2022
9(2), 91-100

Journal of
MATERIALS NANOSCIENCE

Development of an automatic gas sampling cum injection unit and a graphical user interface of a feature extraction toolbox based on PYTHON for sensor array data analysis

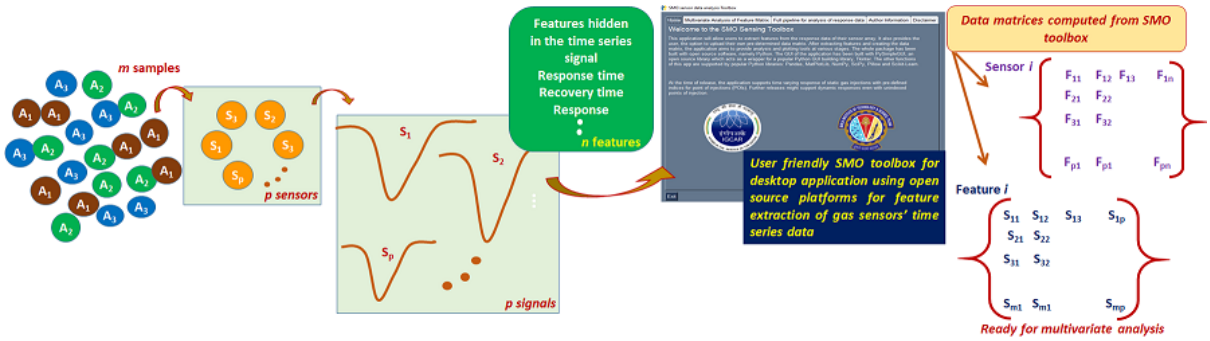
Debaayus Swain[#,1], Kunal Gupta[#,1], A. Sree Rama Murthy[#,2,3,*], V. Jayaraman[2,3]

[1]*Birla Institute of Technology and Science, Pilani, 333031 India.* [2]*Indira Gandhi Centre for Atomic Research, Kalpakkam, 603102 India.*
[3]*Homi Bhabha National Institute, Training School Complex, Anushaktinagar, Mumbai 400094, India*

Submitted on: 29-Oct-2021 Accepted and Published on: 15-Feb-2022

Article

ABSTRACT



An automatic gas sampling cum injection unit (AGSIU) equipped with a timer-controlled sampling pump with the bifurcated flow for carrier and sample with 3-way solenoid valves (SOVs) is designed and fabricated to minimize the errors due to manual samplings and its performance is evaluated. Different features like response time, recovery time, response (or sensitivity), response and recovery slopes maxima, and integral area for single or multiple sensors are processed using their respective time series data. To avoid the laborious manual methods, a software toolbox for the analysis and visualization of time-varying sensor signal data of an array of semiconducting metal oxide sensors (SMOs) is developed and its use is demonstrated. The proposed application has been built completely with open-source software using PYTHON and related platforms with a robust Graphical User Interface (GUI). The application consists of an efficient feature extraction algorithm to extract features from the user input of the point of injection (poi) to visualize the response data and the variation in the features of the data matrix.

Keywords: Gas sampling, Sensor array, Feature extraction, Multivariate analysis, Python, PySimpleGUI

INTRODUCTION

Gas sensors are becoming an integral part of any industrial safety systems or environmental pollution control agencies to ensure either the healthiness of the working personnel or safeguard the green earth at large. They are essential in monitoring the levels and presence of various gases which may be volatile or toxic and must stay under either their Threshold Limit Value (TLV[!]) or lower explosive limits (LEL[$]) whichever is applicable. Instead of a single sensor, array-based systems are being studied and developed to handle cross selectivity, cross-sensitivity and wide dynamic range that enhances the usage of semiconducting metal oxides (SMOs).[1–6] The initial training/calibration of the sensor (and sensor array) is of prime importance for qualitative and quantitative analysis. The sensor (array) gets trained from concurrent data obtained from repetitive sample injections. Typically, in gas sensor studies, the sample gas will be injected either into the static sensor chamber or into the dynamic carrier manually by gas-tight syringes. The latter is a typical procedure adopted in gas chromatographic studies. The

*Corresponding Author: A. Sree Rama Murthy, Novel Chemical Sensors Section, Materials Chemistry Division, MC&MFCG, Indira Gandhi Centre for Atomic Research, Kalpakkam – 603 102.
Tel: +91-44-27480500-24172
Email: asrm07@igcar.gov.in

[#]Contributed equally to this work.

URN:NBN:sciencein.jmns.2022.v9.283
ISSN: 2394-0867

http://.pubs.thesciencein.org/jmns

[!]Threshold Limit Value: The concentration in air to which it is believed that most workers can be exposed daily without an adverse effect (i.e., effectively, the threshold between safe and dangerous concentrations). The values were established (and are revised annually) by the ACGIH and are time-weighted concentrations (TWA) for a 7- or 8-h workday and 40-h workweek, and thus are related to chronic effects.
[$]Lower Explosive Limit: The minimum concentration of a gas, vapour, mist or dust in air at a given pressure and temperature that will propagate a flame when exposed to an efficient ignition source.

sensor signal for the same concentration of an analyte may vary because of parallax error (while reading the graduated syringe), diffusion rates or dilution that depends on the time delay between the sample suction from the gas sampler and injection into the sensor chamber (differs from person to person).[7] Burlachenko et al. presented an elaborated view on the status and prospects of sample handling for electronic nose systems (EN).[8] Literature on dedicated automatic gas samplers for sensor arrays is limited. Some earlier works on automated gas injections show limitations for dynamic gas sensing applications.[9] A dedicated gas sampling-cum-injection module is of great use for highly reproducible injections either for calibration or for field testing of a sensor (array) with remote sampling or for round-the-clock automated periodic sampling and analysis of working ambiance, etc.[10] The fore part of the current work describes the design, fabrication and performance of an automatic gas sampling cum injection unit. After the sampling and injection of gaseous analytes into the sensor (array) chamber, the responses will be collected and analysis will be performed. A typical n-type response was shown in figure 1. Apart from the common response feature ($\Delta R/R_b$),^ other features derived from the signal such as response time (t_{res}), recovery time (t_{rec}), maximum variations in response and recovery slopes $(dR_{res}/dt)_{max}$ and $(dR_{rec}/dt)_{max}$ and integral area under the curve (IA(t)) provide valuable information which have not been exploited so far.

For handling a large volume of data representing the above features, multivariate analysis is adopted as against the univariate analysis.[4,11] With the increase in the number of sensors in an array and for large-scale sample injections, manual estimation of these hidden features and subsequent processing of the data becomes tedious in addition to the ingress of human errors. Hence, the objective of this work is to evolve a feature extraction algorithm to deduce the aforementioned features of single or multiple sensors based on their respective responses and to develop pattern recognition systems.

The most important part of this application for feature extraction is the choice of programming language and PYTHON is selected for its easy syntax and the libraries support for data visualization, data extraction.[12] Pandas library of python is used extensively to manipulate and work with the response data as well as data matrices.[13]

Featured data matrices consisting of response, response time, recovery time, maximum variations in response and recovery slopes and integral area under the curve for the user-defined time interval for different sensors need to be created for further analysis.

Among these, response time is the time required by the signal to reach 90% of the full-scale output (R_{s90}) when the sensor is exposed to a full-scale concentration of the gas whereas the recovery time is defined as the time required by the signal to reach its normal state from the state where the output is 90% of the full-scale output (R_{b90}). Recovery and response slopes are defined as the respective minimum and maximum values of the gradient of the signal. The integral area represents the area under the response vs. time graph.

Developing a graphical user interface using the libraries on the Command Line Interface (CLI) is necessary for reducing the amount of setup work to get a pipeline up and running was felt necessary and useful for even a non-programmer. The proposed application takes inspiration from an existing MATLAB toolbox, DAV^{3}E,[14] which aims to perform feature extraction cum multivariate analysis from a temperature cycled operation of sensor data. One of the recent toolboxes, MVPANI[15] is a GUI-based toolkit that aims to provide multivariate pattern analysis for neuroimaging. There are other toolboxes viz., PyChem[16] and PyMVPA[17] which are Python-based multivariate analysis statistical toolboxes developed for the field of life sciences. Based on the extensive literature review, the authors did not find any available toolbox which provides the proposed functionality in a single interface and this prompted the authors to create a desktop application using PySimpleGUI, a GUI development Python package that transforms conventional GUI development frameworks like Tkinter and Qt into an easier coding interface. PySimpleGUI achieves a simpler syntax by hiding the boilerplate implementations and letting the developer work with high-level elements and develop their application in less time and with less code. The proposed application is capable of displaying user input response data and data matrices in a tabular format. It can also display visualizations and plots within the application. All the plots have been created using a popular Python plotting library called Matplotlib.

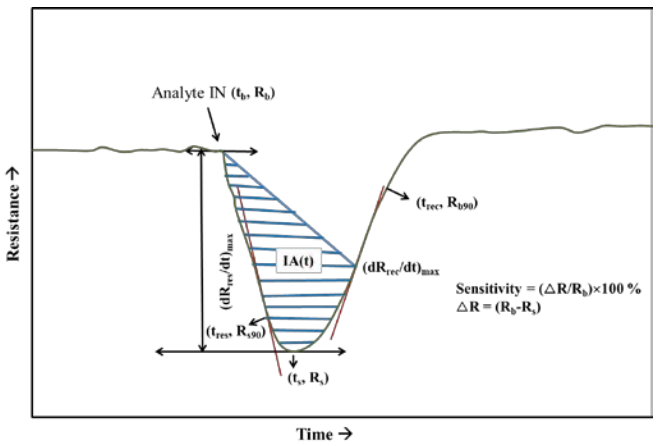


Figure 1 Typical n-type signal of a SMO sensor with different features indicated

Materials and Methods

Design and Fabrication of Automatic Gas Sampling Cum Injection Unit (AGSIU)

An automatic gas sampling cum injection unit was designed and fabricated as shown in the following schematic (figure 2). The carrier will be connected to the carrier port and it flows through the exhaust of a 3/2 way stainless steel solenoid valve (SOV) with kalrez seals (M/s Connexion Developments Ltd., UK) in a normally closed configuration to the sensor (array). The

^Normalized differential resistance, ($\Delta R/R_b$); $\Delta R = (R_b - R_s)$, where R_b and R_s are the resistances of baseline and minima of the signal at time t_b and t_s respectively.

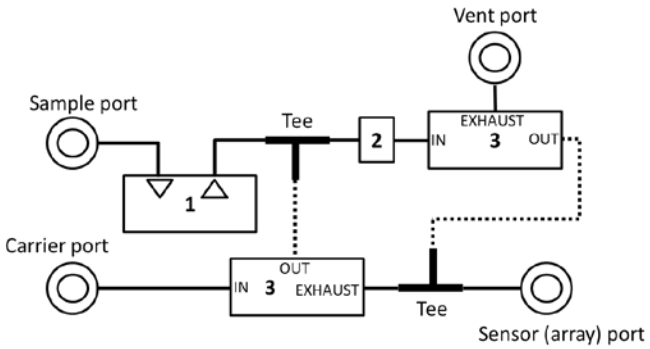


Figure 2 Schematic of automatic gas sampling cum injection unit: 1- KNF one way pump, 2 -Sampler volume, 3- 3/2 way solenoid valves

sample port is connected to the inlet of the pump (M/s KNF, #NMP830, Germany) and the outlet is connected to a sampler of known volume and to another 3/2 way solenoid valve in normally closed configuration. The pump and the solenoid valves are being controlled with digital timers (M/s Selec, India) which electrical line drawing is shown in figure 3. The cyclic timer T1 (#XT546) is for dwell time and pump 'on' time that was configured to OFF first and ON next mode. The timer T2 (#600-DT) is for solenoid valves 'on' duration and was configured as interval after break mode. The timer T3 (#XT546) is for total process and set as a reset timer in cyclic mode similar to T1. The details of terminal connections and configuration modes of the timers can be found in the operating instruction sheets of the respective models.[18]

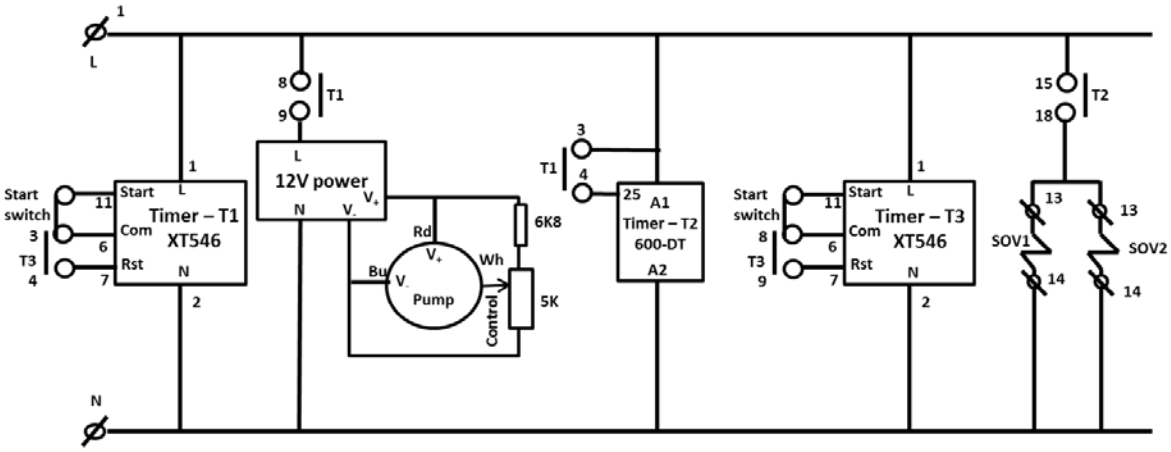


Figure 3 Electrical line diagram with the functionality of timers.T1 is for dwell time and to set pump 'on' time; T2 is for solenoid valves (SOV1 and SOV2) 'on' duration and T3 is for total process and reset timer.

Development of a desktop application

Backend

The most vital part of this application is the feature extraction algorithm. The backend part of this application is divided into three parts – loading the data from a *csv* (comma-separated values) file, applying the feature extraction algorithm and lastly creating the data matrices from the extracted features which could be used for further analysis. For the first part we use the Pandas library of Python to pre-process the file and load it into the program in a way by which the feature extraction algorithm could be applied. This library is mainly built for data analysis and makes pre-processing much easier.

For the core of the backend, i.e., the feature extraction algorithm, the basic definitions of all the six features mentioned in section 1 were considered. Additionally, another redundant feature, ratio of resistances[1] is also computed as it is handy at times for visualizing the sensor's performance.

The first step of calculation of 'response' requires a normalised baseline of the sensor signal before the point of injection (poi) and is estimated by considering an average of 30 data points towards the left side from the poi. The absolute difference between the normalized baseline and the actual resistance of the signal is divided by the normalized baseline resistance. Finally, the maximum value of this baseline normalised difference was computed in percentage as 'response' of that signal. Response slope and recovery slope are maximum and minimum values of the gradient of the signal respectively and obtained from calculating the difference of consecutive resistance values divided by the time gap between them.

For response time and recovery time calculation, the tip of the signal (R_s), defined as a point where the output of the signal reaches its minima is found out. Then, the data corresponding to a decrease in resistance upon introduction of an analyte is considered for the current application development which requires the index of the least resistance value of the signal. The minimum resistance (tip) was computed and its index was stored. After that, 90% of the difference of baseline resistance and the minimum resistance $(\Delta R)_{90}$ was computed. The R_{s90}, R_{b90} were obtained by subtraction and addition of $(\Delta R)_{90}$ from R_b respectively. The time corresponding to R_{s90}, R_{b90} were calculated from the interpolation of their nearest neighborhood data

[1] Ratio = R_s/R_b, where R_b and R_s are the resistances of baseline and minima of the signal at time t_b and t_s respectively.

timestamps respectively. The response time, the difference between the times corresponds to R_{s90} and the poi; the recovery time, the time difference between the tip of the signal and that corresponds to R_{b90} were computed.

For the integral area of the signal between the defined time gap was computed using trapezoidal formulation. Finally, the redundant ratio feature, (1-response) was calculated.

The last part of the backend was storing all these features into data matrices and for this, the Pandas library of Python was used. Two types of matrices were generated based on the choice of the user; (i) all features of the user-selected sensor or (ii) a particular selected feature of all the sensors.

Frontend

The development of a simple and straightforward method of feature extraction algorithm is preferred for sensor array data. This requires the knowledge of setting up a virtual environment and installing Python libraries on a command line, which can be overwhelming for non-programmers and can put a huge obstacle before the research community. Driven by the need, the authors developed a GUI based desktop application with a self-explanatory and intuitive layout which would aid the sensor data analysis at various programming competency levels.

The framework chosen for developing the GUI application was PySimpleGUI, a lightweight and low-code Python package which focuses on the development of high level elements of the GUI. PySimpleGUI aims at wrapping traditional frameworks in a simpler code interface which is intuitive for the developer leading to rapid development of the application. The authors chose the tkinter port of the package due to its wide support and a large number of demo scripts available within the package repository.

The structure of the application is set up in such a way that the user gets two options for the kind of data on which they wish to work. The current release of the application allows the user to either provide raw response data or a pre-processed feature matrix. The data provided can be in the *csv* format which is then pre-processed using the Pandas library. The pre-processing involves removal of unnecessary rows and other methods to get the data ready for feature extraction, i.e., the backend. The application provides the user the option for response data visualization and extraction of features from the provided sensor array data. If a user provides a feature matrix, the application processes the feature matrix for various types of visualization.

The strength of the application is extreme generalization and a highly intuitive layout for the user which has been demonstrated by the various dashboards available for visualization of the corresponding data. The user has been given the option to customize themes, fonts, sizes thereby allowing them to control every single element of the plot they wish to graph. The application also provides various options for saving the data matrices generated and the plots/figures in different formats. This allows the user to control the resolution, size and formats of the plots, thereby getting them handy for publication.

RESULTS AND DISCUSSION

PROCESS FLOW OF AGSIU

The process flow for the AGSIU is shown in figure 4. Once the run switch is 'ON', the cyclic timer T1 (configured to OFF first and ON next mode) will begin with dwell time followed by pump 'on' time.

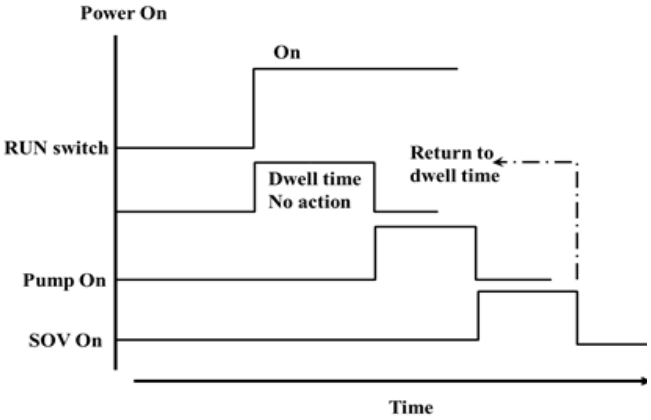


Figure 4 Schematic of process flow of Automatic Gas Sampling cum Injection unit

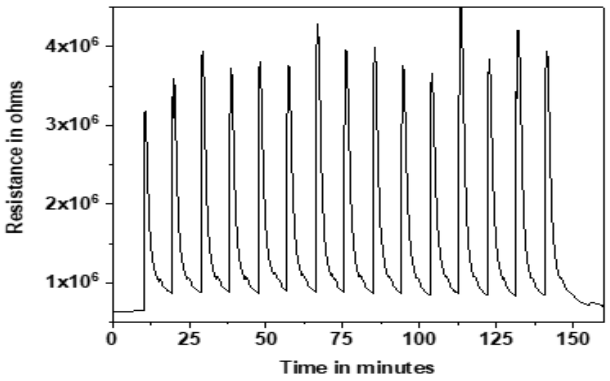


Figure 5 Typical time series data of an SMO showing sample injections in defined time intervals by AGSIU

The timer T2, configured for an interval after break mode, will take action at the end of pump 'on' time by opening the solenoid valves for a given duration. After that, the timer T3 returns to dwell time, and the cycle repeats.

During the dwell time, the carrier gas will flow through the sensor without entering into the sampler volume. After the dwell time, the pump will be triggered by the timer T1 for 'on' duration. The pump will suck the sample gas and fill the sampler volume by continuous venting. After the completion of pump 'on' duration, the timer T2 opens the solenoid valves so that the carrier gas is channeled through the sampler volume and to the sensor for a set duration. At the end of the timer, T2 solenoid valves will get closed and sampler volume is isolated by diversification of carrier flow to the initial path and the reset timer T3 restarts the cycle. A typical time-series data from an SMO sensor connected to the AGSIU with sampling and injection cycle repeated for every 10 minutes towards acetone vapours is shown in figure 5.

DEVELOPMENT OF BACKEND AND FRONTEND OF THE APPLICATION

Backend

To start, a *csv* file obtained from the user was used in the code as a dataframe 'df' and all the operations were performed on it. Different features of the sensors' signals were calculated as below:

i) Response

To calculate the normalized differential resistance, first the baseline resistance, a 30-point average to the left side of the point of injection (poi) was considered as below:

```
def base_line(poi,ser1):
    baseline=0
    for i in range(poi-30,poi):
        baseline+=ser1[i]
    baseline/=30
    return baseline
```

After the baseline calculation, the response was calculated as below:

```
def find_response(sensor,poi, df,next):
    ser1=df.iloc[:,sensor]
    response=0
    baseline=base_line(poi,ser1)
    sens=[]
     for i in range(poi,poi+next):
        sens.append(abs((ser1[i]-baseline)/baseline))
    response = max(sens)
    return response
```

Four parameters, viz., the sensor index, the poi, the data frame and the index for the end of a particular signal (to distinguish different signals) were considered in this calculation. After finding the absolute value of the ratio between the resistances, the maximum value is obtained.

ii) Response and Recovery Slopes

As response and recovery slopes are maximum and minimum values of the gradient of the signal, the gradient function, 'grad' was executed using the following code:

```
def grad(x,poi,next,gap):
    gradient=[]
    for i in range(poi+1,poi+next):
        gradient.append((x[i]-x[i-1])/gap)
    return gradient
```

The input parameters considered for grad calculation are the resistance series (x), poi, index of the end of the signal (next) and the time interval between two consecutive recordings (gap). After the calculation of gradient, the given series was passed to the response and recovery slope functions to find the minimum and maximum of it as below:

```
def recovery_slope(sensor,poi,df,next,gap):
    ser1=df.iloc[poi-1:poi+next-1,sensor]
    gradient = grad(ser1,poi,next,gap)
    recslope = max(gradient)
    return recslope
```

```
def response_slope(sensor,poi,df,next,gap):
    ser1=df.iloc[poi-1:poi+next-1,sensor]
    gradient = grad(ser1,poi,next,gap)
    resslope = min(gradient)
    return resslope
```

iii) Response and recovery times

The first step for calculating the response and recovery times is to find the tip of the signal, which is defined as the point where the output of the signal reaches its maximum peak/nadir point. In the current case as n-type response corresponds to a decrease in resistance hence, the index of the least resistance point of the signal (tipp) was determined from tip function.

```
def tip(ser,poi,next):
    tipp=ser[poi]
    index=poi
    for i in range(poi,poi+next):
        tipp=min(ser[i],tipp)
        if tipp == ser[i]:
            index = i
    return index
```

To calculate response time and recovery time, two parameters viz., delR and R90 were considered next to the tip calculation. The 90% value of the difference between baseline resistance and the minimum resistance (tip) is delR. R90 is defined as the difference of delR from the baseline resistance in case of response time and as the sum of delR and tip value for the recovery time calculations. The suitable interpolation resulted in the accurate calculation of response and recovery times as per the code presented in section S2 of the supplementary material.

iv) Integral area

The integral area is the area traversed by the signal in the user-defined interval of time from the poi that was computed using the trapezoidal rule. The area under the signal (CurveAr) is subtracted from the area of the trapezoid formed by connecting the poi and the user-defined value (LineAr) is subtracted to obtain the actual area of interest.

```
def integral_area(sensor,poi,df,points,gap):
    ser1 = df.iloc[poi:poi+points+1,sensor]
    LineAr = 0.5*(ser1[poi+points]+ser1[poi])*(points*gap)
   CurveAr = 0
for i in range(poi+1,poi+points+1):
```

```
    CurveAr += 0.5 * (ser1[i]-ser1[i-1])*gap
area = LineAr - CurveAr
return area
```

v) Ratio of resistances

A redundant feature, the ratio of a particular resistance to that of baseline is also considered for feature matrix generation.

vi) Creation of Data Matrices

All the features calculated above have to be integrated into a data matrix for analysis and visualization purposes. For multiple sensor data visualization, we need two types of data matrices. One will be a data matrix (type I) composed of all the features of different responses of a given sensor and the second one is a data matrix (type II) of particular feature entries of all the sensors of different responses. The corresponding codes are presented in section S1 of supplementary material. In the end, the user has provided an option of saving the data matrices of type I and type II in *csv* formats.

Frontend

The application has its attribute in ease of use and access with varying levels of comfort of programming. The interface is quite intuitive and self-explanatory. The authors have followed a tab-based layout for better dissemination of information. The home page (figure 6) provides the user two ways of moving forward with their data analysis, i.e. uploading a pre-extracted data matrix for multivariate analysis of feature matrix or the full pipeline for analysis of response data that extracts features and visualizes them from the raw response data uploaded by the user.

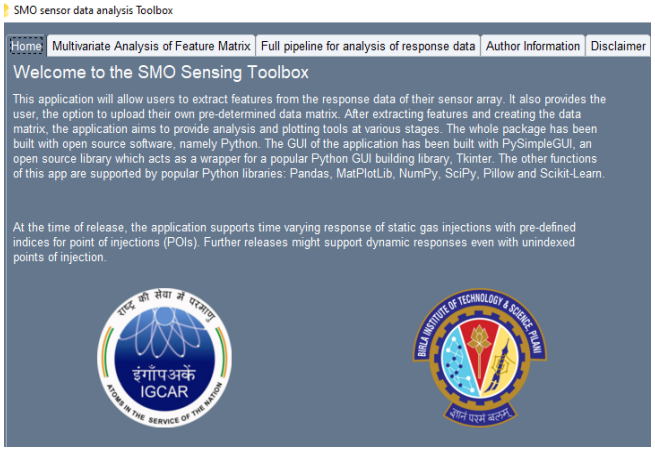


Figure 6 Screenshot of the landing page of the application with each tab explaining their respective functions

The Multivariate Analysis of Feature Matrix (figure S1) tab allows the user to upload a pre-extracted data matrix based on the provided description. Uploading a data matrix here allows the user to visualize the features as plots, allows the user to add columns based on their needs, and visualize those as well. A typical uploaded data view is shown in figure S2.

Uploading a data matrix takes the user to the screen which displays the uploaded data matrix with 2 user inputs:

1. Row number: This option is to allow the user to skip the first few rows in case there are additional comments or information rows before the real data matrix. This option is predominantly useful for response data. Since the headers displayed here are the true column headers, the user is expected to enter X.
2. Delimiter: The delimiter (aka separator) is a character separating the values in the table (like: , |, (space)). In the above

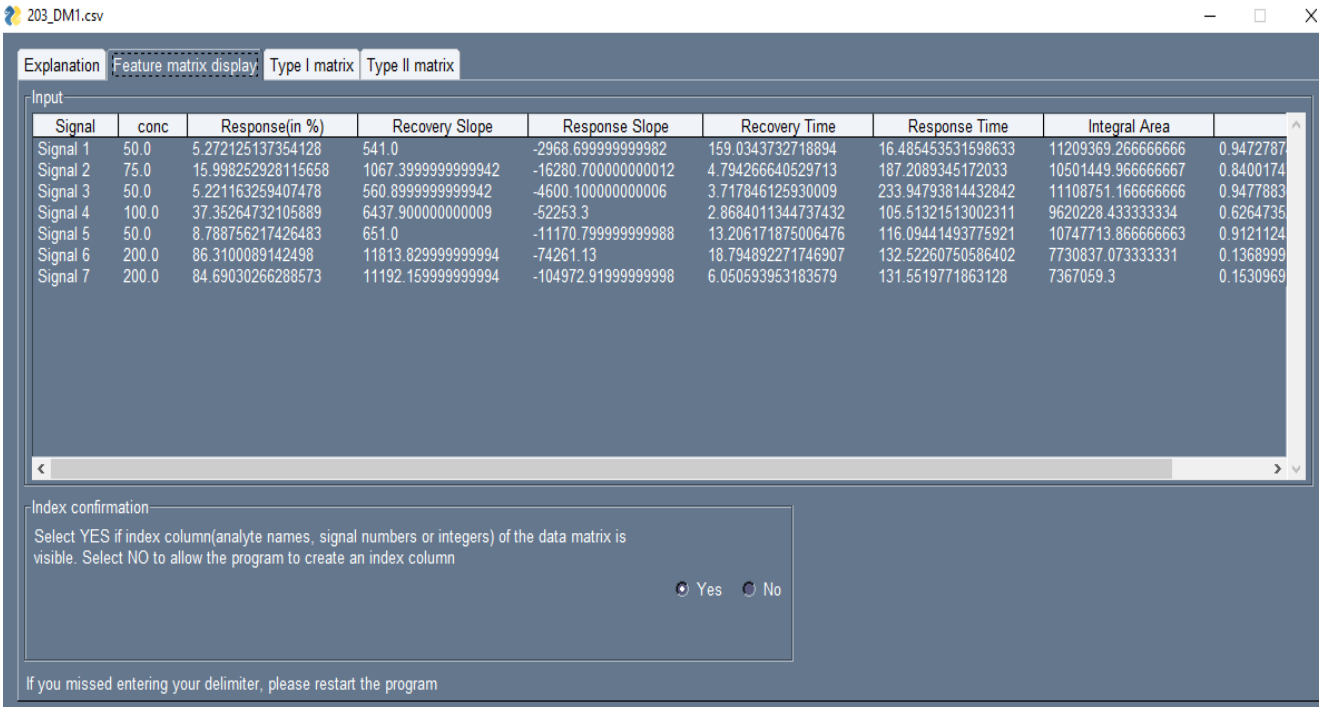


Figure 7 Screenshot of the feature matrix display tab with formatted data matrix

example, the user is expected to enter comma (,) as the delimiter in the input box provided, without any other extra text.

Pressing 'Submit' takes the user to the following page with the Explanation tab (figure S3) giving a short summary of the type of matrix and the features of the signals which can be visualized.

The Feature matrix display (figure 7) tab shows the user the formatted data matrix based on the user input (delimiter and skip rows) from the last page with an option for index confirmation.

The other two tabs are specific to the types of data matrices. The application expects the user to know the type of data matrix they have uploaded. In this example, a Type I matrix was uploaded, therefore the only parameter we have to enter is the name of that sensor (figure 8 (a)).

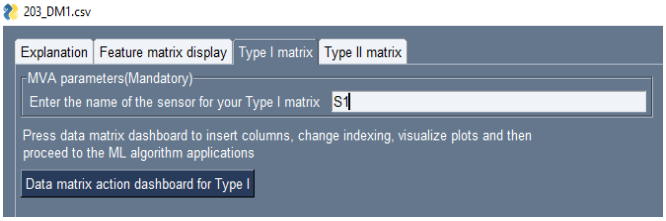


Figure 8 (a) Screenshot of type I data matrix selection with sensor S1

If the user uploads a Type II matrix, the application expects the user to input the corresponding feature (figure 8(b)).

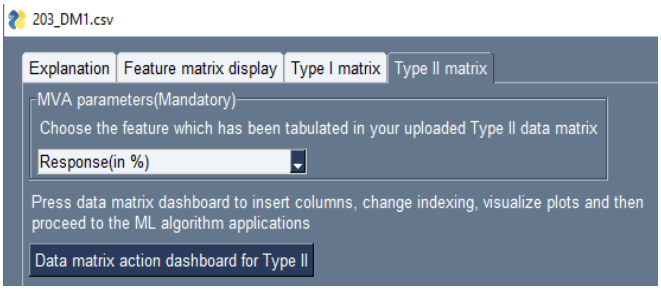


Figure 8(b) Screenshot of type II data matrix selection with response feature

The Data Matrix Dashboard backend is common for both the input options that give the user insight into the graphical representation of the data.

Full pipeline for analysis of response data tab allows the user for complete feature extraction of the time series data of multiple sensors. Figure 9 shows an example of the format of the response data expected from the user.

Uploading response data takes one to the above screen which displays the uploaded data matrix with 2 user inputs of formatting (figure S4) similar to that of the Multivariate Analysis of Feature Matrix tab and submission will forward to the next screen of inputs for visual inspection (figure S5) with three entry options viz., index column, timestamp column and column number of 1st sensor's data. The submission of which results in a response curve dashboard.

Figure 10 is the typical response curve based on the response data. The user has an option to customize the dashboard. The user

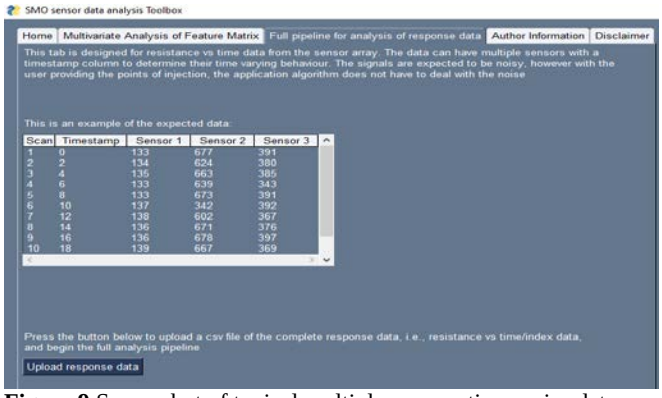


Figure 9 Screenshot of typical multiple sensors time series data uploaded by the user

can press the Save Plot button which takes them to the plot saving dashboard. A plethora of formats and more control over the quality of the plot were provided while saving. Pressing browse allows the user to choose a folder exactly like how the user was given the option to choose their *csv* file. The plot saving formats available are TIFF, PNG, JPEG, PDF and SVG (figure S6). This allows the user to customize quite a few aspects of the plot.

1. Theme: The matplotlib library offers various themes which the user can experiment with. A few examples include the ggplot theme which is a copy of the theme used for visualizations in the R package.
2. Width and Height: These parameters can be controlled and will be visible only when the user saves the plot. These changes don't reflect in the preview tab due to limitations on the changeable size of the canvas
3. Line width: This parameter controls the thickness of the lines in the response curve line plot.
4. Size of x/y-tick labels: These parameters control the font size of the ticks visible under the x and the y axes.
5. Number of x/y ticks: This parameter controls the maximum number of tick labels visible under the x or y-axis. Due to the number of entries, the default has been kept as 4, but the user can change it according to their preferences.
6. Font size of the legend: This parameter controls the font size of the legend which appears on the right side of the plot to avoid overlapping with the plot.

The "Choose data" tab of the plotting dashboard (figure 11) allows the user to customize the data-related aspects of the plot. The user can choose to plot a single column, or multiple columns based on their choice with an option for choosing the X-axis. The updates or changes of the parameters by the user can be visualised by pressing the "Preview Plot" button. The zoomed plotting tab is provided to view a particular section of the plot (figure S7).

The **Feature Extraction Dashboard button** of Choose data tab requests the user to enter the details of the points of injections (pois) and time interval/gap between successive data points, which is necessary for feature extraction of any type of matrix that the user wishes to extract (figure S8). It follows the user to compute matrices of choice.

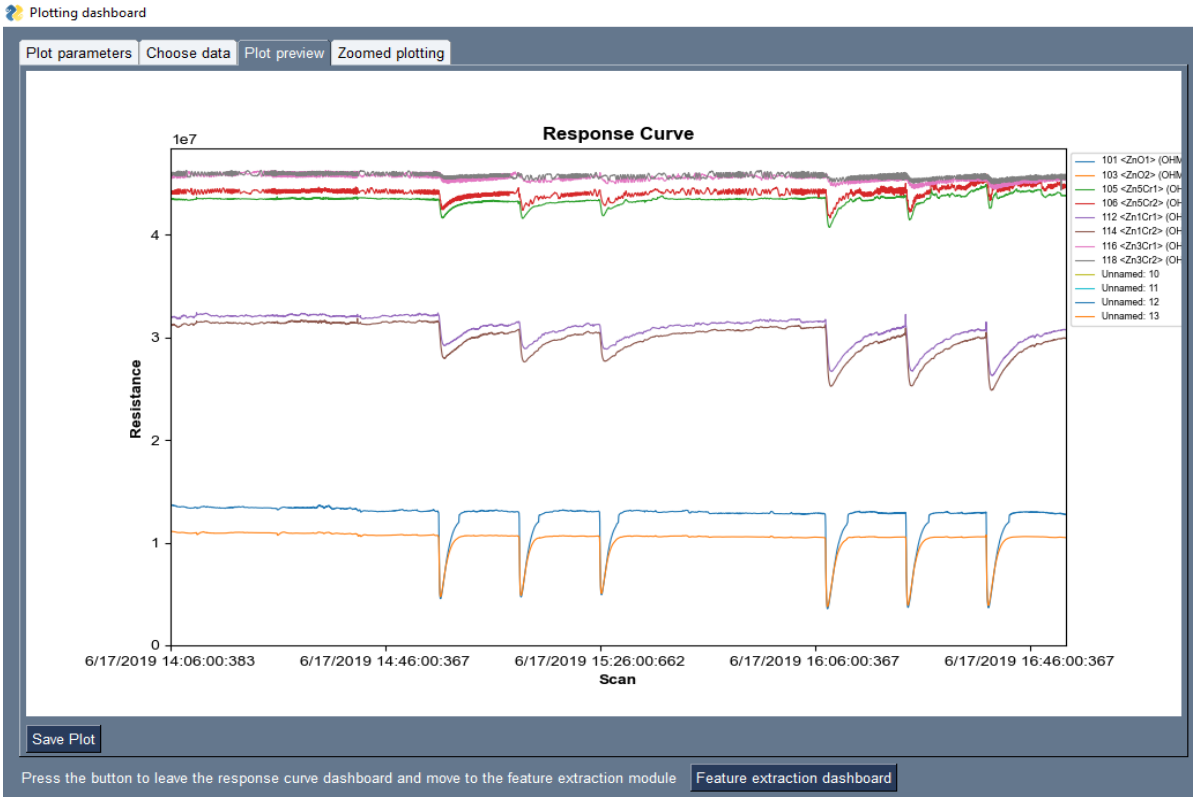


Figure 10 Typical response curve dashboard to visualize the responses from all the sensors

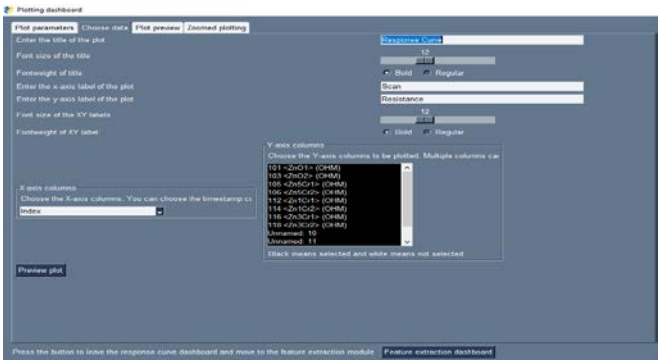


Figure 7 Screenshot providing the options for selection of X- axis and single or a group or all the sensors for feature extraction

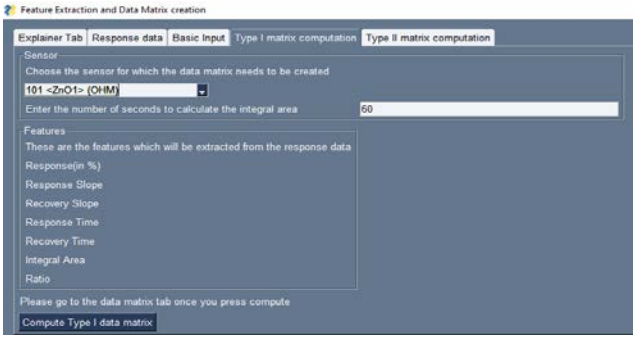


Figure 8 (a) Screenshot of computation of a type I data matrix with all the features of a particular sensor

Figure 12 (a) corresponds to the tab that allows the user to select a particular sensor for Type I matrix computation and then proceeds to extract all the features of multiple signals of a chosen sensor as per the algorithms presented in the backend. The Type II matrix computation in figure 12(b) allows the user to choose a particular feature wherein the application then proceeds to extract the chosen feature for all the sensors.

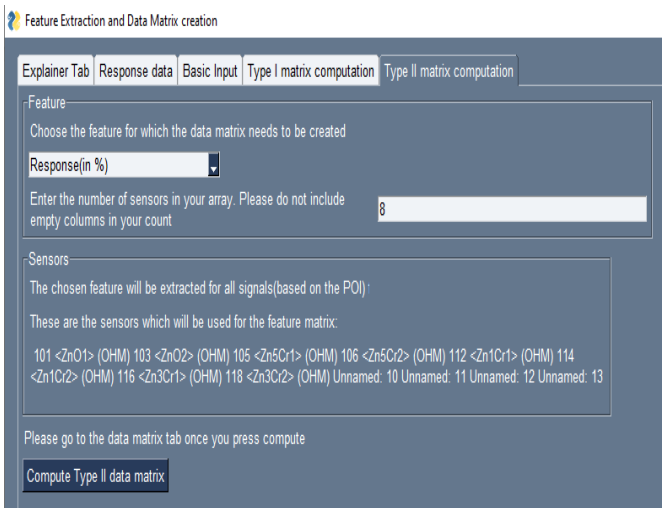


Figure 12(b) Screenshot of computation of a type II data matrix of a chosen feature of all the sensors

Once, the user has completed entering the details necessary for feature extraction, they led to the data matrix dashboard based on the chosen type. This dashboard provides options to append the

data matrix with choices of indices and concentration data (optional) to proceed for visualization with an option for saving the modified data matrix (figures S9 to S13).

Finally, the authors have provided an option for the user to save the data matrix in various formats (figure S14) which allow the user to continue their analysis using the secondary MVA pipeline instead of repeating the whole process again.

The 'Next' tab of the data matrix dashboard in figure S15 proceeds to visualization. The 'Reset' button provides an option to review the inputs and regenerate the modified data matrices by taking them back to the feature extraction dashboard.

Feature Plotting Dashboard provides different options to view the current data matrix (figure 13) and plotting. The bottom Data matrix dashboard button takes the user back to the data matrix dashboard for any modifications to be incorporated.

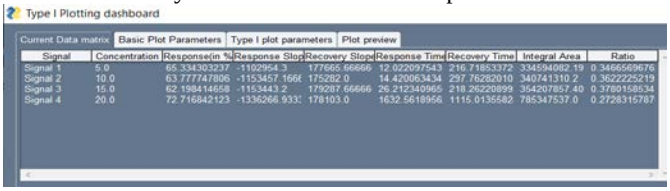


Figure 9 Screenshot of plotting dashboard with modified type I data matrix

Most of the parameters in the feature plotting dashboard are borrowed from the response curve dashboard with the exceptions of size and transparency value of the markers in the scatter plot. It is equipped with basic plot parameters (figure S16), type I and type II plot parameters corresponding to the initial choice of user (figures 14 (a) and (b)) and the plot preview tabs (figures 15(a) and (b)).

Figure 15(a) shows the resulting plot of Response(%) vs. Concentration. Similarly, every single feature can be tweaked according to the user's choice. Whereas, figure 15(b) displays the scatter plot of Response(%) (Chosen feature during computation) of four sensors vs. Concentration. The plot saving option shares common with the plotting dashboard described in earlier paragraphs. The design flow sheets explaining the decision paths multivariate analysis and full pipeline for analysis are presented in figures S17 and S18 of the supplementary material respectively.

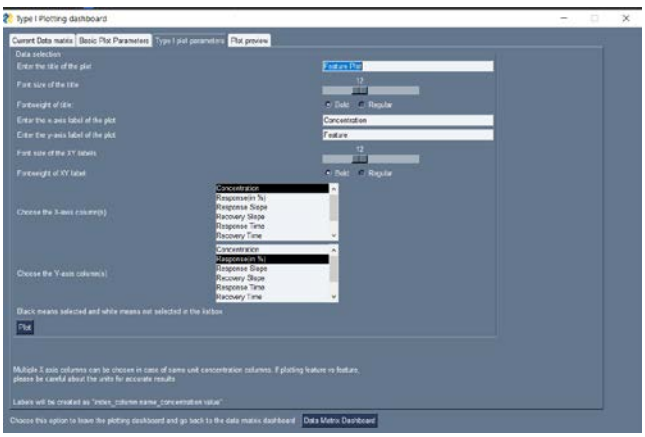


Figure 10 (a) Screenshot of Type I plot parameters tab to choose features of choice for visualization

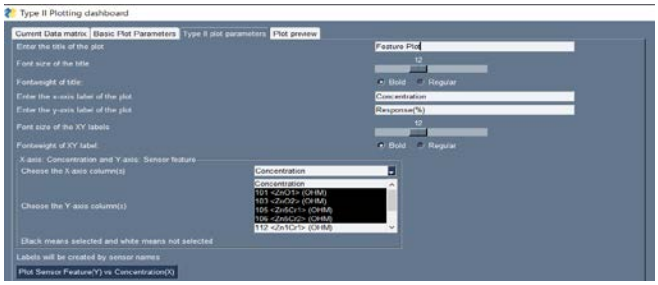


Figure 14(b) Screenshot of Type II plot parameters tab to choose features of choice for visualization

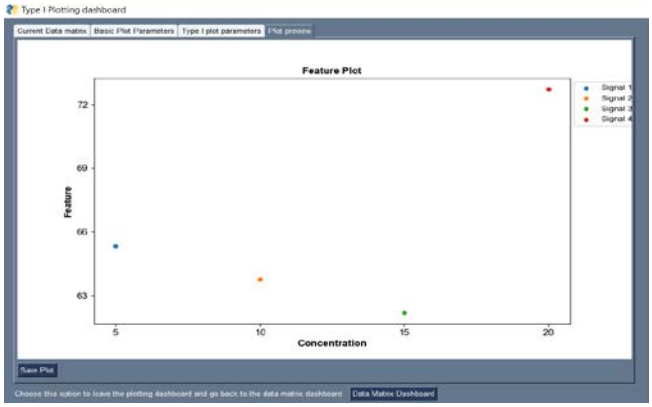


Figure 11 (a) Typical preview plot of Type I data matrix

Figure 15(b) Typical preview plot of Type II data matrix

CONCLUSIONS

Development and demonstration of a general-purpose software for the analysis of multisensory time series data using a custom-designed automatic gas sampling cum injection system is presented. The AGSIU has flexible time settings for the user, depending on the sensor's sensing and recovery aspects. The digital timers used in the instrumentation can be configured in different modes allowing the user to set a specific injection methodology. It presents further scope to equip the AGSIU with a pressure transducer to monitor the gas pressure inside the sample reservoir, an alarming lamp in case of any power fluctuations, and also with a rotameter to minimize the flow-related noise during sampling. A desktop application software built on open-source platforms with robust algorithms makes the application a very good tool for feature extraction from

multisensory time series data for gas sensor data analysis intended for the researchers in this field. Built-in with sophisticated libraries of Python, the application saves a good amount of time in computing different features from multiple sensors' responses and is provided with 'ML algorithm application dashboard' for further development of better analytical solutions. With pattern recognition systems incorporated, this feature extraction application certainly augment the electronic nose/tongue development. The application has scope to expand further to p-type responses also.

ACKNOWLEDGMENTS

Authors acknowledge Dr. K. I. Gnanasekar, Head, Novel Chemical Sensors Section for his valuable suggestions and support during the development of this application. We thank Dr. Rajesh Ganesan, Head, Materials Chemistry Division and Dr. N. Sivaraman, Director, Materials Chemistry and Metal Fuel Cycle Group, Indira Gandhi Centre for Atomic Research, Kalpakkam for their constant encouragement.

SUPPLEMENTARY INFORMATION

Supplementary information contains the python codes for feature extraction and data saving in required formats. Further, it also includes selective screenshots of the application. The details of the supplementary information were cited at appropriate places in the manuscript.

REFERENCES AND NOTES

1. J. Albert, Keith, S. Lewis, Nathan, L. Schauer, Caroline, et al. Cross-reactive chemical sensor arrays. *Chem. Rev.* **2000**, 100, 2595–2626.
2. N. Tayebi, V. Kollia, P.S. Singh. Metal-Oxide Sensor Array for Selective Gas Detection in Mixtures. *arXiv Prepr.* **2021**, 12990.
3. G.S. Kim, Y. Park, J. Shin, Y.G. Song, C.Y. Kang. Metal oxide nanorods-based sensor array for selective detection of biomarker gases. *Sensors* **2021**, 21 (5), 1–9.
4. A. Sree Rama Murthy, D. Pathak, G. Sharma, et al. Application of principal component analysis to gas sensing characteristics of Cr0.8Fe0.2NbO4 thick film array. *Anal. Chim. Acta* **2015**, 892, 175–182.
5. A. Sree Rama Murthy, K.I. Gnanasekar, V. Jayaraman, A.M. Umarji. Application of principal component analysis to the conductometric Cr1-xFexNbO4 (x = 0, 0.5, 1.0) thick films gas sensors. *ISPTS 2015 - 2nd Int. Symp. Phys. Technol. Sensors Dive Deep Into Sensors, Proc.* **2015**, 70–73.
6. J. Lee, Y. Jung, S.H. Sung, et al. High-performance gas sensor array for indoor air quality monitoring: The role of Au nanoparticles on WO3, SnO2, and NiO-based gas sensors. *J. Mater. Chem. A* **2021**, 9 (2), 1159–1167.
7. P.D. Virutkar, A.P. Mahajan, B.H. Meshram, S.B. Kondawar. Conductive polymer nanocomposite enzyme immobilized biosensor for pesticide detection. *J. Mater. Nanosci.* **2019**, 6 (1), 7–12.
8. J. Burlachenko, I. Kruglenko, B. Snopok, K. Persaud. Sample handling for electronic nose technology: State of the art and future trends. *TrAC - Trends Anal. Chem.* **2016**, 82, 222–236.
9. A. Supriyanto, R. Anggriani, S.W. Suciyati, et al. A Control System on the Syringe Pump Based on Arduino for Electrospinning Application. *J. Phys. Sci.* **2021**, 32 (1), 1–12.
10. S.B. Kondawar, A.M. More, H.J. Sharma, S.P. Dongre. Ag-SnO2/Polyaniline composite nanofibers for low operating temperature hydrogen gas sensor. *J. Mater. Nanosci.* **2017**, 4 (1), 13–18.
11. B. Zhang, P.X. Gao. Metal oxide nanoarrays for chemical sensing: A review of fabrication methods, sensing modes, and their inter-correlations. *Front. Mater.* **2019**, 6.
12. K. Bhosle, B. Ahirwadkar. Deep learning Convolutional Neural Network (CNN) for Cotton, Mulberry and Sugarcane Classification using Hyperspectral Remote Sensing Data. *J. Integr. Sci. Technol.* **2021**, 9 (2), 70–74.
13. S.B. Siledar, S. Tamane. Quadratic difference expansion based Reversible Watermarking for relational database. *J. Integr. Sci. Technol.* **2021**, 9 (2), 107–112.
14. M. Bastuck, T. Baur, A. Schütze. DAV^{3}E – a toolbox for multivariate sensor data evaluation (submitted). *Sensors* **2018**, 489–506.
15. Y. Peng, X. Zhang, Y. Li, et al. MVPANI: A Toolkit With Friendly Graphical User Interface for Multivariate Pattern Analysis of Neuroimaging Data. *Front. Neurosci.* **2020**, 14.
16. R.M. Jarvis, D. Broadhurst, H. Johnson, N.M. O'Boyle, R. Goodacre. PYCHEM: A multivariate analysis package for python. *Bioinformatics* **2006**, 22 (20), 2565–2566.
17. M. Hanke, Y.O. Halchenko, P.B. Sederberg, et al. PyMVPA: A python toolbox for multivariate pattern analysis of fMRI data. *Neuroinformatics* **2009**, 7 (1), 37–53.
18. R. Kumar, M.P. Chaudhary, M.A. Shah, K. Mahajan. A mathematical elucidation of separation membrane operations and technology of chemical and physical processes: An advanced review. *J. Integr. Sci. Technol.* **2020**, 8 (2), 57–69.