84%. The testing set is shown in Fig. 31.

It's worth mentioning that the dataset is small. Further testing aims to increase the data set in order to provide better and more efficient forecasts.



Fig. 30: LDA Analysis of trees in training set (Diginose)

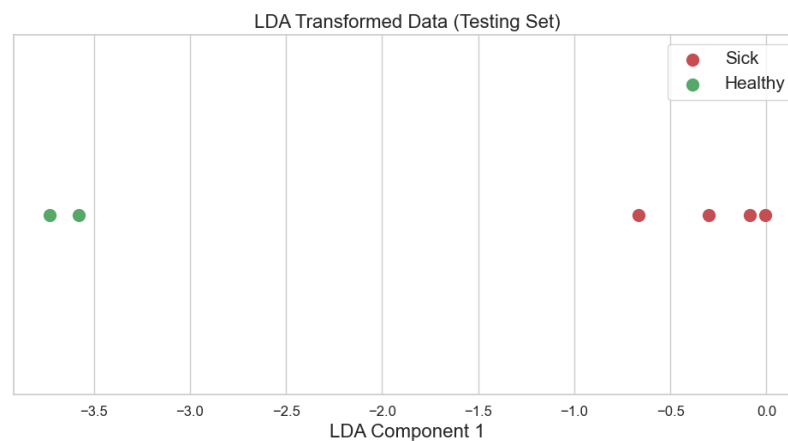


Fig. 31: LDA Analysis of trees in testing set (Diginose)

Certain substances were classified during the substance data analysis, while others proved difficult to classify. D-limonene, ethanol, verbanol, and air were clearly separated, whereas 2-methyl, 3-carene, alpha pinene, and a combination overlapped. This issue arises as a result of the MOx sensors' resistance, which can produce identical findings for distinct scents at varying concentrations. This was demonstrated in the theory part of Fig. 3. Fig. 32 and Fig. 33 show an LDA analysis of chemicals.

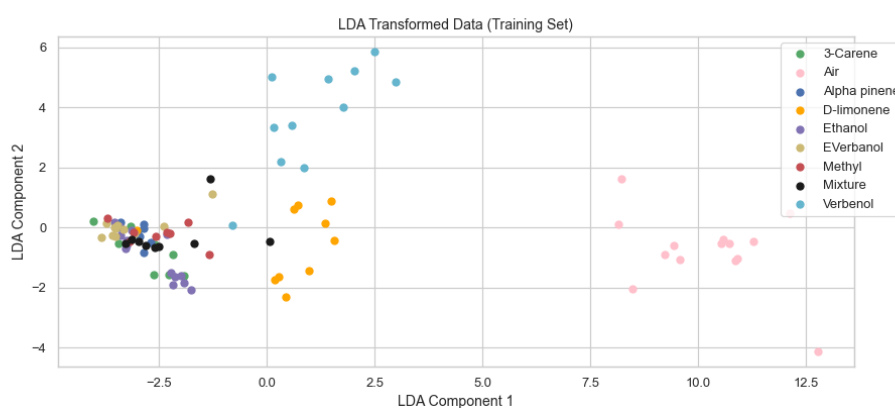


Fig. 32: LDA Analysis of substances in training set (Diginose)

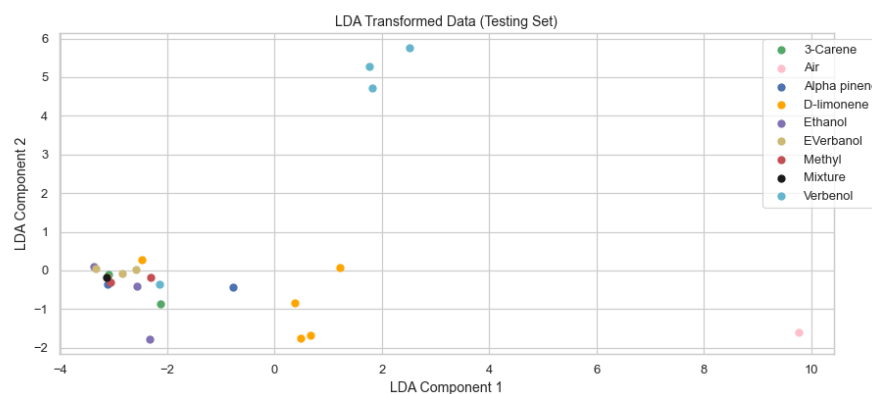


Fig. 33: LDA Analysis of substances in testing set (Diginose)

Similar to a tree analysis, 80% of the data was used for training and 20% for testing purposes. In the substance analysis, the accuracy score ranged about 80%, the training accuracy for substances was around 70%, and the testing accuracy dropped to about 75%.

## 4.2 SMELL INSPECTOR:

As previously stated, the Smell inspector was utilized to ensure that the Diginose readings were within acceptable limits. Data from trees and chemicals were collected in the same approach as with Digi-nose, and LDA analysis was performed. Analysis of tree data yielded 88% accuracy in the training set, however testing accuracy was relatively low, about 33%. This resulted in overfitting, which might be attributed to a small dataset. In the future, the goal is to use machine learning algorithms that are resilient to overfitting, such as random forest, and to increase the quantity of data sets. Fig. 34 and Fig. 35 show LDA analysis of Smell inspector data.

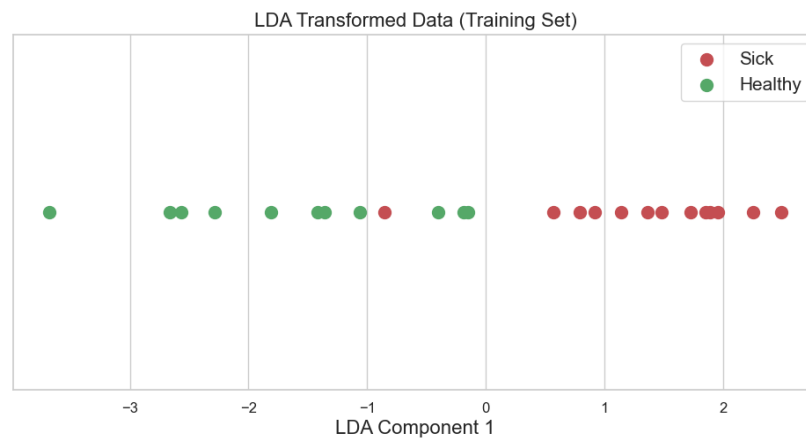


Fig. 34: LDA Analysis of trees in training set (Smell Inspector)

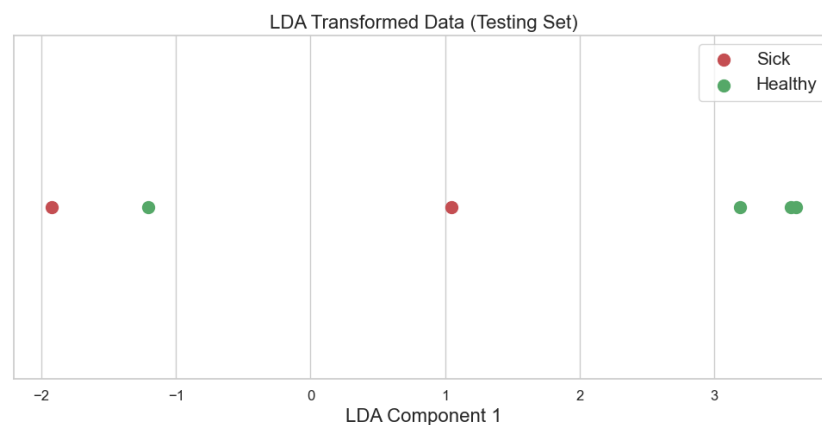


Fig. 35: LDA Analysis of trees in testing set (Smell Inspector)

Substance analysis produced better results than tree analysis. High accuracy of 96% on a training set demonstrated great separability of compounds, with overlap of chemicals such as mixture, alpha pinene, and 2-methyl. Testing accuracy decreased marginally but remained good at 88%. Fig. 36 and Fig. 37 show the LDA results.



Fig. 36: LDA Analysis of substances in training set (Smell Inspector)

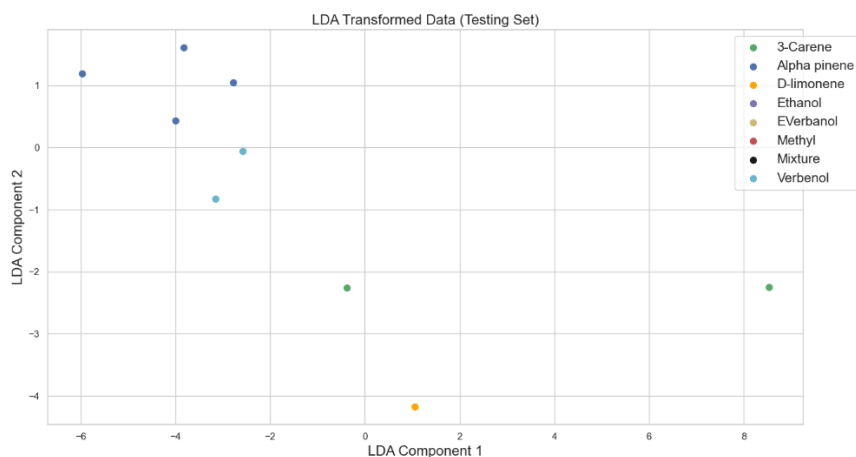


Fig. 37: LDA Analysis of substances in testing set (Smell Inspector)

## 5 CONCLUSION

In this project, an electronic nose using metal-oxide gas sensors to differentiate between healthy and sick trees by distinguishing unique volatile organic compounds (VOCs) was successfully implemented. The e-nose system, together with an Arduino Due microcontroller, further improved by Python data analysis, proved an effective way for detecting and classifying VOCs related to forest health.

Our methodology included developing a gas sensor array that would be capable of detecting a broad range of different odors and providing information on how healthy trees are. The procedure involved the application of PCA and LDA for data analysis and the design of a graphical user interface for real-time monitoring. Initial outcomes demonstrated the ability to show a difference between healthy and ill trees with a reasonable level of accuracy. While the PCA provided initial separation of data, the LDA provided clearer class separation, with some overlapping in some cases. The development of a custom PCB further enhanced the sensor array, improving the e-nose's efficiency and reliability.

Despite the success, an obstacle such as the MOx sensors' limited selectivity for different gases has shown a level of difficulty. A future study will focus on solving this problem by implementing an artificial neural network to enhance e-noses' selectivity for other gases.

In conclusion, e-nose represents an optimistic and cost-effective solution to more prevalent approaches such as GC-MS for real-time monitoring of VOCs. This technique has the potential to be applied not only for forest health monitoring but also for agricultural applications, contributing to environmental sustainability while making monitoring options more accessible. Further improvements in sensor selectivity and machine learning algorithms will be necessary to fully exploit the potential of this newly developed technology.

## 6 OUTLOOK

Several improvements could be made to strengthen the system's reliability. To avoid temperature dependency in the circuit, a temperature compensation circuit could be introduced. Another improvement might be made to the machine learning aspect. Given the possibility of data overfitting, methods such as random forest could be used to create neural networks with a low likelihood of overfitting, resulting in more robust and generalizable models. Furthermore, different gas concentrations could be evaluated to generate characteristics for essential compounds in trees. By training a neural network on varied gas concentrations, it may enhance its capacity to differentiate between distinct smells and minimize its probability of making inaccurate predictions.

## 7 LIST OF FIGURES

<i>Fig. 1: Schematic representation of MOx sensor</i> .....	5
<i>Fig. 2: Response of a sensor to switching on and off a reducing gas</i> .....	6
<i>Fig. 3: Linear correleation between resistance and gas concentration</i> .....	7
<i>Fig. 4: Equivalent circuit of a gas sensor</i> .....	7
<i>Fig. 5: Construction of the principlal component</i> .....	10
<i>Fig. 6: Class separability in LDA</i> .....	11
<i>Fig. 7: Inter-Integrated Circuit communication protocol</i> .....	12
<i>Fig. 8: Setup of trees in grow box</i> .....	15
<i>Fig. 9: Measurement setup for tree measurements</i> .....	16
<i>Fig. 10: Measurement setup for substance measurements</i> .....	17
<i>Fig. 11: Equivalent circuit of 3.3V power supply</i> .....	20
<i>Fig. 12: Design of e-nose system</i> .....	21
<i>Fig. 13: PCB-layout using KiCad</i> .....	21
<i>Fig. 14: Component layout of the PCB</i> .....	22
<i>Fig. 15: Data flow</i> .....	22
<i>Fig. 16: Establishing I2C communication</i> .....	23
<i>Fig. 17: Setup function</i> .....	24
<i>Fig. 18: Reading values from sensors</i> .....	25
<i>Fig. 19: String data</i> .....	25
<i>Fig. 20: Arduino-Python communication</i> .....	26
<i>Fig. 21: Data processing in Python</i> .....	27
<i>Fig. 22: Principal Component Analysis in Python</i> .....	28
<i>Fig. 23 Linear Discriminant Analysis application in Python</i> .....	29
<i>Fig. 24: User interface features</i> .....	30
<i>Fig. 25: New measurement feature</i> .....	31
<i>Fig. 26: Import data feature</i> .....	32
<i>Fig. 27: PCA Visualization</i> .....	33
<i>Fig. 28: LDA Visualization</i> .....	34
<i>Fig. 29: Resistance measurement</i> .....	35

<i>Fig. 30: LDA Analysis of trees in training set (Diginose)</i> .....	36
<i>Fig. 31: LDA Analysis of trees in testing set (Diginose)</i> .....	36
<i>Fig. 32: LDA Analysis of substances in training set (Diginose)</i> .....	37
<i>Fig. 33: LDA Analysis of substances in testing set (Diginose)</i> .....	37
<i>Fig. 34: LDA Analysis of trees in training set (Smell Inspector)</i> .....	38
<i>Fig. 35: LDA Analysis of trees in testing set (Smell Inspector)</i> .....	38
<i>Fig. 36: LDA Analysis of substances in training set (Smell Inspector)</i> .....	39
<i>Fig. 37: LDA Analysis of substances in testing set (Smell Inspector)</i> .....	39

## **8 LIST OF TABLES**

<i>Table 1: Type of sensors</i> .....	18
<i>Table 2: Load sensor values</i> .....	19

## 9 BIBLIOGRAPHY

- [1] Antonelli, M., Donelli, D., Barbieri, G., Valussi, M., Maggini, V., & Firenzuoli, F. (2020). Forest volatile organic compounds and their effects on human health: A state-of-the-art review, *International Journal of Environmental Research and Public Health* 17.
- [2] Bishop, C. M., & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*, volume 4, Springer.
- [3] Built In. (2020, November 17). Step-by-step explanation of principal component analysis. Built In. <https://builtin.com/data-science/step-step-explanation-principal-component-analysis>
- [4] Digitalogy. (2021, September 7). Node.js vs. Python for backend development. Digitalogy. <https://www.digitalogy.co/blog/nodejs-vs-python-for-backend-development>
- [5] Eranna, G. (2012). *Metal Oxide Nanostructures as Gas Sensing Devices*. CRC Press.
- [6] Gardner, J. W., & Persaud, K. C. (2001). *Electronic Noses and Olfaction 2000: Proceedings of the 7th International Symposium on Olfaction and Electronic Noses*, Brighton, UK, July 2000, CRC Press.
- [7] Holopainen, J. K., & Gershenzon, J. (2010). Multiple stress factors and the emission of plant VOCs, *Trends in plant science* 15.
- [8] Jaaniso, R., & Tan, O. K. (2013). *Semiconductor gas sensors*, Elsevier.
- [9] Khorramifar, Ali et al. (2023). Environmental Engineering Applications of Electronic Nose Systems Based on MOX Gas Sensors. *Sensors (Basel, Switzerland)* vol. 23,12 5716. 19 Jun. 2023.
- [10] Wang, Y., Zhang, W., Gao, R., Jin, Z., & Wang, X. (2021). Recent advances in the application of deep learning methods to forestry, *Wood science and technology* 55.

- [11] Yadav, Anurag, et al. (2023). Nanotechnology for Precision Agriculture. Encyclopedia. Web. 13 June, 2023.
- [12] Hameed, M. M., Masood, A., Srivastava, A., Abd Rahman, N., Mohd Razali, S. F., Salem, A., & Elbeltagi, A. (2024). Investigating a hybrid extreme learning machine coupled with Dingo Optimization Algorithm for modeling liquefaction triggering in sand-silt mixtures. *Scientific reports*, 14(1), 10799. <https://doi.org/10.1038/s41598-024-61059-6>
- [14] Hudson, D. L. (2009). *Improving accuracy in microwave radiometry via probability and inverse problem theory* (Master's thesis, Brigham Young University). BYU ScholarsArchive. <https://scholarsarchive.byu.edu/etd/1991>
- [15] Zampieri, G., Vijayakumar, S., Yaneske, E., & Angione, C. (2019). Machine and deep learning meet genome-scale metabolic modeling. *PLoS computational biology*, 15(7), e1007084. <https://doi.org/10.1371/journal.pcbi.1007084>
- [16] Mehdi Ben Lazreg, Morten Goodwin, Ole-Christoffer Granmo. Information Abstraction from Crises Related Tweets Using Recurrent Neural Network. 12th IFIP International Conference on Artificial Intelligence Applications and Innovations (AIAI), Sep 2016, Thessaloniki, Greece. pp.441-452,10.1007/978-3-319-44944-9\_38. hal-01557644
- [17] Eliassi-Rad, T. (2001). Building intelligent agents that learn to retrieve and extract information (Doctoral dissertation, University of Wisconsin – Madison).
- [18] Ramuhalli, Pradeep, Walker, Cody, Agarwal, Vivek, & Lybeck, Nancy. Nuclear Power Prognostic Model Assessment For Component Health Monitoring. United States. <https://doi.org/10.13182/T124-34275>
- [19] eLearning Online Academy. (n.d.). *What is PCA in machine learning?* Retrieved August 6, 2024, from <https://elearningonlineacademy.com/amp/what-is-pca-in-machine-learning/>

[20] Ismail, I. A., Chukwuka, E. E., & Abiola, M. K. (2024). *Improving business insights using machine learning algorithms*.

[21] Abraham, A., Pllana, S., Casalino, G., Ma, K., & Bajaj, A. (Eds.). (2023). *Intelligent Systems Design and Applications: 22nd International Conference on Intelligent Systems Design and Applications (ISDA 2022) Held December 12-14, 2022-Volume 4* (Vol. 717). Springer Nature.

[22] Zanelli, F., Castelli-Dezza, F., Tarsitano, D., Mauri, M., Bacci, M. L., & Diana, G. (2021). Design and Field Validation of a Low Power Wireless Sensor Node for Structural Health Monitoring. *Sensors* (Basel, Switzerland), 21(4), 1050. <https://doi.org/10.3390/s21041050>

[23] Nijhof, B., Castells-Nobau, A., Wolf, L., Scheffer-de Gooyert, J. M., Monedero, I., Torroja, L., Coromina, L., van der Laak, J. A., & Schenck, A. (2016). A New Fiji-Based Algorithm That Systematically Quantifies Nine Synaptic Parameters Provides Insights into Drosophila NMJ Morphometry. *PLoS computational biology*, 12(3), e1004823. <https://doi.org/10.1371/journal.pcbi.1004823>

[24] Khorramifar, A.; Karami, H.; Lvova, L.; Kolouri, A.; Łazuka, E.; Piłat-Roz'ek, M.; Łagód, G.; Ramos, J.; Lozano, J.; Kaveh, M.; et al. Environmental Engineering Applications of Electronic Nose Systems Based on MOX Gas Sensors. *Sensors* 2023, 23, 5716. <https://doi.org/10.3390/s23125716>

[25] Capelli, L., Sironi, S., & Rosso, R. D. (2014). Electronic noses for environmental monitoring applications. *Sensors*, 14(11), 19979-20007.

# 10 APPENDIX

## 10.1 APPENDIX I: ARDUINO CODE

```

float VCC = 3.3;
int RL_1330 = 47000;
int RL_2330 = 220000;
int RL_MQ3 = 10000;
int RL_MQ135 = 20000;
float RS_1330 = 0;
float RS_2330 = 0;
float RS_10330 = 0;
float MQ_3 = 0;
float TGS2600=0;
float MQ_135 = 0;
float temp1 = 0;
float temp2 = 0;
float temp3 = 0;
float temp4 = 0;
float R_3=910000;
float R_2=420000;

#include <light_CD74HC4067.h>
#include "initialization.h"
#include "Wire.h"
#include "DFRobot_BME280.h"
#include <DFRobot_ENS160.h>

// Define MUX pins (S0, S1, S2, S3).
#define MUX_SELECT_PIN_S0 52
#define MUX_SELECT_PIN_S1 53
#define MUX_SELECT_PIN_S2 51
#define MUX_SELECT_PIN_S3 50

//ENS160 -----
DFRobot_ENS160_I2C ENS160(&Wire, /*I2CAddr*/ 0x53); // Define ENS160, use wire library for
I2C and assign .

// BME280 -----
typedef DFRobot_BME280_IIC BME;
BME bme(&Wire, 0x76); // Define BME280, use wire library and assign address.

// From library
void printLastOperateStatus(BME::eStatus_t eStatus)
{
    switch(eStatus) {
        case BME::eStatusOK: Serial.println("everything ok"); break;
        case BME::eStatusErr: Serial.println("unknow error"); break;
        case BME::eStatusErrDeviceNotDetected: Serial.println("device not detected"); break;
        case BME::eStatusErrParameter: Serial.println("parameter error"); break;
        default: Serial.println("unknow status"); break;
    }
}

void setup() {
    Serial.begin(9600); // Begin serial communication.

    // Start ENS160.
    while( NO_ERR != ENS160.begin() ){
        Serial.println("Communication with device failed, please check connection");
        delay(3000);
    }

    ENS160.setPWRMode(ENS160_STANDARD_MODE);
    ENS160.setTempAndHum(/*temperature=*/25.0, /*humidity=*/50.0); // Initial temperature
and humidity setting.

    // Start BME280.
    bme.reset();

    while(bme.begin() != BME::eStatusOK) {
        Serial.println("BME280 begin failed");
        printLastOperateStatus(bme.lastOperateStatus);
        delay(2000);
    }
}

```

```

// Select MUX selection pins as output.
pinMode(MUX_SELECT_PIN_S0, OUTPUT);
pinMode(MUX_SELECT_PIN_S1, OUTPUT);
pinMode(MUX_SELECT_PIN_S2, OUTPUT);
pinMode(MUX_SELECT_PIN_S3, OUTPUT);

// ADC setup // Must assign them as inputs, otherwise they dont behave nicely.
pinMode(A0, INPUT);
pinMode(A1, INPUT);
pinMode(A2, INPUT);
pinMode(A3, INPUT);
pinMode(A4, INPUT);
pinMode(A5, INPUT);
pinMode(A6, INPUT);
pinMode(A7, INPUT);
pinMode(A8, INPUT);
pinMode(A9, INPUT);
pinMode(A10, INPUT);
analogReadResolution(12); //By default arduino uses 10 bit resolution. This way 12 bit
resolution is set.

    delay(150);
}

void setMuxChannel(uint8_t channel) // Library used before didn't work as expected, so
just written my own function.
{
    digitalWrite(MUX_SELECT_PIN_S0, channel & 0b0001 );
    digitalWrite(MUX_SELECT_PIN_S1, channel & 0b0010 );
    digitalWrite(MUX_SELECT_PIN_S2, channel & 0b0100 );
    digitalWrite(MUX_SELECT_PIN_S3, channel & 0b1000 );
}

void loop() {

float time=micros()/1e6; // Time count.

// BME280 temperature and humidity data reading.
delay(150);
float  t = bme.getTemperature();
float  h = bme.getHumidity();

ENS160.setTempAndHum(/*temperature=*/t, /*humidity=*/h); // Set temperature and humidity
from BME280 sensor.

// ENS160 AIR Quality Sensor, read CO2 and TVOC values.
float CO2=ENS160.getECO2();
float TVOC=ENS160.getTVOC();

// Reading from MUX from MQ-3 Sensors.
delay(150);

// Initial setting
temp1 = 0;
temp2 = 0;
temp3 = 0;
temp4 = 0;
delay(10);

// Reading MQ-3
setMuxChannel(1);
temp1 = analogRead(A0)*3.3/4095;
delay(150);

setMuxChannel(2);
temp2 = analogRead(A0)*3.3/4095;
delay(150);

setMuxChannel(4);
temp3 = analogRead(A0)*3.3/4095;
delay(150);

setMuxChannel(6);
temp4 = analogRead(A0)*3.3/4095;
delay(150);
}

```

```

// Calculating resistance of MQ-3
MQ_3 = (((5*R_3)/temp1-R_3-R_2-RL_MQ3) + ((5*R_3)/temp2-R_3-R_2-RL_MQ3)+ ((5*R_3)/temp3-
R_3-R_2-RL_MQ3)+((5*R_3)/temp4-R_3-R_2-RL_MQ3)) / 4;

delay(150);
// Reset values
temp1 = 0;
temp2 = 0;
temp3 = 0;
temp4 = 0;
delay(10);

//Reading MQ-135
setMuxChannel(0);
temp1 = analogRead(A0)*3.3/4095;
delay(150);

setMuxChannel(3);
temp2 = analogRead(A0)*3.3/4095;
delay(150);

setMuxChannel(5);
temp3 = analogRead(A0)*3.3/4095;
delay(150);

setMuxChannel(7);
temp4 = analogRead(A0)*3.3/4095;
delay(150);

// Calculating resistance of MQ-135, same as MQ-3
MQ_135 = (((5*R_3)/temp1-R_3-R_2-RL_MQ3) + ((5*R_3)/temp2-R_3-R_2-RL_MQ3)+ ((5*R_3)/temp3-
R_3-R_2-RL_MQ3)+((5*R_3)/temp4-R_3-R_2-RL_MQ3)) / 4;

// Reading Sensor GGS-1330
delay(150);
temp1 = analogRead(A2)*3.3/4095;
temp2 = analogRead(A4)*3.3/4095;
RS_1330 = (((VCC - temp1)*RL_1330)/temp1) + (((VCC - temp2)*RL_1330)/temp2) / 2;

// Reading sensor GGS-2330
delay(150);
temp1 = analogRead(A5)*3.3/4095;
temp2 = analogRead(A6)*3.3/4095;
RS_2330 = (((VCC - temp1)*RL_2330)/temp1) + (((VCC - temp2)*RL_2330)/temp2) / 2;

// Reading sensor GGS-10330
delay(150);
temp1 = analogRead(A1)*3.3/4095;
temp2 = analogRead(A3)*3.3/4095;
RS_10330 = (((VCC - temp1)*RL_2330)/temp1) + (((VCC - temp2)*RL_2330)/temp2) / 2;

// Reading sensor TGS-2600
delay(150);
int sensor2600_1 = analogRead(A7);
float voltage2600_1 = sensor2600_1 * (3.3 / 4095);
float TGS2600_1 = (3.3/ voltage2600_1 - 1) * 4700;
delay(150);
int sensor2600_2 = analogRead(A8);
float voltage2600_2 = sensor2600_2 * (3.3 / 4095);
float TGS2600_2 = (3.3/ voltage2600_2 - 1) * 4700;
float TGS2600_AVG=(TGS2600_1+TGS2600_2)/2;

// Reading sensor TGS-822
delay(150);
int sensor822_1=analogRead(A9);
float voltage822_1=sensor822_1*(3.3/4095);
float TGS822_1=(3.3/voltage822_1 - 1)*4700;
delay(150);
int sensor822_2=analogRead(A10);
float voltage822_2=sensor822_2*(3.3/4095);
float TGS822_2=(3.3/voltage822_2 - 1)*4700;
float TGS822_AVG=(TGS822_1+TGS822_2)/2;

// Write all values in one string and that is passed to Python
delay(150);

Serial.print(time);
Serial.print(",");
Serial.print(CO2);
Serial.print(",");
Serial.print(TVOC);
Serial.print(",");
Serial.print(h,1);
Serial.print(",");
Serial.print(t,1);
Serial.print(",");
Serial.print(MQ_3);
Serial.print(",");
Serial.print(MQ_135);
Serial.print(",");
Serial.print(RS_1330);
Serial.print(",");
Serial.print(RS_2330);
Serial.print(",");
Serial.print(RS_10330);
Serial.print(",");
Serial.print(TGS2600_AVG);
Serial.print(",");
Serial.print(TGS822_AVG);
Serial.print("\n");

delay(10);
}

```