DEVELOPMENT OF AUTOMATED MOS BASED GAS SENSING DEVICE FOR SOUR SKIN DISEASE DETECTION IN ONIONS

by

THARUN KUMAR KONDURU

(Under the Direction of Changying Li and Glen C. Rains)

ABSTRACT

The overall objective of this study was to develop an automated gas sensing device to detect the presence of sour skin infected onions among healthy onions based on their volatile profiles. In the first study, a metal oxide semiconductor (MOS) based gas sensing device was developed. It is comprised of mechanical, electronic and software component. The mechanical component was designed by docking MOS sensors inside a Teflon chamber attached with a gas (or volatile) delivery system. The electronic component of the device consists of a microcontroller, memory chip, clock chip, LCD, MOS sensors and other peripheral components. Software was developed at microcontroller and personal computer level. Microcontroller level software instructs the device to function on a time-based schedule resulting in automatic operation of the device. PC-based software was designed with user interface to configure the sensor, collect data and process data for analysis. The device was characterized and tested using three different concentrations of four chemicals. In the second study, the sensor's potential was tested by analyzing its response to a group of healthy and infected onions. Best baseline corrected feature was selected for this application. Based on the

classification results, it was evident that the gas-sensing device was capable of identifying the presence of infected onions among healthy onions.

INDEX WORDS: enose, sensor, sour skin, onions, automation, customized, array, MOS

DEVELOPMENT OF AUTOMATED MOS BASED GAS SENSING DEVICE FOR SOUR SKIN DISEASE DETECTION IN ONIONS

by

THARUN KUMAR KONDURU

B.TECH, ANNA UNIVERSITY, INDIA, 2009

A thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2013

DEVELOPMENT OF AUTOMATED MOS BASED GAS SENSING DEVICE FOR SOUR SKIN DISEASE DETECTION IN ONIONS

by

THARUN KUMAR KONDURU

Major Professor:	Changying Li
	Glen C. Rains
Committee:	Ron D. Gitaitis
	Mark Haidekker

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2013

DEDICATION

I would like to dedicate this work to my family, mentors and all those hard-working farmers for their efforts.

ACKNOWLEDGEMENTS

I would like to extend my deepest gratitude to my advisors Dr. Changying Li and Dr. Glen Rains for giving me an opportunity to work on this project. Their constant encouragement, support, advice and technical guidance helped me throughout the course of this research project. It was solely because of them that I have a number of skills to be proud of since I joined the masters program. I am proud of myself for what I have become and am especially thankful to Dr Li for spending his valuable time in training me.

I would also like to thank Dr. Ron Gitaitis and Dr. Mark Haidekker for their valuable suggestions and support during the course of the project.

Special thanks to technician Mr. Dewayne Dales for offering his expertise in building sensor gas chamber and preliminary casing designs in the Tifton engineering department workshop and the extent of his skills are evident from his work.

I could not have accomplished this without the moral support and guidance offered by Mr. Weilin Wang, a colleague, a friend and a well wisher. Special thanks to Mason, Rui, Adnan and Chugunov for their support and constructive criticism which helped me improve the quality of the work.

None of this would have been possible without the love and unconditional support of my parents Mr. Bhuvaneswar Raju and Ms. Usha Raju. They never failed to offer financial support even through their most difficult times. Thanks to my mom for patience and encouragement.

A friend in need is always a friend indeed. Thanks to all my friends Krishna, Chaitanya, Manoj, Varun, Gaana, Matina, Onkar, Amp, Ishakh and all others in Athens for their support, care, concern that they have always been showing.

Thanks to the University of Georgia and the College of Engineering for giving me an opportunity to pursue my master's degree and conduct research. I am thankful to USDA NIFA Specialty Crop Research Initiative grant for providing funds to conduct this project.

Sincerely,

Konduru Tharun kumar

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER 1

INTRODUCTION

OVERVIEW

Onion (*Allium cepa*) is one of the most important vegetables, used in a broad range of dishes and cuisines. Approximately 20 pounds of onions are estimated to be consumed annually per person in the US (National Onion Association, 2011). Onions are cultivated in at least 175 countries around the world with China, India, United States, Turkey and Pakistan being the leading producers. Onions are harvested, cleaned and sorted prior to cold storage for a time period ranging from a few weeks to several months before processing. The storage room is maintained with cold and well-ventilated air to maintain marketable quality of the onions. In most cases, onions are stored for longer periods to meet the market demands during the off season. During storage, the pathogen in the onions develops over time and the disease spreads to the neighboring healthy onions, resulting in the major portion of onions becoming spoiled or non-consumable (Vikram et al., 2005).

MAJOR POST-HARVEST DISEASE

The post-harvest disease in onions usually matures or develops during storage. Even though the onions are visually inspected during post-harvest handling, the pathogen initially originates in the field during harvesting and the infected onion becomes non-consumable while in storage. The stored onions need to be constantly monitored for the presence of infection, so that infected onions can be removed. The presence of sour skin infection, caused by the bacteria *Burkholderia cepacia* in the

onion during storage can threaten the marketability of the onions. Sour skin disease in onions can be identified based on various symptoms, including pale yellow to light brown decay, softening of neck region and in advanced stages onion scales slipping off during handling. Sour skin disease is seen in various *Allium* species, although it is mainly known to be a disease of onions (Schwartz and Mohan, 2008). The disease can be prevented to a major extent by following control measures which includes harvesting the crop after proper maturation and drying after harvesting. Additionally a clean supply of irrigation water and avoiding recycled water can prevent the pathogen from spreading the disease to healthy onion crops. These measures can not eliminate the possibility of sour skin infection in onions. Hence, various innovative methods are being investigated to address this situation. There is an urgent need for a non-destructive method to detect the presence of sour skin disease in onions. A wide range of non- destructive techniques such as X-ray imaging, hyper-spectral imaging are being studied. These methods are primarily used to sort and remove bad onions prior to storage. This study focuses on designing a completely automated stand alone gas sensor array capable of identifying the presence of sour skin infected onions among healthy onions during storage.

VOLATILES RELEASED BY HEALTHY AND INFECTED ONIONS

Onions release various types of volatiles of which a major group belongs to sulfur and aliphatic classes. The types of volatiles released by control, botrytis and sour skin infected onions have been clearly studied using GC-MS and documented (Li et al., 2011). Principal component analysis (PCA) demonstrated that Cyranose 320 responded differently to botrytis neck rot, sour skin and healthy onions. Cyranose 320 is a commercially available portable e-nose which was not designed for operating in storage

rooms as it required repeated configuration every time the device was initiated. The accuracy and sensitivity of the sensors depended on the amount and type of volatiles exposed to the e-nose sensors. The total amount of volatile compounds detected in the pathogen inoculated bulbs was one or two orders of magnitude higher than those in the control. This study served as the foundation for designing a customized gas sensor array to identify the difference in the type and concentration of volatiles released by the healthy and sour skin infected onions. A customized sensing device had not been developed for sour skin disease detection in onions based on the previously documented studies.

ELECTRONIC NOSE TECHNOLOGY

Various types of gas sensing materials are being investigated for use in various gas analysis applications. Some of them include conducting polymers (CP), surface acoustic wave (SAW) sensors and metal oxide semiconductors (MOS); each has their own strengths and weaknesses. The MOS sensors are commercially available with an affordable price and are resistant to sulfur poisoning compared to other types of sensors (Rouseff and Cadwallader, 2001b). The disadvantage of MOS sensors is its high power consumption.

METAL OXIDE SEMICONDUCTORS AND ITS OPERATING PRINCIPLE

MOS sensors have been used extensively in e-nose devices and are commercially available (Taylor and Schultz, 1996). Typical materials include oxides of tin, zinc, titanium, tungsten and iridium, doped with a noble metal catalyst such as platinum or palladium (Nagle and Gutierrez-Osuna, 1998). These metal oxides are intrinsically n-type semiconductors (Pearce, 2003) responding to reducible gases at temperatures ranging from 200º - 500º C increasing their conductivity, when exposed to

target gases. FIGARO® and FIS® industries offer different types of sensors that are sensitive to a wide class of volatiles.

The operating principle of MOS sensors involves a sensing material typically made of the metal oxide SnO_2 and various other metal oxides such as ZnO, Fe_2O_3 and WO_3. When the crystal is heated up to certain high temperatures, oxygen is adsorbed on the metal oxide crystal surface with a negative charge. The donor electrons are transferred to the adsorbed oxygen at the sensor surface (Barbara, 1996) resulting in a decrease in conductivity, thereby resulting in low sensor response measured in mV (milliVolts). When sample gas odor comes in contact with the sensing material, the concentration of oxygen molecules decreases leading to an increase in conductivity. The oxide and the operating temperature affect the sensitivity and selectivity of the sensor. Small amounts of impurities and catalytic metal additives such as palladium (Pd) or platinum (Pt) are known to affect the conductivity of the sensor. The selectivity of the sensor is changed by the doping of the catalytic metal of the sensor or by coating with a thin catalytic metal film on the sensor surface.

SENSOR DEVELOPMENT

Sensing the volatiles released by healthy and infected onions required fabricating a gas sensor array with gas delivery system. Commercially available gas sensors could have solved the issue, but are very expensive ranging from $7,000 - $80,000 and in most cases unaffordable for onion growers. Other than the price of these devices, they are usually non portable and require human presence for operation. Hence, there is a need for developing a customized gas sensor array that can be designed specifically based on the requirements. Requirements include the capability to function automatically in a timely manner and the ability to differentiate healthy and sour skin

infected onion. Hence, monitoring the health of onions in a room automatically reduces the cost and time required to inspect onions in cold storage. Additionally, cost of the device can be considered an important factor. There are various challenges to be faced and conditions to be met for fabricating the device. The first step involved fabricating mechanical components of the device to facilitate delivery of the sample odor effectively to the MOS sensors. For this, it was crucial for the MOS sensors to be securely enclosed inside a customized chamber to allow maximum exposure of the sample gas and to minimize them from being exposed to moisture content in the atmosphere. Above all, it was important for the chamber material to be odorless, non reactive to chemicals, highly tolerant to temperature and most importantly, sample gas under investigation not adhere to the material. The failure to meet any of the above mentioned conditions would affect the entire study as the MOS sensors might pick up the stray odor signals within the chamber. In the second step, electrical circuit was designed to obtain the response signal from each MOS sensor and perform basic computation automatically. For every sensor response, response features were extracted and computed to understand the nature of the volatile profile. The results have to be saved in a non volatile on-board memory chip which can be accessed whenever required. The device should also have the facility to transmit the data to the PC for real time data acquisition for detailed analysis. Along with these functions, the mechanical components had to be controlled in a timely fashion. Thirdly the microcontroller needed to be programmed to provide instructions automatically and a graphical user interface for the user to configure the device, collect and process data conveniently.

RESEARCH OBJECTIVES

- Construct a chamber containing the gas sensor array and functional gas delivery system.
- Design an electronic circuit to automate the device and to connect with the computer to transfer data and commands.
- Develop software programs at both the microcontroller and computer levels to perform data sampling, storage and processing.
- Characterize the sensor and test the device ability to classify four chemicals and to discriminate different concentrations of each chemical.
- Validate the gas sensor array by detecting the presence of sour skin infected onions among healthy onions.

STRUCTURE OF THE THESIS

The entire study is presented in three additional chapters. Chapter 2 discusses the various components and methods used for fabricating individual components (mechanical, electrical and software) of the device. These individual components are integrated together to function as one single device. The device is characterized and its potential to discriminate between sample odors tested with three concentrations of acetone, ethanol, acetonitrile and ethyl acetate. The results are presented along with statistical analysis. Chapter 3 discusses the successful discrimination of healthy and sour skin infected onions and the best baseline corrected feature selected. The progress of the disease within the onion samples is observed with the increase in the number of dai (days after inoculation). The MOS sensor that contributed the most in discriminating

the healthy and diseased onions is identified and classification accuracy studied and discussed. Final chapter summarizes the results of this study.

REFERENCES

Barbara, H. (1996). "Solid state, resistive gas sensors." Handbook of Chemical and Biological Sensors, Taylor & Francis.

Li, C., Schmidt, N. E., and Gitaitis, R. (2011). "Detection of onion postharvest diseases by analyses of headspace volatiles using a gas sensor array and GC-MS." *LWT - Food Science & Technology*, 44(4), 1019-1025.

Nagle, H. T., and Gutierrez-Osuna, R. (1998). "The how and why of electronic noses." *IEEE Spectrum*, 35(9), 22.

National Onion Association. (2011). About onions: bulb onion production, http://www.onions-usa.org/all-about-onions/consumption.

Pearce, T. C. (2003). *Handbook of machine olfaction : electronic nose technology / [edited by] T.C. Pearce ... [et al.]*, Weinheim [Germany] : Wiley-VCH, c2003.

Rouseff, R. L., and Cadwallader, K. R. (2001a). *Headspace analysis of foods and flavors : theory and practice / edited by Russell L. Rouseff and Keith R. Cadwallader*, New York : Kluwer Academic/Plenum Publishers, 2001.

Schwartz, H. F., and Mohan, S. K. (2008). *Compendium Of Onion and Garlic Diseases And Pests*, Amer Phytopathological Society.

Taylor, R. F., and Schultz, J. S. (1996). *Handbook of chemical and biological sensors / edited by Richard F. Taylor, Jerome S. Schultz*, Bristol ; Philadelphia : Institute of Physics Pub., c1996.

Vikram, A., Hamzehzarghani, H., and Kushalappa, A. C. (2005). "Volatile metabolites from the headspace of onion bulbs inoculated with postharvest pathogens as a tool for disease discrimination." *Canadian Journal of Plant Pathology*, 27(2), 194-203.

CHAPTER 2

DEVELOPMENT AND CHARACTERIZATION OF METAL OXIDE SEMICONDUCTOR BASED CUSTOMIZED GAS SENSOR ARRAY[1]

[1] Konduru, T., Li, C. and Rains, G.C. To be submitted to *Sensors and Actuators B.*

ABSTRACT

The shelf life of onions could be reduced due to post-harvest diseases such as sour skin caused by the bacterium *Burkholderia cepacia*. Recent studies have indicated the difference in the type and concentrations of volatiles released by the healthy and sour skin infected onions. To detect rotten onions in storage, we developed a gas sensor array, which consisted of seven MOS (Metal Oxide Semiconductor) sensors that are sensitive to wide range of organic volatile compounds. These MOS sensors were enclosed in a specially designed Teflon chamber, which was equipped with gas delivery system to deliver volatiles from the onion samples to the chamber. The device was automated using a specially designed electronic circuit comprised of a microcontroller (18F4550), memory chip, a trickle-charge real time clock chip and several other electronic components. Instructions were programmed into the microcontroller to function based on the user preferences. These preferences are communicated with the on board microcontroller through a graphical user interface program developed using LabVIEW. Three features (area under the curve, slope and relative response) were extracted from the sensor response. The developed gas sensor array was characterized and the discrimination potential was tested by exposing it to different concentrations (20 µl, 100 µl and 400 µl) of acetone (ketone), acetonitrile (nitrile), ethyl acetate (ester) and ethanol (alcohol). The gas sensor array could differentiate the four chemicals of same concentrations and different concentrations within the chemical with significant difference based on MANOVA tests. An additional experiment was conducted with 0.5 and 5.0µL concentrations of methlypropyl-di-sulfide and 2-nonanone. The MOS sensors Tgs 826 and SB AQ8 responded the most to 0.5 µL of methylpropyl-di-sulfide and 2-nonanone. Extracted features (Area, slope and

relative response) discriminated 0.5 µL and 5 µL of methlypropyl-di-sulfide. Area and relative response features extracted for 0.5 µL and 5 µL of 2-nonanone was significantly different.

INTRODUCTION

Onions are the third largest fresh vegetable crop in the United States (National Onion association, 2011). It is cultivated in at least 175 countries around the world of which China, India, United States, Turkey and Pakistan are among the leading producers (Vikram et al., 2005). Once harvested, onions are kept in cold storage facilitated by a cold, dry and well-ventilated atmosphere for a period ranging from a few weeks to six months before processing. During storage, they are highly prone to a number of post-harvest diseases, most due to infection and mechanical damages during transportation and harvest. This study focuses on the post-harvest disease called sour skin caused by the bacteria *Burkholderia cepacia* which reduces the shelf-life of onions significantly. It has been a general tradition to hand grade and cull the diseased onions before storing them. These methods fail to completely remove the infected onions from the lot due to lack of visual symptoms when onions are still in the early stages of disease development. As a result, certain onions with latent diseases could end up in storage and serve as the vehicle for disease propagation. Differences in the type and concentration of volatiles released by healthy and infected onions were studied by Li et al. (2011). There is a need in the onion industry for a sensitive and inexpensive gas sensing device that can differentiate between healthy and diseased onions.

Devices based on electronic nose (E-Nose) technology were built to detect different odors and there are various kinds of commercially available E-noses for different applications (Pearce, 2003). For instance, commercial E-noses were used to

predict the pleasantness of 22 essential oil odorant mixtures based on the response obtained from the MOSES II e-nose by comparing it with the inputs provided by human subjects (Haddad et al., 2010). Fruity odors were detected and classified successfully (Kea-Tiong et al., 2010) as well. Also, based on the concentration of air contaminants such as NO_2 and CO indoor air quality was monitored with customized e-nose (Zampolli et al., 2004). In the field of microbiology, E-nose technology showed potential to discriminate bacterial colonies in a suspension (Green et al., 2011). Commercial e-nose technology was also used to detect and classify post-harvest diseases in blueberry fruits (Li et al., 2010), and to differentiate various types of *Allium* based on their headspace volatiles and pungency (Abbey et al., 2001b). However, the commercial e-noses are expensive and in some cases unaffordable. The cost of these devices is roughly in the range of $7000 - $ 80,000. It would be cost prohibitive if multiple e-noses are needed to be deployed in certain settings, such as large onion storage rooms.

The development of a customized e-nose requires a combination of volatile sampling, volatile sensing, electronic control and data recording. There have been several documented efforts to develop a customized E-nose in the laboratory. For instance, a group used MOS sensors and a data acquisition card (USB 6008) to record sensor responses to classify the change in thai herbs of northern Thailand (Klinbumrung et al., 2009). E-nose prototypes were fabricated to classify beverages (Mamat et al., 2011) and to perform spoilage classification of beef (Panigrahi et al., 2006). MOS based gas sensor array was customized for discriminating coffee aromas, essential oils and volatile compounds with different functional groups (Aishima, 1991). However, these

prototypes either do not have on-board computation capabilities or are not specific to onion post-harvest disease detection. To fill this gap, we developed a customized gas sensor array.

Using MOS sensors, an automated electronic circuit board and user friendly software interface capable of sour skin disease detection in onions was developed. Due to its low cost, we can deploy multiple such devices in a large onion storage room.

The device and the methodology employed in this project could potentially be modified to match the requirements needed for detecting post-harvest diseases in other specialty crops.

The objectives of this study were to:

- Construct a mechanical chamber containing gas sensor array attached with fully functional gas delivery system.
- Design an electronic circuit to automate the device and to connect with the computer to transfer data and commands.
- Develop software programs at both the microcontroller and computer levels to perform the data sampling, storage and processing.
- Characterize the sensor and test the device ability to classify four chemicals and to identify different concentrations of each chemical.

OVERVIEW OF THE SENSING SYSTEM

The gas sensor array system was designed to specifically detect volatile profiles of the odor released by onions. The system consisted of mechanical, electronic and software program components. The mechanical component had a gas delivery

system to transport the volatiles from the headspace to the MOS sensors docked inside a chamber. The electronic component included a circuit board with a microcontroller, memory chip, time keeping chip and other peripheral devices. The software programs were at two levels. At the microcontroller level, the program was developed using PIC BASIC PRO (PBP, MicroEngineering Labs, Colorado Springs, CO, USA) to calculate multiple features namely area under the curve, slope and relative response based on each MOS sensor response. At the computer level, a graphical user interface (GUI) was developed using LabVIEW v8.2 (National Instruments, Austin, TX, USA) to configure the sensor, download and process the data. The construction, characterization and testing of the device are explained in the following sections.

MECHANICAL DESIGN

The mechanical components of the device were designed to facilitate delivery of the odor or fresh air to the chamber containing MOS sensors and also to effectively remove chemical odor from previous experiments. Other considerations included accommodating additional MOS sensors for future studies, low cost and convenient assembly/disassembly.

Polytetrafluoroethylene (PTFE or trademarked name ‘Teflon’) material was used to constructing the body of the chamber. Teflon material had several advantages pertaining to the application of this device. It was inert to practically all chemicals and withstands temperature variations. The use of Teflon material also eliminated the possibility of sample odor contamination. A square-shaped Teflon block was cut into three 132 x 132 x 19.8 mm blocks placed on top of each other after customizing the middle and lower block. Holes were bored into the lower block to mount the MOS sensors. Each hole varied in size and shape based on the size of the individual MOS

sensor. In the middle block, a circular hole of diameter 112 mm was cut through the block to provide a headspace for the sensors. The arrangement of the three blocks is shown in Figure 2.1. Any change in MOS sensor size or shape could be easily addressed by replacing the lower block with a suitable design at low cost.

The MOS sensors were connected to their respective sockets and were plugged into the lower block with their sensing element facing towards the headspace between the upper and lower block, thereby, exposing the odors to the sensors. A simple 'leak test' was performed using a regulated air pressure hose to identify the leaks in the chamber. One end of the chamber was connected to an air pressure hose, which was regulated by a pressure gauge (range 0 to 5 psi) (Norgren Inc, Littleton, CO, USA)**.** The other end of the chamber was connected to an air flow meter (ADM 2000, Agilent technologies, Santa Clara, CA, USA), which measured the flow rate of the air. This method helped determine the location of the leaks and the flow rate of the air at certain pressures in the chamber. A minor leak was identified between the blocks and the problem was addressed by placing a rubber fiber gasket wrapped with Teflon tape between the blocks.

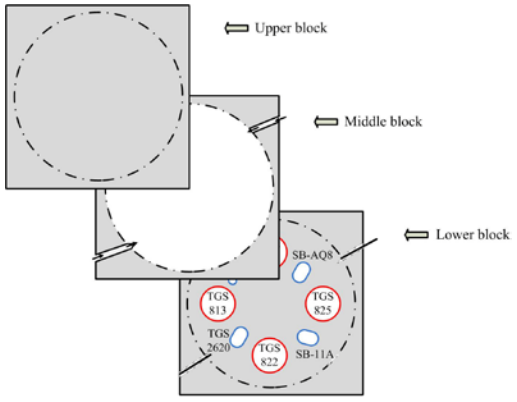

Figure 2.1: Schematic diagram of the arrangement of three Teflon blocks used for making the chamber.

Selection of pump and valve

The pump and the valve were selected for their oil-free performance, ensuring uncontaminated sample odor or clean airflow to the chamber. Additionally, the pump was selected for its capability to deliver the sample odor or clean air through the sealed chamber, compact size, pneumatic performance, high durability and pump speed which could be controlled using the microcontroller. A valve was selected for its type of actuation (i.e., three way diverter), fluid type (air), port size and flow capacity. Both the pump and valve components were operated at 12V DC. Clean air was supplied to the chamber by providing 12V to the valve and cutting off the voltage supply forces the valve back to its default position, thereby allowing the sample odor to flow into the chamber.

Gas samples were drawn inside the chamber using a pump (NMP 830 KNDC B BLDC motor, KNF Neuberger, INC, New Jersey, USA), and a 3-way valve (225T031, NResearch Incorporated, New Jersey, USA) to select one of two pneumatic circuits. One was provided with a filter containing charcoal and desiccant (removes humidity and odor) to deliver fresh air to clean the chamber. Teflon tubing (Outer diameter = 4.76 mm) was used to deliver the gas to the chamber. The schematic diagram of the mechanical design is provided in Figure 2.2. The mechanical components namely pump, valve, Teflon chamber and a filter are shown in Figure 2.3

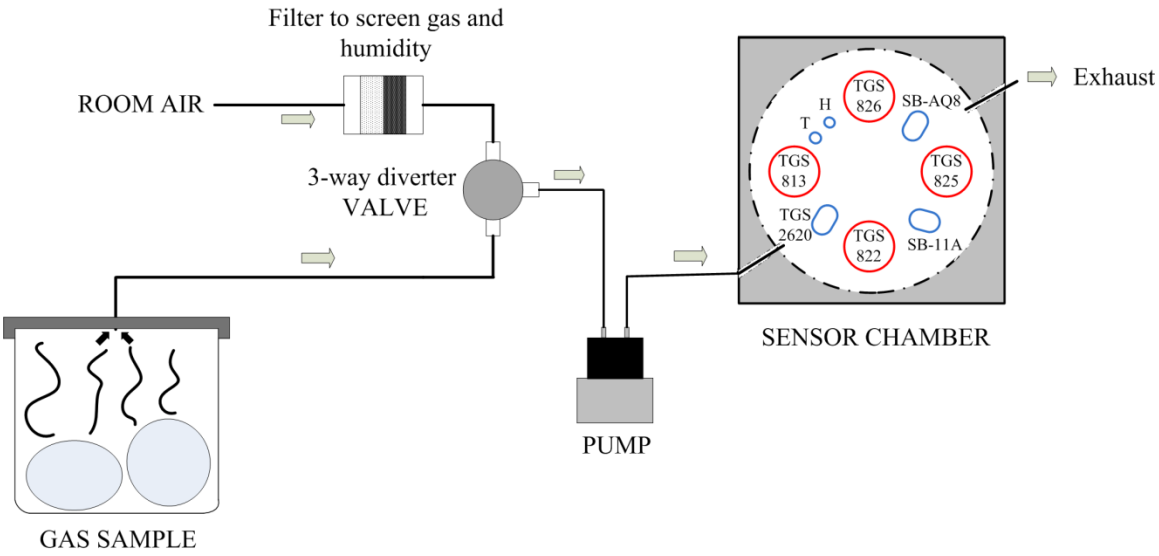

Figure 2.2: Schematic diagram of the mechanical design of the customized gas sensor array.

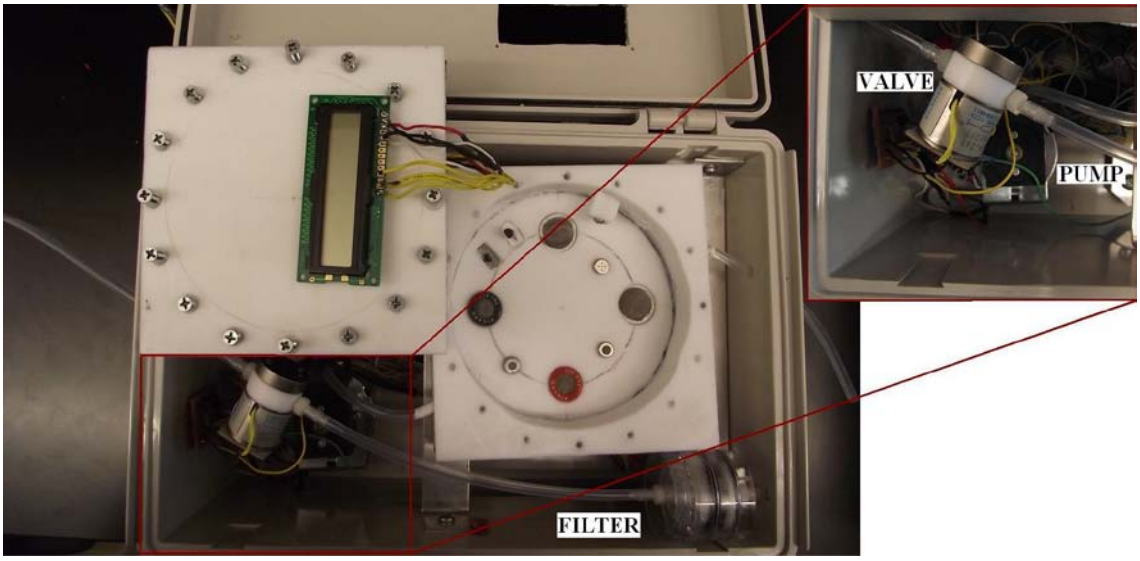

Figure 2.3: Top view of the mechanical component of the sensing device.

The chamber was in one of the three states baseline purging, sampling and purging at any given time. 'Baseline purging' involved injecting clean filtered air into the chamber preparing the device for sampling phase. 'Sampling phase' involved injecting sample odor for investigation into the chamber and the purging phase involved

removing the sample odor from the chamber by purging clean air. This step always followed sampling phase.

The chamber design was tested for the individual MOS sensor's sensitivity and response time at different pump speeds and effect on response time and sensitivity of MOS sensor due to their positioning inside the chamber.

ELECTRONIC DESIGN

The circuit was designed with an on-board microcontroller PIC 18F4550 (Microchip Technology Inc., Arizona, USA), which performed basic computation and controlled the functions of the mechanical components in a timely fashion. It was a 40-pin microcontroller consisting of 13 A/D (Analog/Digital) conversion channels connected to 5V power supply and a 20MHz external ceramic oscillator. It served as the main component of the electronic design. Individual sensor's response was received by microcontroller in terms of voltage (mV) readings, which was subjected to 10-bit A/D conversion. Features were extracted from real time data to compute area under the curve, relative sensor response and slope. The electronic components and their functions are explained in detail below.

Metal Oxide Semiconductor (MOS) sensors

The selection of MOS sensors was primarily based on chemical specificity. Secondary parameters included size, cost and power consumption. MOS sensors were not commercially available to detect the specific volatiles released by onions when they are healthy or diseased. Hence, multiple MOS sensors sensitive to wide range of organic volatile compounds (VOCs) were selected for this application. The volatiles released by the infected onions primarily contained compounds belonging to sulfur and aliphatic

groups (Li et al., 2011). Aliphatic groups contain compounds such as methane, ethane, propane and butane. MOS sensors sensitive to sulfur, aliphatic compounds and other related compounds were chosen. Table 2.1 lists the MOS sensors used in the gas sensor array.

Table 2.1: List of MOS sensors used to build gas sensor array and the gases they are sensitive to. Sensors 1-5 were purchased from FIGARO Inc. and sensors 6 and 7 are purchased from FIS Inc.

S. No	MOS sensor	Specificity
1	TGS 813	Methane, Ethane, Propane
2	TGS 822	Organic solvent vapors
3	TGS 825	Low conc. of H_2S
4	TGS 826	Ammonia
5	TGS 2620	Alcohol and organic solvent vapors
6	SB-11A	Hydrocarbon
7	SB-AQ8	VOCs

Microcontroller

The microcontroller was selected based on its high computing capability with the minimum availability of 9 A/D conversion pins. The pulse width modulation (PWM) pin in the microcontroller was used to control the pump speed. The data collected from the MOS sensors was transmitted to the PC through RS232 communications. Hence, microcontroller selection criteria also included I^2C (Inter-Integrated Circuit) and USART (Universal Synchronous/Asynchronous Receiver/Transmitter) communications capability. The microcontroller collected the analog response from the sensors and converted them to digital values. The microcontroller chip had 40 pins with 35 I/O pins of which 13 can performed A/D conversion. Fifteen data points were collected every second with a 20MHz ceramic

oscillator functioning with the microcontroller.

Memory chip

The memory chip was selected primarily based on the memory space requirements and communication protocol. I^2C 24AA1025 (Microchip Technology Inc., Arizona, USA). The memory chip selected was a Serial Electrically Erasable PROM (SEEPROM), which operated with 5V supply voltage. It had a data storage capacity of 1024 kbit with a typical writing speed of 3ms/ page. With each dataset consuming 24 bytes of space, a total of 5,461 datasets were stored. The data stored in the memory was non-volatile, hence the data was preserved in the chip even if the power supply was cut off. Three features were extracted from the digital values and along with the time stamp were serially written on the memory chip.

Other peripheral components

The microcontroller, memory chip and the MOS sensors served as the important components of this study. However, additional components were essential to perform various tasks. DS1302 trickle-charge real time clock chip (Maxim Integrated, San Jose, CA, USA) was used to obtain time information. It was connected to a 32.76 KHz crystal oscillator and powered by a 3V coin battery, which allowed continuous update of time even when the device was powered off. MAX232IN (Texas Instruments, Dallas, Texas, USA) was used to serially communicate with the PC and transfer data from the device to the PC. A parallel LCD was used to display the status of the device.

The valve was connected to the MOSFET (metal oxide semiconductor field effect transistor), which functioned as a switch controlling the supply of voltage based on the timed instructions coded in the microcontroller. The valve was used to select the

passage of fresh air or sample odor. The speed of the pump was operated using a pulse width modulation (PWM) pin available in the microcontroller. The pump was switched off by providing a ‘zero’ through the PWM pin. Two pump speed selections were provided in which one selection operated the pump at half of its speed and the other at its full speed described as ‘Low’ and ‘High’ respectively. The entire circuit required three different power supplies. The major portion of the circuit was designed to operate with a power supply of 5V except for the valve and the pump, which required voltage supply of 12V. An external coin battery (3V) was provided to the DS1302 facilitating the chip to function constantly even when the device was switched off. This eliminated the need for updating the time details every time the device was switched ‘on’. The pin connections for the above described components are shown in Figure 2.4. Top view of the circuit board designed for the gas sensing device is shown in Figure 2.5.

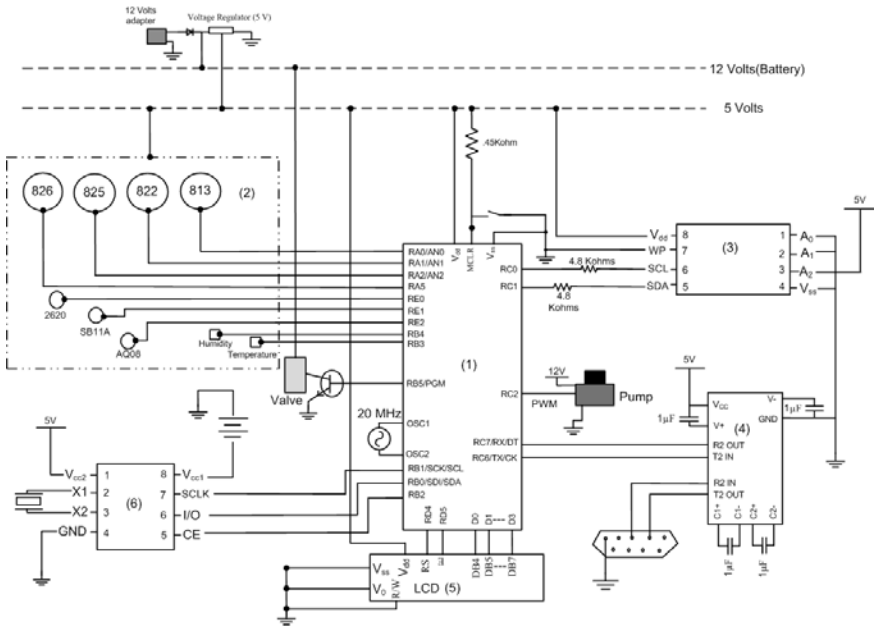

Figure 2.4: Electronic circuit designed for automation of the customized gas sensor array.

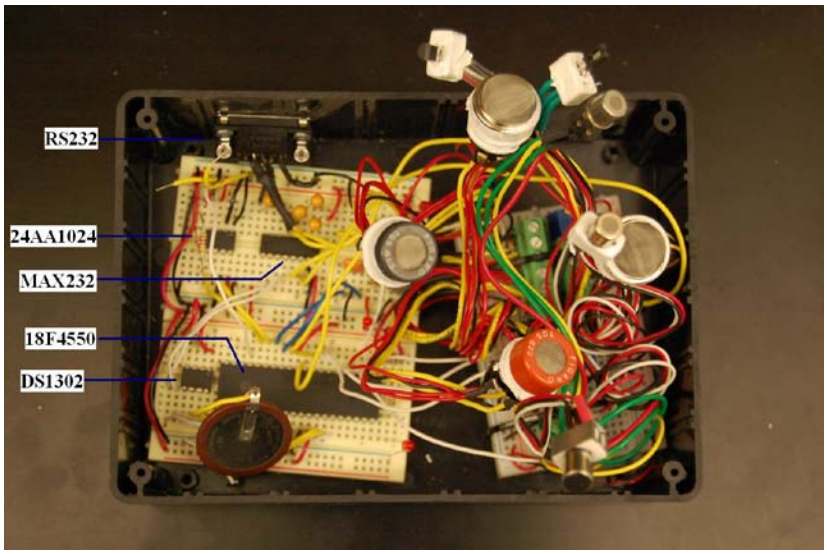

Figure 2.5: Top view of the electronic circuit designed for gas sensing device

The functions of the above described components are summarized below:

The operation of pump and valve in timely fashion facilitated the collection of each MOS sensor's response to the sample odor under investigation. The data was computed to obtain three features namely slope, area and relative response. The computed data was saved in an I^2C24AA 1025 memory chip. The DS1302 clock chip (Maxim Integrated, San Jose, CA, USA) was used to calculate the time allocated by the user for each phase. The circuit has a MAX232IN chip (Texas Instruments, Dallas, Texas, USA) and its primary function was to facilitate convenient communication between the user and the sensor. The user could download the data saved in the memory chip on the sensor board or display the data in real time.

SOFTWARE DESIGN

Two types of software programs were designed for the device. Firstly, microcontroller program was developed using PIC BASIC PRO software. Secondly, user instructions were provided to the microcontroller using the graphical user interface (GUI) software program designed on the PC using LabVIEW 8.2. This GUI was used to

configure the sensor, download and process the data.

Microcontroller program

The microcontroller program was coded in a PICBASIC PRO (PBP, MicroEngineering Labs, Colorado Springs, CO, USA) compiler and was programmed via an in-circuit serial programmer (ICSP programmer, Micro Engineering labs, CO, USA). The device was programmed to receive instructions from the user to configure it. The user could provide instructions to collect data, download data, erase data, auto run, force sampling and force purging. Each of the instructions required inputting user preferences and was programmed to operate the sensor conveniently. For data collection, the user provided the number of seconds required to perform baseline purge, sampling and purging cycles, pump speed and feature selection. The device executed the instructions and automatically collected data, calculated features and saved the data in the memory chip along with the time information. When the device was connected to the PC, real-time data was collected in a text (.txt) format. The results were saved after every data collection in the memory chip and retrieved using the “Download data” command into the specified folder in text format. After downloading the data, the user could completely delete the data in the memory chip to save more data using the “Erase data” option. In the event of collecting data for a long time intervals, an “Autorun” function was provided. This function retrieved the default configuration provided by the user, hence the user need not configure the sensor every time. This function instructed the sensor similar to the “Collect data” function. “Forced sampling” and “Forced purging” selection forced the microcontroller to skip directly to the sampling cycle and purging cycle respectively.

Each user selection was directed to a sub-routine for executing a specific function. After receiving the user configuration, the microcontroller was programmed to perform A/D conversions for the seven MOS sensors, temperature and humidity sensors. The data was then used to extract parameters namely minimum value, maximum value, data values at every 1 second interval and the time at which the maximum value was attained. These parameters were used to calculate the three features (area under the curve, relative response and slope). User configuration and feature calculations are explained in the following sections. The calculated features along with the time stamp were saved in the memory chip automatically. The time stamp was obtained from a trickle-charge real time clock chip (DS1302) and the time details were programmed to be retrieved from this chip. The device was designed and programmed to communicate with the PC through RS- 232 cable connection. This was achieved by setting up the USART (Universal Synchronous/Asynchronous Receiver/ Transmitter).

Sensor configuration

The microcontroller operated completely based on the instructions provided by the user. When the device was switched ‘on’ for the first time, the user inputs the time information (mm/dd/yyyy). String 1 in Figure 2.6 shows the format in which the time information was provided. The microcontroller was programmed to calculate the time automatically even when the device was not operating or completely switched ‘off’. The time information was sent to the microcontroller by the user only when the battery connected to DS1302 was replaced or disconnected.

The device operated based on multiple user-provided instructions in the form of

a string of numerical values. The microcontroller was coded to respond to each numerical value in a specific way. The string contained 12 digits of which 11 digits were allocated to input command information and the 12th digit for command type. Figure 2.7 shows the string containing spaces to input information for each function. In the string, BP represented baseline purge. In this space the user was allowed to enter three digit numbers representing the number of seconds desired to perform baseline purge (Eg: 050 seconds will perform baseline purge for 50 seconds). Similarly, the time information was provided for sampling and purging (P). Following this information, a slot was provided to enter the code for feature selection (1 = area under the curve, 2 = relative response, 3 = slope and 4 = all the features). The codes for feature selection are presented in Figure 2.7.

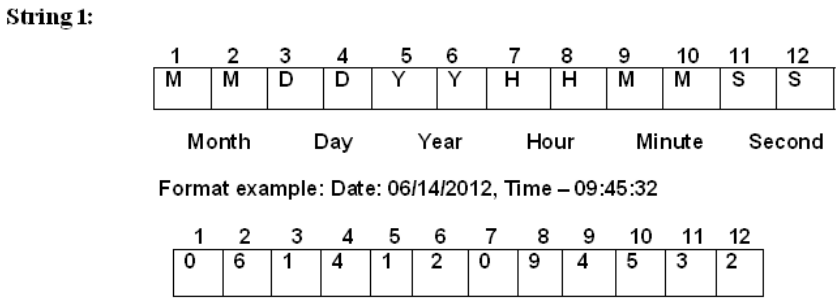

Figure 2.6: String of numberical values inputted by the user to set the time in the microcontroller

String 2:

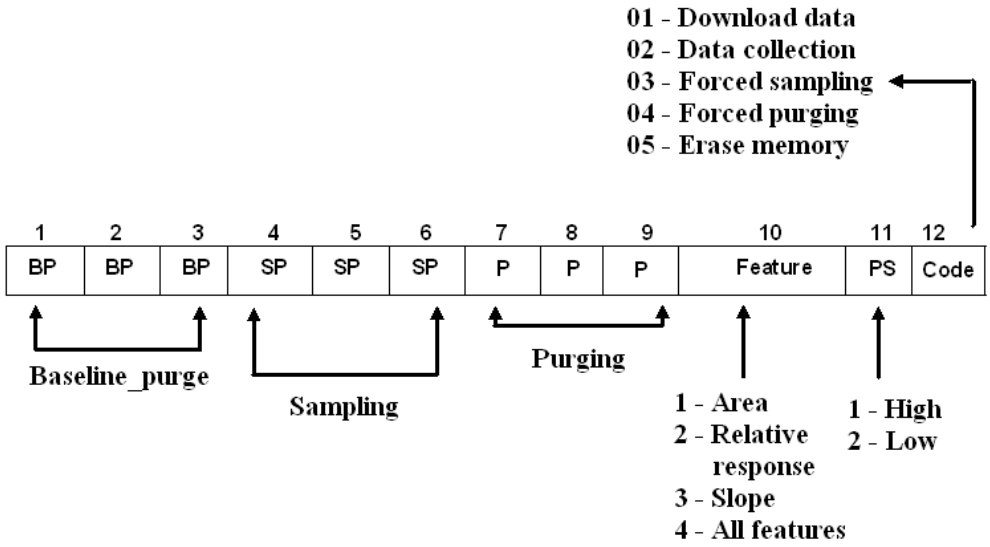

Figure 2.7: User preferences/selections in the form of string sent to the microcontroller as an instruction.

Pump speed was represented as PS in the Figure 2.7 and two selections 'High speed' and 'Low speed' were provided for user flexibility. Final slot was allocated to provide command type. Each numerical value instructed the microcontroller to use the command information to perform a specific operation. The codes for each instruction were provided within the braces. Download data to the PC (1), data collection (2), forced sampling (3), forced purging (4) and erase memory (5). These instructions were programmed into LabVIEW software interface program and were provided by clicking the corresponding buttons on the GUI software. If the user was required to input the string of numbers, the LCD or PC displayed the following message for the time string in Figure 2.6 "Input date and time yy/mm/dd-hh/mm/ss", whereas for the configuration string shown in Figure 2.7 "Enter desired configuration". The structure of the microcontroller program is illustrated in Figure 2.8.

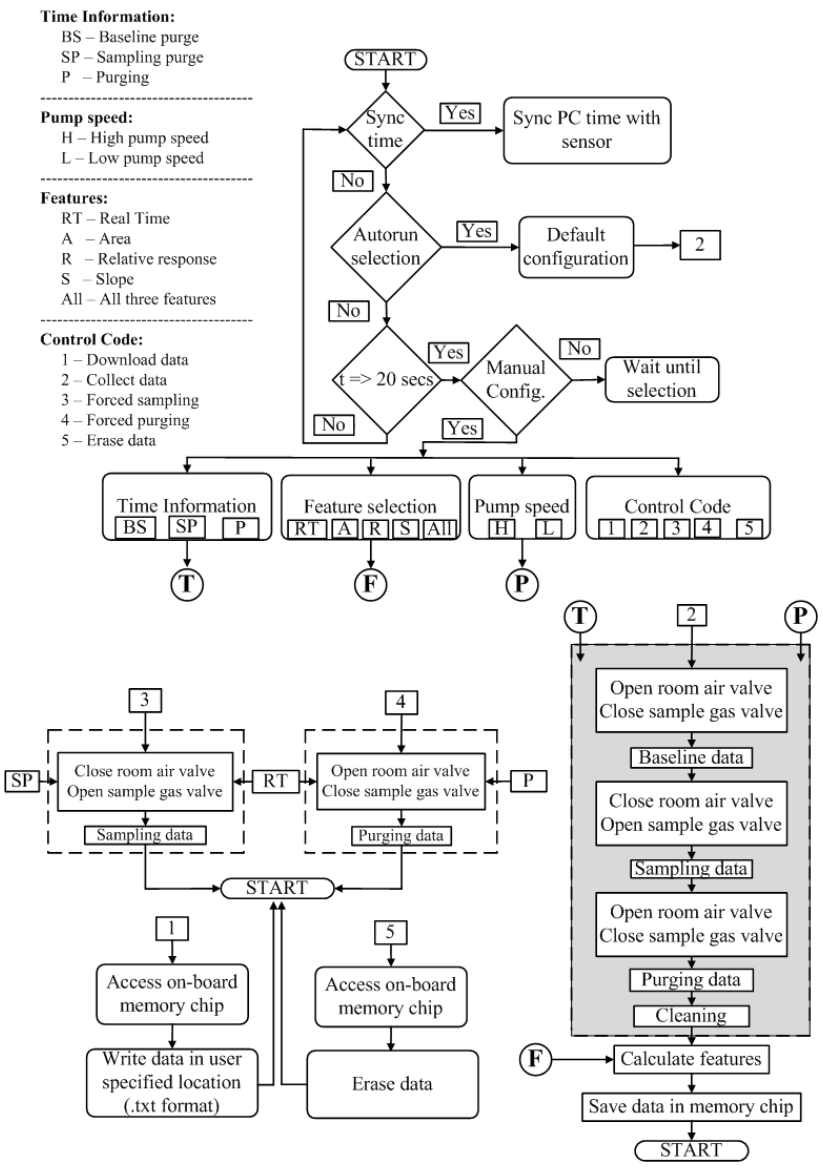

Figure 2.8: Flowchart of the microcontroller program used for device automation.

Feature calculation

The device measured the volatile profile of the sample gas under investigation. It was programmed to derive multiple information such as maximum value, minimum value, time taken to reach the maximum value and data point at every one second interval for each MOS sensor as shown in Figure 2.9. This information was used to calculate three different features namely area under the curve, slope and the relative

response. The feature calculation is discussed below in detail.

Area under the curve

Area under the curve was calculated using trapezoidal rule or trapezium rule shown in equation (1). The data recorded at every one second interval was used in equation 2. This feature was used to obtain the magnitude of the response due to the lack of computational power in the microcontroller, equation 1 was not used directly. An approximate numerical solution to equation 1 was used for microcontroller programming and is presented in equation 2.

$$\int_a^b f(t)dt \qquad \text{…. [1]}$$

$$A(i) = [(a(i) + b(i)) / 2] * dt \qquad \text{…. [2]}$$

where,

a = response of sensor ‘i’ at time‘t’,

b = response of sensor ‘i’ at time‘t+1’,

i = number of sensors (i = 1, 2...7),

A (i) = area under the curve for the MOS sensor ‘i’,

dt = change in time, which was usually set to 1 second interval.

Slope

Slope feature was computed using the minimum, maximum and the time taken to reach the maximum value. Equation 3 was used to obtain the feature:

$$Slope(i) = (Max(i) - Min(i)) / t \qquad \text{…. [3]}$$

where,

Slope(i) = Slope obtained from the sensor ‘i’.

Max(i) = Maximum value obtained from the MOS sensor ‘i’

Min(i) = Minimum value obtained from the MOS sensor ‘i’

t = Time taken to reach the maximum response in the sampling phase.

Relative response

Relative response feature was computed using the maximum and minimum values extracted from each MOS sensor’s response.

$$R(i) = [Max(i) - Min(i)] / Min(i) \quad \text{.... [4]}$$

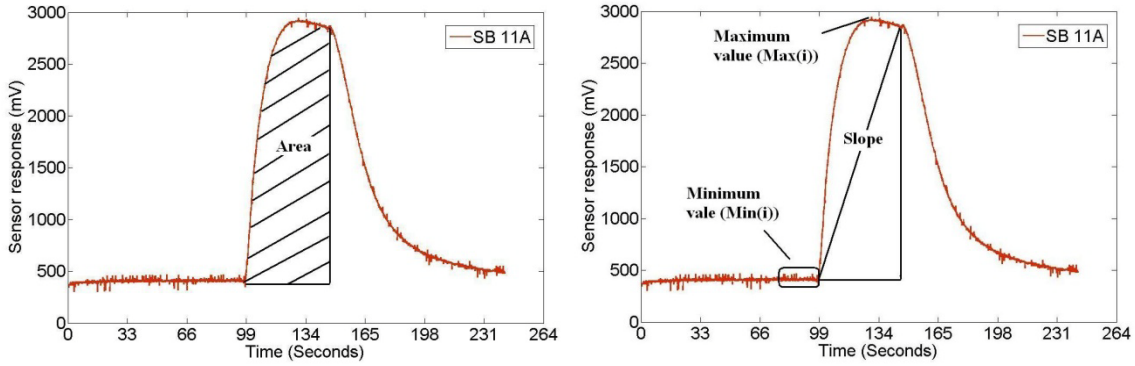

Figure 2.9: Response of one MOS sensor and the features extracted from the response.

Similarly, features were calculated from all the MOS sensors.

LabVIEW program

The graphical user interface software in the PC was developed using LabVIEW. The purpose of designing this software was mainly to facilitate communication between the user and the device, data processing, downloading and display. It was mainly used to interface with the device. The device was configured by the user by making selections on the GUI and transferring them to the microcontroller in the form of instructions (shown in Figure 2.7). The instructions were sent to the device through the RS232 cable connected to the device.

The software was designed to easily configure the device even for the first-time user. When the device was initiated for the very first time, it was programmed to inform the user to sync the time on the PC with the device. This procedure need not be repeated every time the device was switched on due to individually powered real time clock chip (DS1302). The number of seconds was selected by the user for each phase namely baseline, sampling and purging. The type of features and the pump speed can be selected from the drop down list. The user can start recording the volatile profile by clicking the "Start Recording" button shown in Figure 2.10.

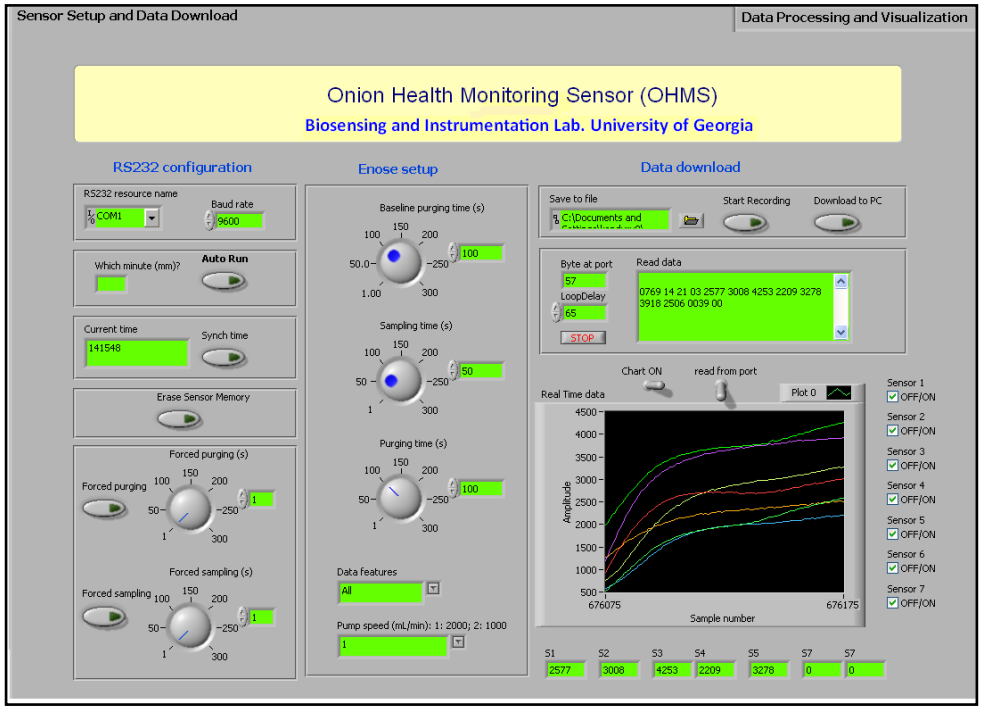

Figure 2.10: Sensor setup and "download data" tab of the GUI software. Functions namely recording each MOS sensor response, Displaying the response, Downloading the saved data in the device to the PC, Erasing the device memory, Synching time and forced purging and sampling was performed.

"Download to PC" selection instructs the microcontroller to transfer the data

saved in the memory chip to the PC. A .txt format file was created automatically in which the data was saved. Each line contained the serial number, time stamp, area under the curve, slope and relative response values for each MOS sensor, temperature and humidity value. The location of the file could be found in the user specified path in the "save to file" box shown in the "data download" section in Figure 2.10. The GUI software could also be used to erase all the data stored in the memory chip on the sensor board. This was performed by clicking the "Erase Sensor memory" option.

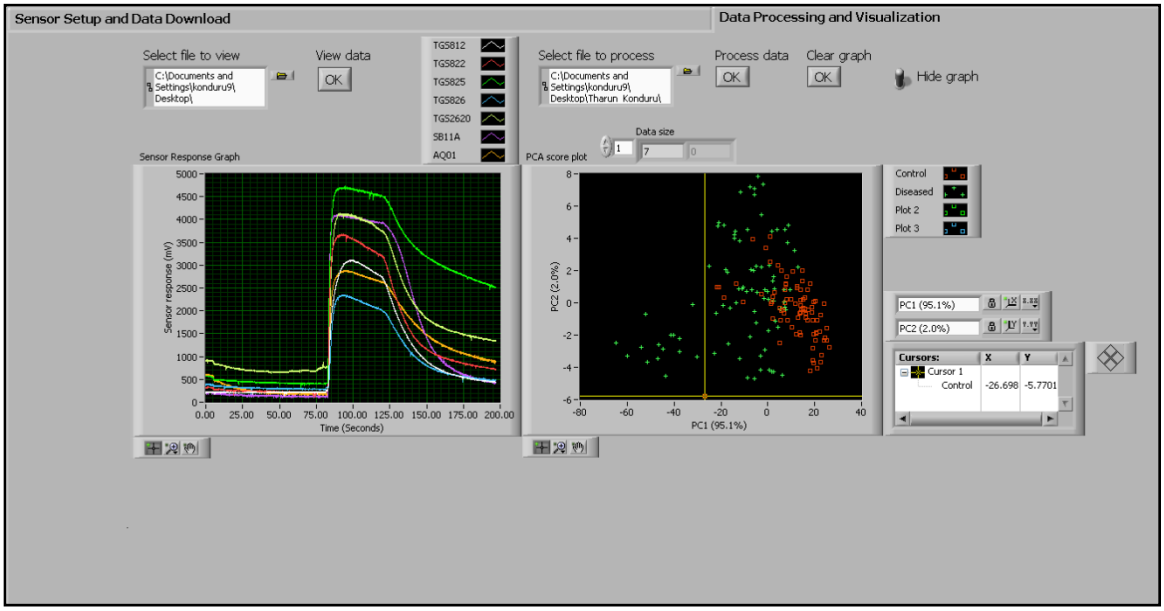

Figure 2.11: Data processing and visualization tab was used to view the collected data in the form of graphs (shown on the left side of the image). Principal component analysis was performed using GUI software. The PCA score plot was obtained along with the principal component values.

In Figure 2.11, the data processing and visualization tab is shown. The real time data was obtained from the device in the .txt format. This file was used to generate an excel (.xlx format) file and the data was graphed. This process was carried out by specifying the path of the text file (.txt format) in the "select file to view box" and

clicking the view data button. This action lead to copying the data in the text file to an excel sheet where it was then graphed in the dedicated graph box. Figure 2.11 shows the baseline, sampling and the purging phases. The generated graph was used by the user for quick observation of sensor responses. On the right side of Figure 2.11, principal component analysis (PCA) was conducted by specifying the path of the file in the "Select file to process" box and clicking "Process data" option. This automatically conducts the PCA analysis. The results were presented in the graph displayed on the right side of Figure 2.11, which was used for quick evaluation and diagnosis.

DEVICE CHARACTERIZATION

The hardware and electrical portion constitute the physical components of the device. As for the brain of the device, the software was designed and coded with instructions for automation. The following step involved testing the device to characterize its operation and understand its strengths and weaknesses.

Determine the effects of two different pump speeds on the sensor response

The pump was programmed to operate at two different speeds. The higher pump speed setting might attain the peak response much faster compared to the lesser pump speed settings. But, it was essential to determine if there was significant difference in peak response attained by each MOS sensor for the two different pump speed settings. A student's t-test was conducted and the results discussed.

100% ethanol solution was drawn from an ethanol bottle using a sterile 3 mL syringe and 0.1 mL of solution injected into three clean 250 mL glass jars (Fisher scientific, Pittsburg, PA, USA). The mouths of the jars were covered with clean aluminum foil and were sealed by fastening a metal hollow cap on the beaker. The jar

was allowed to stand for 30mins to allow complete evaporation of the sample solution into the headspace inside the jar. The sample inlet pipe was placed into the jar approximately 1 to 2 seconds before the device reached the sampling phase and the sample was drawn into the chamber for analysis. The pump speed affected the rate at which samples were delivered to the MOS sensors resulting in varied response time. The time taken by the MOS sensors in both cases was recorded and presented in Figure 2.12(b).

All the MOS sensors other than Tgs 813 and SB 11A reached maximum response for the given concentration of ethanol at faster rate for high pump speed (95%) compared to the 58% of the pump speed. Time taken by Tgs 813 and SB 11A response was not significantly affected by the different pump speeds based on the t-test results. The maximum response attained by each MOS sensor for two different pump settings is presented in Figure 2.12(b). The maximum response of Tgs 813 and Tgs 826 MOS sensors which are situated near the entry and exit points respectively showed significant difference for two different pump speed settings. Based on the results, a default setting of high pump speed was recommended for all the samples for its fast response and recovery time compared to the low pump speed.

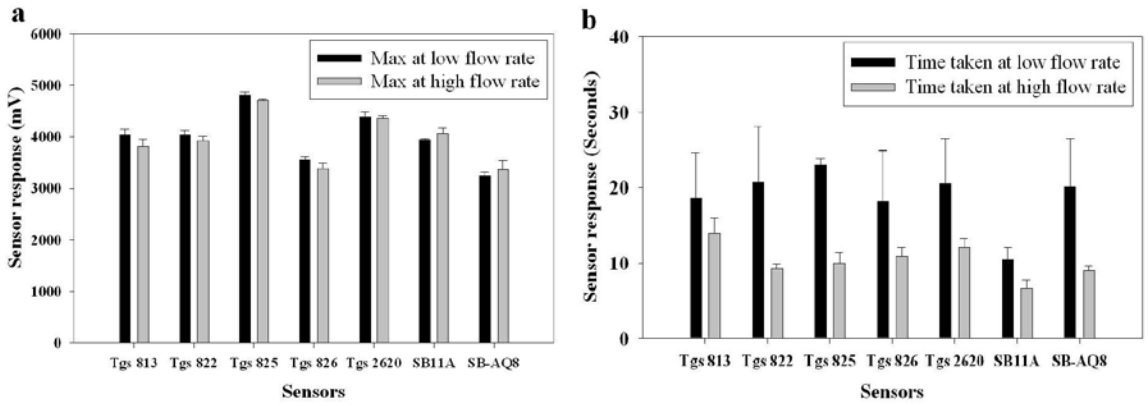

Figure 2.12: (a) Maximum response (mean ± S.D) for 100% ethanol solution obtained by each of the seven MOS sensors at 'Low' and 'High' pump speeds. Except Tgs 813 and Tgs 826 rest of the 5 MOS sensors do not have significant difference in achieving maximum response at 95% confidence interval (b) Time taken (mean ± S.D) by each of the MOS sensors to reach maximum response at 'Low' and 'High' pump speeds. Tgs 813 and SB 11A MOS sensors did not show significant difference in time taken to respond for two different pump speeds

Table 2.2: Mean comparison values obtained as a result of student's t test ($\alpha = 0.05$) for the maximum sensor response at two different pump speed settings along with the time taken to reach maximum response.

MOS sensor response at two different pump speeds (High and low)	Mean values of maximum response	Mean time taken to reach maximum response
Tgs 813_low	4040.6^{C}	18.57^{ABC}
Tgs 813_high	3819.6^{D}	13.92^{BCD}
Tgs 822_low	4042.0^{C}	20.68^{AB}
Tgs 822_high	3926.0^{CD}	9.18^{DE}
Tgs 825_low	4816.0^{A}	23.04^{A}
Tgs 825_high	4705.0^{A}	$9.95D^{E}$
Tgs 826_low	3555.3^{E}	18.15^{ABC}
Tgs 826_high	3382.3^{F}	10.92^{DE}
Tgs 2620_low	4386.3^{B}	20.44^{AB}
Tgs 2620_high	4355.3^{B}	12.13^{CDE}
SB 11A_low	3948.6^{CD}	10.50^{DE}
SB 11A_high	4069.3^{C}	6.63^{E}
SB AQ8_low	3246.3^{F}	20.10^{AB}
SB AQ8_high	3359.0^{F}	8.89^{DE}

*Variables that are not connected with the same letter are significantly different

Determine the effect of sensor response due to sensor positioning

The MOS sensors were positioned in a circular fashion inside the chamber. Two MOS sensors were positioned near the inlet and outlet holes through which the sample gas enters and leaves the chamber respectively. It was important to determine if these MOS sensors obtained a higher response because of their close proximity to the entry/exit points. The following test procedure addresses this question.

The same procedure was followed as the previous test, but the sample was drawn in two different directions represented as “Flow 1” and “Flow 2”. Flow 1 was represented for the sample flow where the entry point was close to TGS 813 and TGS 825 was the exit point. Flow 2 was represented for the sample flow where the entry point was TGS 825 and the exit point was close to TGS 813. The Flow1 and Flow 2 directions are shown in Figure 2.14 and the maximum values attained by individual MOS sensors are shown in Figure 2.13.

Paired t-test ($\alpha = 0.02$) was performed using JMP® Pro 9.0.2 (SAS Campus Drive, Cary, NC, USA). The statistical test was performed for all the three features. The mean change in the MOS sensor response for both the flow directions was not significantly different when observed with area under the curve (P-value = 0.1042), slope (P-value = 0.3434) and relative response (P-value = 0.8127) readings. Hence, the sensor response was not affected by the sensor positioning at the entry and exit points.

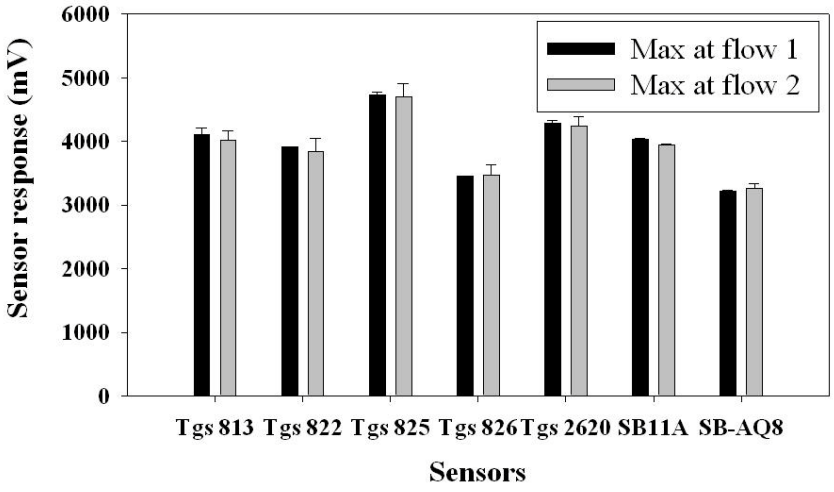

Figure 2.13: Maximum response (mean ± SD) for 100% ethanol reached by the MOS sensors when tested for two different flow directions.

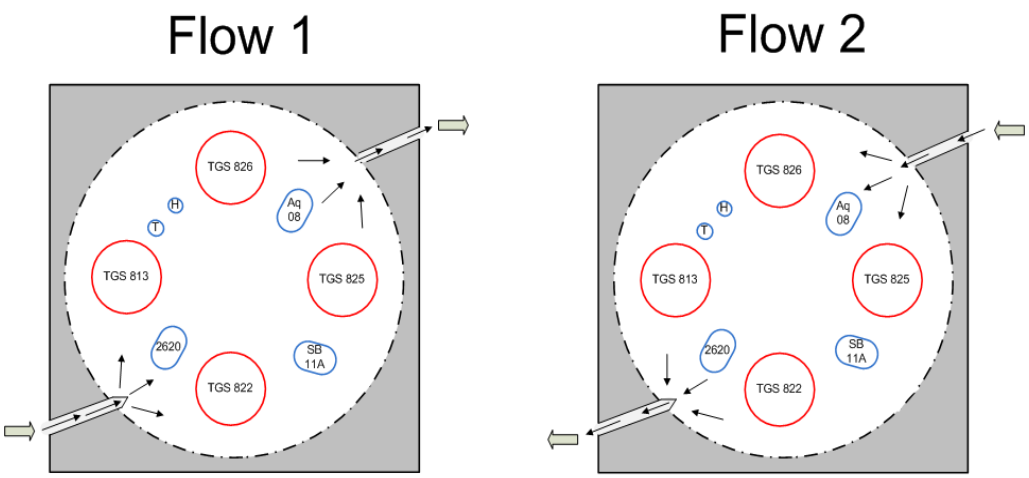

Figure 2.14: Odor sample flow directions (Flow 1 and Flow 2) into the sensor chamber

Discrimination of different concentrations of four chemicals

The operation and sensitivity of the device was tested by exposing it to the four chemicals, acetone (ketone), ethyl acetate (ester), acetonitrile (nitrile) and ethanol (alcohol).

The chemicals acetone (ketone), which is a colorless, flammable organic compound, ethyl acetate (ester), which is a sweet smelling, colorless organic compound, acetonitrile (nitrile), which is a simpler form of colorless nitrile and ethanol (alcohol), which is a flammable, colorless volatile liquid were exposed to the device. Each of 20 µl, 100 µl and 400 µl of these chemicals were placed in clean 500 ml glass jars (Fisher scientific, Pittsburg, PA, USA). Eight replicates were prepared. The beakers were covered with aluminum foil and sealed by fastening a metal hollow cap before allowing them to stand for 15 minutes to ensure complete transition of chemical from liquid to gaseous state. The state of the chemical was also visually confirmed by looking for traces of the chemical on the base of the beaker. The inlet pipe of the sensor was then

carefully and immediately placed inside the beaker. The sensor was then configured by providing the parameters to the microcontroller. For the sensitivity tests, 100 seconds was set for both baseline purging and purging, whereas sampling was done for 35 seconds.

Concentration of the chemical was calculated in units of moles per liter (molarity). Assuming that the chemical had completely evaporated into the beaker after confirming visually that there is no trace of sample in the jar, the concentration of each chemical was calculated using the following equation.

$$C = D / MW \times V \qquad \text{(Rains et al., 2004)} \quad \ldots\ldots\ldots\ldots [4]$$

where,

C = concentration of the chemical calculated theoretically, units = mol L^{-1}

D = dosage in jar, units = grams

MW = molecular weight of chemical, units = g mol^{-1}

V = volume of air in jar (0.5 L).

Table 2.3: Lists the concentration (Units: M) of four chemicals at three different quantities.

	Ethanol x 10^{-3} M	**Acetone x 10^{-3} M**	**Ethylacetate x 10^{-3} M**	**Acetonitrile x 10^{-3} M**
20 µl	6.85	5.4	4.07	7.65
100 µl	34.25	27.13	20.3	38.29
400 µl	137.01	108.5	81.45	153.17

All the experiments were conducted with a minimum of eight replicates to confirm the repeatability of the response for certain samples. The three features (slope,

area and relative response) were used to identify the best feature that could discriminate all four chemicals.

Concentrations of 20µl, 100 µl and 400 µl were considered for the test. MANOVA (Multivariate Analysis of Variance) was performed on all three features extracted from the sensor response.

The null hypothesis was tested that there was no significant difference between the same concentrations of all four chemicals and the concentrations within each chemical for all three features. MANOVA test was performed using JMP® Pro 9.0.2 (SAS Campus Drive, Cary, NC, USA). Wilk's Lambda statistic was used for testing the significant difference in concentrations within each chemical and between the chemicals. Based on the P-value (<0.0001) results, the null hypothesis was rejected for all comparisons, which indicated that the device responded with significant difference for all three concentrations within the chemical and between the chemicals.

Discrimination of volatiles released by onions

Methylpropyl-di-sulfide (90%) (Sigma-Aldrich, St. Louis, MO, USA) was released by sour skin and botrytis neck rot diseased onion on 3rd and 6th day after inoculation whereas 2-Nonanone (>99%) (Sigma-Aldrich, St. Louis, MO, USA) was released by sour skin diseased onion on both 3rd and 6th day after inoculation. Both the chemicals were not released by control onions (Li et al., 2011). The selected MOS sensors were tested for their response to two different concentrations of these volatiles. 0.5 µL and 5 µL of both the chemicals were pipette into a clean 500 mL beaker. Seven and eight replicates were prepared for each concentration of methylpropyl-di-sulfide and 2-

Nonanone respectively. The beaker was sealed and allowed to stand for 15 minutes. The collected headspace volatiles were exposed to the sensor for 35 seconds.

In Figure 2.15, it was observed that Tgs 826 and SB AQ8 sensors had high response to both the chemicals at 0.5 µL compared to other MOS sensors. Tgs 813 responded the least.

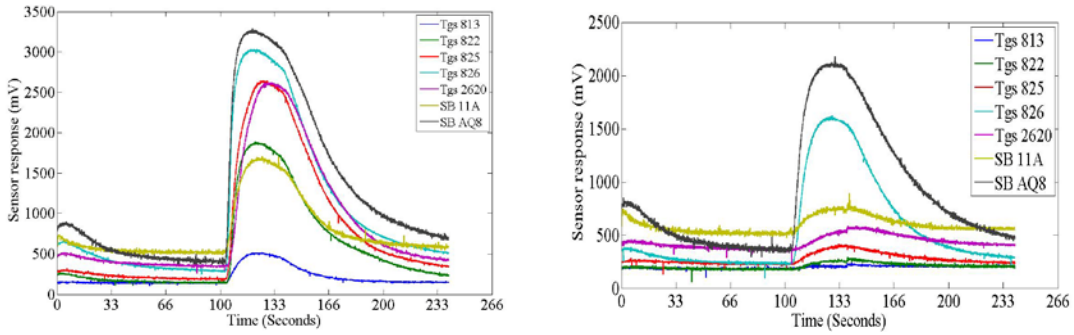

Figure 2.15: Response obtained from 7 MOS sensors to 0.5 µL of methylpropyl-di-sulfide (a) and 2-nonanone (b).

The features extracted from the sensor response were statistically analyzed using MANOVA statistical procedure. The three features relative response, area and slope obtained for 0.5 µL and 5 µL concentrations of methylpropyl-di-sulfide showed significant difference with a p-value of 0.0012, 0.0003 and 0.0353 respectively. Area and relative response features extracted for 0.5 µL and 5 µL of 2-Nonanone chemical showed significant difference based on p values obtained as 0.0164 and 0.0122 respectively. However, slope feature extracted from sensor response to two concentrations of 2-Nonanone was not significantly different (p-value = 0.0678).

CONCLUSIONS

In this study, a low cost gas sensor array was fabricated with an automated gas delivery system and data acquisition features to detect the presence of volatile

compounds. The sensor was less expensive costing as low as $900 which would enable users to use multiple sensors based on the application. The device was automated with a specially designed electrical circuit and interfaced with GUI program developed using LabVIEW. The device was designed to be compatible for multiple applications with minor modification to the instrument. The device was characterized and the results showed that the sensor operated well at high pump speeds irrespective of the size of the sample headspace. The location of the MOS sensors within the chamber did not have a significant effect on sensor response and hence the MOS sensors can be placed at any place within the chamber. The device was able to differentiate all the four chemicals namely ethanol, acetone, acetonitrile and ethyl acetate when exposed with the same concentrations and was able to differentiate differences in the concentrations within and between the chemicals. Tgs 826 and SB AQ8 was observed to respond better than other MOS sensors for 0.5 µL concentration of methylpropyl-di-sulfide and 2-Nonanone. All three extracted features (area, slope and relative response) showed a statistical difference between the two concentrations (0.5 µL and 5 µL) of methylpropyl-di-sulfide. A similar test revealed that 2 of the 3 extracted features (relative response and area under the curve) discriminated between 0.5 and 5.0 µL concentration of 2-nonanone.

ACKNOWLEDGMENTS

The project was funded by USDA NIFA Specialty Crop Research Initiative grant (Award Number: 2009-51181-06010). Special thanks to technician Mr. Dewayne Dales for making himself available and humbly offering his expertise for fabricating the device.

REFERENCES

Abbey, L., Aked, J., and Joyce, D. C. (2001). "Discrimination amongst Alliums using an electronic nose." *Annals of Applied Biology*, 139(3), 337-342.

Aishima, T. (1991). "Aroma discrimination by pattern recognition analysis of responses from semiconductor gas sensor array." *Journal of Agricultural and Food Chemistry*, 39(4), 752-756.

Green, G. C., Chan, A. D. C., Dan, H., and Lin, M. (2011). "Using a metal oxide sensor (MOS)-based electronic nose for discrimination of bacteria based on individual colonies in suspension." *Sensors and Actuators B: Chemical*, 152(1), 21-28.

Haddad, R., Medhanie, A., Roth, Y., Harel, D., and Sobel, N. "Predicting Odor Pleasantness with an Electronic Nose." *PLoS Comput Biol*, 6(4), e1000740.

Kea-Tiong, T., Shih-Wen, C., Chih-Heng, P., Hung-Yi, H., Yao-Sheng, L., and Ssu-Chieh, L. (2010). "Development of a Portable Electronic Nose System for the Detection and Classification of Fruity Odors." *Sensors (14248220)*, 10(10), 9179-9193.

Klinbumrung, A., Jaroensutasinee, K., Pratontep, S., and Kerdcharoen, T. "Low - cost Electronic nose evaluated on Thai-herb of Northern Thailand samples using multivariate analysis methods." *AIP Conference Proceedings*, 497.

Li, C., Krewer, G. W., Ji, P., Scherm, H., and Kays, S. J. (2010). "Gas sensor array for blueberry fruit disease detection and classification." *Postharvest Biology and Technology*, 55(3), 144-149.

Li, C., Schmidt, N. E., and Gitaitis, R. (2011). "Detection of onion postharvest diseases by analyses of headspace volatiles using a gas sensor array and GC-MS." *LWT - Food Science & Technology*, 44(4), 1019-1025.

Mamat, M., Samad, S. A., and Hannan, M. (2011). "An Electronic Nose for Reliable Measurement and Correct Classification of Beverages." *Sensors*, 11(6), 6435-6453.

Panigrahi, S., Balasubramanian, S., Gu, H., Logue, C. M., and Marchello, M. (2006). "Design and development of a metal oxide based electronic nose for spoilage classification of beef." *Sensors and Actuators B: Chemical*, 119(1), 2-14.

Pearce, T. C. (2003). *Handbook of machine olfaction : electronic nose technology / [edited by] T.C. Pearce ... [et al.]*, Weinheim [Germany] : Wiley-VCH, c2003.

Rains, G. C., Tomberlin, J., D'Alessandro, M., and Lewis, W. J. (2004). "Limits of volatile chemical detection of a parasitoid wasp, Microplitis croceipes, and an electronic nose: a comparative study." *Transact Am Soc Agri Engin*, 47, 2145-52.

Vikram, A., Hamzehzarghani, H., and Kushalappa, A. C. (2005). "Volatile metabolites from the headspace of onion bulbs inoculated with postharvest pathogens as a tool for disease discrimination." *Canadian Journal of Plant Pathology*, 27(2), 194-203.

Zampolli, S., Elmi, I., Ahmed, F., Passini, M., Cardinali, G. C., Nicoletti, S., and Dori, L. (2004). "An electronic nose based on solid state sensor arrays for low-cost indoor air quality monitoring applications." *Sensors and Actuators B: Chemical*, 101(1-2), 39-46.

CHAPTER 3

DETECTING THE PRESENCE OF SOUR SKIN INFECTED ONIONS AMONG HEALTHY ONIONS USING A CUSTOMIZED GAS SENSOR ARRAY[2]

[2] [2] Konduru, T., Li, C. and Rains, G.C. To be submitted to *Sensors and Actuators B.*

ABSTRACT

Onion storage losses in the United States are mainly due to post-harvest diseases. There are various methods to detect diseases of which gas sensing technology was investigated in this study. The study focuses on early detection of sour skin infection in onions using a customized gas sensor array. The sensor was built specifically to detect the difference in type of volatiles released by healthy and sour skin infected onions. It consisted of an array of seven commercially available metal oxide semiconductor (MOS) gas sensors mounted inside a sensing chamber with a sample delivery system. Three features (relative response, area under the curve and slope) were extracted from the sensor responses. The testing phase was carried out in two batches. Each batch contained eight replicates for each of two treatments (control and sour skin). Sour skin infection was introduced into the onions by inoculation with *Burkholderia cepacia*, which is the causal pathogen for sour skin infection. Noise was filtered from raw data using a "5-point mean" filter. Based on a MANOVA test, the "relative response" feature corrected with the relative baseline correction method provided the best detection of infected onions among healthy ones. A PCA score plot demonstrated the effect of infection on the volatiles released by onions over a period of 4 days. Statistical analysis showed that Tgs 813 was the least significant sensor followed by Tgs 2620. Whereas, Tgs 826 and SB-AQ8 performed well in discriminating healthy and rotten onions. Classification performance was analyzed for three different sensor selections (7-sensor, 6-sensor and 5-sensor) of which all seven MOS sensors were observed to be best for detecting infected onions. The study demonstrated the potential of a gas sensor array to discriminate infected onions placed among healthy onions in storage.

INTRODUCTION

Onion is the second most important commercial vegetable crop for fresh market in the United States (USDA-NASS, 2012). It is estimated that approximately 105 billion pounds of onions are produced each year. Across the world, average annual onion consumption per person is estimated to be 13.67 pounds (National Onion Association, 2011). Onions like any other vegetable are threatened by various bacterial or fungal diseases. This study focuses on detection of the post-harvest bacterial disease, called sour skin, caused by the bacteria *Burkholderia cepacia*. The pathogen survives in the soil and is splashed onto the leaves and into the neck of the onion during rain or overhead irrigation. The presence of wounds and water soaked tissue provides an opening to these bacterial cells into the onion. Warm weather (~ 30 ºC) and high humidity promotes bacterial growth resulting in brown and water soaked scales under the first or second onion layer (Li et al., 2011). During the earlier stages of bacterial infection, the symptoms are limited to a very small portion of inner scales of the onion. Hence, it is difficult to sort the infected and healthy onions during human visual inspection before storage (Gitaitis and Tollner, 2005). This disadvantage leads to spreading of infection among onions during cold storage causing up to 50% losses in some individual storage houses (Gitaitis and Tollner, 2005; Tollner et al., 1995).

A wide range of non-destructive testing techniques such as X-ray imaging and hyper-spectral imaging have been employed to test food quality. Currently, human visual inspection (HVI) is preferred for quality sorting in packing houses due to lack of reliable technology. However, this method has increased error in detecting internal infection due to lack of prominent symptoms during earlier stages of infection. Once the

onion passes the HVI, they are kept in cold storage as long as needed. The pathogen inside the onion spreads to surrounding healthy onions during cold storage. There is an immediate need in the onion industry for a sensing system that can monitor the quality of onions during storage, thereby minimizing the frequent and time consuming human inspection process. The potential of electronic nose technology to detect sour skin in onions was explored in this study by using a customized gas sensor array to differentiate diseased onions placed among healthy ones.

Literature on the use of electronic nose technology

Developing an effective detection technology that can identify the presence of pathogens in onions at their initial stage of infection can reduce onion post-harvest losses. Fruits, vegetables and meat products release varying compositions of volatile organic compounds (VOCs) based on the physiological changes occurring within them. Changes in these volatiles can be used as an indicator to determine their condition. Hence, there is a need for an automated sensing system which can monitor the quality of the fruit or vegetable.

In order to develop an application-specific sensing system, the right type of sensors has to be selected based on the application. Hence, it is crucial to determine the various types of volatile compounds released by the sample under investigation. One gas chromatography –mass spectrometry (GC-MS) study was conducted that identified the various VOCs released by onions when treated with *B.cepacia* and *B.alli* (Li et al., 2011). This study served as an important source of information to select sensors for developing gas sensing technology for sour skin disease detection in onions. Unlike GC-MS, which provides information on individual volatile compounds, gas sensing

technologies provides information on difference in volatile profiles. Gas sensors are a rapid and inexpensive alternative compared to GC-MS. The gas sensor array's (also known as electronic nose (E-Nose)) invention led to applications for a wide range of industrial applications by using commercially available or customized sensing technologies. Electronic nose mimics how human nose perceives smell. During inhalation, the volatiles enter the nasal cavity and binds to specific receptors. These receptors signal the brain which then processes the information to identify the smell. Similarly, electronic nose consists of a gas delivery system which delivers the volatiles to the sensing material. The sensing material reacts to the volatiles based on their concentration and outputs voltage signal accordingly. The voltage signal can be processed and analyzed to identify the differences in odor profiles.

Various commercially available and customized sensing devices have been used for applications involving food products namely such as meat, fruits and vegetables. It should be noted that these gas sensing devices comprise of the gas sensors, gas delivery system, data acquisition and data processing components.

Commercial gas sensing devices

Commercial gas sensing instruments are manufactured for general purposes without targeting a specific application. These devices have shown successful discrimination based on the volatiles exposed to it. However, they cannot guarantee successful results for a specific application unless tested and documented. For the testing procedures, they have to be purchased commercially. Commercial gas sensing instruments used for various applications are as follows:

A commercially available sensing device named FOX 2000 (6 MOS sensors) was used for discriminating eight species of microbes (Rossi et al., 1995; Vernat-Rossi et al., 1996) and FOX 3000 (12 MOS sensors) with an HS 100 odor scanner auto sampler to investigate the microbiological quality of raw poultry meat. The study was conducted with a change in storage time and temperature conditions (Boothe and Arnold, 2002). These studies are indicative of the potential of MOS sensors to discriminate changes in volatile composition with respect to time. On the other hand, other type of gas sensors such as conducting polymers (CP) and surface acoustic wave sensors (SAW) were used in Cyranose 320 Enose and zNose respectively to detect the difference in volatile compounds released by damaged and undamaged apples (Li et al., 2007). Conducting polymer based sensing device was also used for evaluation of onion bulb quality (Abbey et al., 2004) and discrimination amongst *Alliums* (Abbey et al., 2001a). Although this is the case with CPs, PEN 2 which is composed of MOS sensors showed potential to assess the different fruit maturity state (Benady, 1995; Gómez, 2006) and quality (Di Natale et al., 2001). Based on the market availability and price MOS sensors were a better choice compared to CP and SAW. MOS sensors in combination with a Quartz microbalance sensor (MOSES II e-nose) was used to identify the pleasantness of an odor by comparing it with the rating provided by human subjects concluding that the e-nose used was capable of providing human-like ratings (Haddad et al., 2010).

Customized gas sensing devices

Customized gas sensing devices are those devices that are custom made for specific applications. In some cases, these devices are fabricated and equipped with

commercially available data acquisition and processing products. These devices have proven to be successful in a wide range of applications as well in the following examples.

A customized gas sensing device (21 MOS sensors) was developed and used with commercial data acquisition software to assess the ripeness of pinklady apples during shelving period (Brezmes et al., 2001). Whereas, 14 MOS sensors were used to assess the grape juice quality based on the initial stages of alcoholic fermentation (Gutierrez et al., 2007). MOS sensors were also used successfully to measure subtle changes in volatile profile indicating the state of maturity and damage in blueberries (Simon et al., 1996). Each sensing device was designed for an application involving one type of fruit. On the other hand, multiple complex fruit odors namely lemon, banana and litchi was classified with 95% classification accuracy using a device with as low as 8 MOS sensors connected to commercial data acquisition software (Tang et al., 2010). In the case of vegetables, multiple sensors were employed in the sensing device to detect the presence of soft rot in stored potato tubers (de Lacy Costello, 2000). Detecting changes in the chemical concentrations of acetone in the air was also achieved using an array of MOS sensors (Gonzalez-Jimenez et al., 2011) as well. Other volatile sensors have been employed to measure volatile odors. Thin film metal oxide gas sensors were used to monitor the indoor air quality based on the concentration of various air contaminants (Zampolli et al., 2004).

Gas sensors have also shown potential value in detecting odor profiles of other food products such as meat. For example, 7 MOS sensors were assembled in an array connected to a commercially available data acquisition system to detect the volatiles

released by beef strip loins to classify beef at two different temperatures (Panigrahi et al., 2006). Most of the gas sensing devices were customized using commercially available data acquisition and processing software and required human presence to commence sampling and other functions. Hence, there is a need for an automated sensing device designed for disease detection in onions functioning in a timely manner without human intervention. The data acquisition and data processing software needs to be designed specifically for the device as well.

Based on the studies discussed above, commercial devices are the primary choice for studies involving gas detection eliminating the need to develop one. Although this is the case, customized gas sensor array can be designed for low cost compared to the high priced commercially available gas sensing devices. The use of MOS sensors in customized gas-sensing technology has shown to be successful for a wide range of applications (meat, fruits, vegetables) requiring detection of changes in volatile profile. Also, MOS sensors are easy to operate and are readily available for purchase at a low price and are designed to be sensitive to a wide range of compounds with high sensitivity. The drawback of using MOS sensors are their high power consumption to maintain high temperatures for effective sensing (Rouseff and Cadwallader, 2001a) and lack of specificity. This study was conducted to determine if a customized sensor array designed using an array of MOS sensors could be developed to detect sour skin disease in onions.

The various metabolites released by sour skin and botrytis neck rot infected onions were identified (Li et al., 2011) in which the volatiles released by sour skin infected onions were considered for the sensor selection. Detecting an odor emitted by

rotten onions while being masked by the odor of surrounding healthy onions is a major challenge. This study is the first attempt to test the ability of a completely automated, customized gas sensing device in detecting the presence of diseased onions.

The objectives of this study are to 1) Compare three different baseline corrected features and determine the most suitable one for this application. 2) Conduct principal component analysis (PCA) and develop classification models to investigate the potential of the device to distinguish healthy and sour skin infected onions. 3) Select the best combination of MOS sensors from the seven available sensors that contributed the most to discriminating healthy and sour skin infected onions.

MATERIALS AND METHODS

Preparing onion samples

Jumbo yellow onion bulbs were purchased from a commercial store. The onions for the first batch experiment were shipped from Oregon and the second batch from Washington, USA. The dry skin of all the onions was peeled off and the bulbs were washed with distilled water to remove dirt. An additional layer of onion was peeled off if a minor physical bruise or scratch was observed. Onions with major bruises and damages were discarded. Onions were then surface sterilized with 70% ethanol solution and allowed to stand for 10 minutes before washing with distilled water to remove chemical residues. The moisture on the onions was wiped clean using a clean paper towel. The onions were then dried by placing them on a clean surface for 1 hour at room temperature.

Inoculation and infection

Cultures of *Burkholderia cepacia*, strain Bc 98-4, were obtained from the Natural Products Laboratory at the University of Georgia, Tifton. The cultures were grown on tryptic soy agar after incubating for ~48 hrs at 30 °C. The petri plates were then placed in room temperature at 26 °C to slow down the growth of bacteria. Using a sterile loop, the bacterial colonies were transferred carefully to a vial containing 45 ml of distilled water. The vial was thoroughly shaken until the bacterial colonies were mixed with water.

Using a spectrophotometer with an Eppendorf Biophotometer 22331 (Eppendorf, Hauppauge, NY, USA), an approximate number of bacterial cells were measured based on the quantitative measurement of the transmission and reflection properties. Distilled water was taken as reference in the cuvette and the bacterial solution was taken as the sample for the analysis. It was estimated that there were ~5 x 10^8 bacterial cells per 45ml of solution.

Using a 3ml sterile syringe (Becton Dickinson and company, Luer-Lok Tip, Mexico), 1ml each of bacterial inoculum was injected on two opposite sides of the neck region of the onion by inserting the needle ~30mm deep at an approximate angle between 45º to 55º from the base. The syringe and the onion bulb were handled by wearing clean sterilized gloves. Pressure was applied on the inoculated region using the thumb finger to make sure the bacterial solution did not flow back through the injected region. A tray of labeled infected onions was placed in an incubator for 48 hrs at 30 °C which was the optimum growth temperature for bacteria to facilitate sour skin infection in the onion samples. Similarly, the control onions were prepared by following the same

procedure and instead of injecting the bacterial solution, 1ml of distilled water was injected into each of the onion sample. Healthy onions were prepared by removing the dry skin layer and cleaning with distilled water before sterilizing with 70% ethanol solution. The sour skin inoculated onions and control onions were placed in a clean room maintained at room temperature (24 ± 2 ºC) for 3 hrs prior to placing the onions in a clean box and sealed using a thin strip of parafilm. Each box was prepared beforehand by spraying 70% ethanol solution, washed with distilled water, wiped thoroughly with clean paper towels and dried for 3 hrs to avoid presence of moisture inside the box. The onions were allowed to stand for 6 hrs for the volatiles to accumulate in the headspace for volatile analysis using the sensor. Allowing the onion samples in the room temperature for 8 hrs eliminated the possibility of onions being in the same temperature as incubation temperature when the volatile compounds in the headspace were analyzed for all the treatments.

Experiment design

Experiments were designed to investigate the potential of the customized gas sensor array to differentiate the presence of sour skin infected and healthy onions. The experiment was conducted in two batches. In each batch, 16 onions were used as control, 16 onions were used as sour skin infected onions and 48 onions were used as healthy onion samples. Control onion samples were treated with distilled water, whereas healthy onions were not subjected to any treatment. Sixteen clean plastic boxes were used for these experiments, in which 8 boxes (also referred to as replicates) were allocated for control onions named “Control onion box (COB)” and the remaining 8 boxes (also referred to as replicates) were used for sour skin infected onions named

"Infected onion box (IOB)". Each COB had three healthy onion bulbs and two control onion bulbs and each IOB had three healthy onion bulbs and two sour skin infected onion bulbs. The volatiles released by the onions were measured from the 3rd (the day onion samples were removed after 48 hrs of incubation) day after inoculation (dai) through the 7th dai for both batches. The total numbers of measurements taken are presented in table 3.1.

Table 3.1: The number of measurements from each COB and IOB over a period of 5 days after inoculation with batches of onion data combined together.

DAI	Control	Sour skin	Total
3rd dai	40	40	80
4th dai	40	40	80
5th dai	47	47	94
6th dai	48	48	96
7th dai	48	48	96
Total	223	223	446

Customized gas sensor array

The volatiles accumulated in the headspace were introduced to the customized gas sensor array. The gas sensor array consisted of seven individual metal oxide semiconductors (MOS). Five were purchased from FIGARO USA (Glenview, IL) and two from FIS Inc (Hyogo, Japan). In addition to the MOS sensors, a temperature (LM 35DZ, National Semiconductor, Santa Clara, CA, USA) and humidity sensor (HIH 4000 -002, Honeywell International Inc., Morristown, NJ, USA) were used to record the

environmental conditions. The sample inlet pipe in the sensor was attached to a moisture-resistant acetal push–to-connect adapter (McMaster-Carr, NJ, USA). It was an air tight adapter used to connect the device sampling tube to the onion sampling box. The pump pulled the headspace volatiles and the valve directed the flow to the chamber. This allowed free flow of headspace volatiles to the gas chamber without any leakage. Each measurement involved the following stages: 1. Baseline: The chamber was cleaned by introducing charcoal filtered air for 100 seconds and brought the sensor response to its baseline. 2. Sampling: Sample gas was introduced for 50 seconds. 3. Purging: The chamber was again cleaned by introducing charcoal filtered air for 100 seconds, preparing it for the next phase of experiments. The schematic for gas sensor array and the arrangement of onion samples in the sampling box is presented in Figure 3.1. During the sampling process, the MOS sensors within the chamber detected the presence of specific volatiles and responded in terms of voltage (mV) based on the concentration of the volatiles. The sensor was programmed to compute features and to transfer the data to a personal computer (PC) and save the calculated area, slope and relative response for each measurement in the memory chip along with the time stamp in .txt format. If the data was required to be saved directly in the PC in real time, the device was plugged into the PC using a RS232 cable. A unique response pattern from all seven sensors under investigation was obtained.

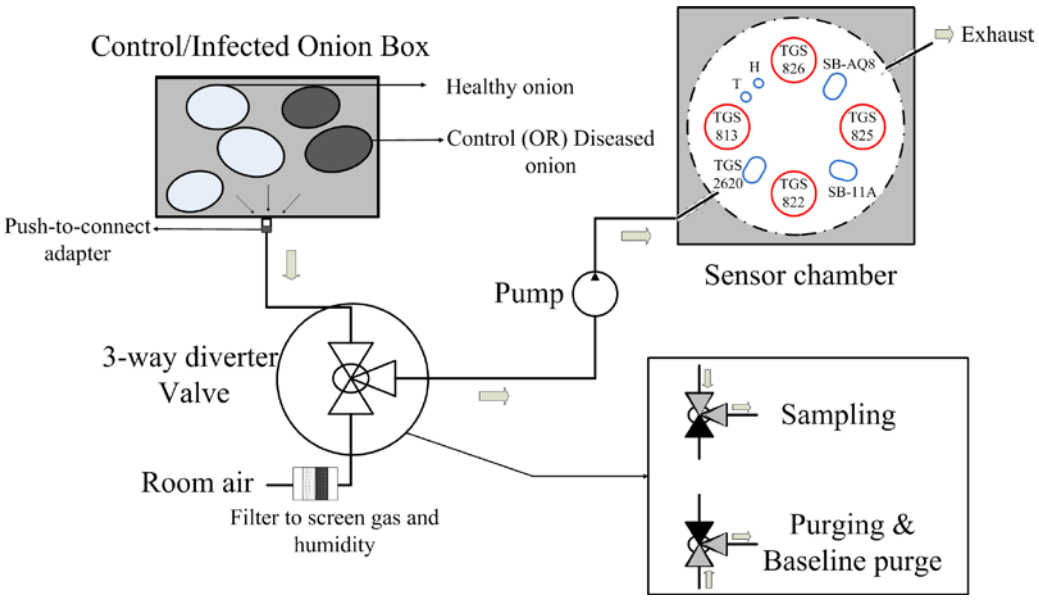

Figure 3.1: Schematic diagram of the customized gas sensing device along with the arrangement of healthy and control/sour skin diseased onions. The headspace gas accumulated in the sample box is drawn into the sensing chamber.

Onion box preparation prior to sampling

Clean plastic boxes were purchased to place the onion samples. Each box was sterilized by spraying 70% ethanol solution and wiping them with clean towels. The boxes were placed in an air conditioned room maintained at 25 °C ± 2 °C. Two control onions were placed in each control box and the two infected onions were placed in the sample boxes. Three healthy onions were placed in all the boxes. The boxes were covered and sealed using parafilm 6 hrs prior to data collection to facilitate accumulation of volatiles in the headspace. The samples were analyzed 3rd through 7th day after inoculation (dai).

The sensor sample input tube was inserted into a push-to-connect adapter attached to the side of the onion sample box and control box. This adapter held the input

tube firmly to prevent volatile gas leaking from the box and ensured efficient volatile delivery. The sample gas was drawn from the box and was delivered to the chamber containing MOS sensors with the combined effort of pump and valve.

Digital noise filter and baseline manipulation

The voltage signals obtained from individual MOS sensors were subjected to various types of noise. The noise in the signal could disrupt or decrease the potential for pattern recognition techniques used for classification of the healthy and diseased onions. Hence, filtering the noise from the data was crucial and an important part of the signal processing method. Signal-to-noise (SNR) ratio was calculated to the part of the baseline purge phase. A 5-point mean filter was used to reduce the noise in all the MOS sensors. The SNR values obtained before and after filtering were compared with each other. Filtering the data was followed by correcting the sensor response with respect to its baseline.

Commonly used techniques such as differential, relative and fractional baseline manipulation were employed to compensate for drift and noise and shown to be effective based on the nature of the gas sensors used and the type of application (Pearce, 2003). For this study, selection of the right baseline correction (BC) method was required to increase the classification performance. Three baseline correction methods were investigated to select the best one that enhanced the performance of the classification model (Green et al., 2011), (Arshak et al., 2004). The three baseline correction methods are discussed in detail below.

The differential baseline correction method used involved removing the mean baseline value from the value obtained at time‘t’. This method was used to remove the additive noise in the raw sensor response voltage signal.

$$Y_{i(t)} = X_{i(t)} - M_i \quad \text{..... [1]}$$

where,

$Y_{i(t)}$ = Baseline corrected value of sensor ‘i’ at time‘t’,

$X_{i(t)}$ = Raw data of sensor ‘i’ at time‘t’,

M_i = Mean baseline value of 5 seconds of data used prior to sampling stage of sensor ‘i’.

The relative baseline corrected technique was performed for individual MOS sensor’s response. This method removed the short term baseline drift in the response and improved the contrast considerably. The resulting data was dimensionless (Gutierrez-Osuna, 2002)

$$Y_{i(t)} = \frac{X_{i(t)}}{M_i} \quad \text{..... [2]}$$

where,

$Y_{i(t)}$ = Baseline corrected value of sensor ‘i’ at time‘t’,

$X_{i(t)}$ = Raw data of sensor ‘i’ at time‘t’,

M_i = Mean baseline value of 5 seconds of data used prior to sampling stage of sensor ‘i’.

Fractional baseline correction method is used to remove both additive and multiplicative noise in individual MOS sensor response signal.

$$Y_{i(t)} = \frac{X_{i(t)} - M_i}{M_i} \quad \text{..... [3]}$$

where,

$Y_{i(t)}$ = Baseline corrected value of sensor ‘i’ at time‘t’,

$X_{i(t)}$ = Raw data of sensor ‘i’ at time‘t’,

M_i = Mean baseline value of 5 seconds of data used prior to sampling stage of sensor ‘i’.

The baseline correction methods were used on all three features (area, relative response and slope). Area feature was calculated by applying the trapezoidal rule to the section of the response when the device was exposed to the sample odor. Slope feature was calculated by dividing the difference between the maximum and minimum values with the time taken for each individual MOS sensor to reach the maximum sensor response. Relative response feature was calculated by dividing the difference between the maximum and minimum with the minimum sensor response.

Data analysis and processing

MULTIVARIATE SIGNIFICANCE TESTING AND PRINCIPAL COMPONENT ANALYSIS:

The multivariate analysis of variance (MANOVA) test was performed on all three features (area, relative response and slope), each corrected with differential, relative and fractional BC methods. Each feature was corrected with each of the baseline correction methods resulting in three datasets for each feature. Hence, a total of 9 datasets were obtained. The MANOVA statistical test procedure was performed to observe if the features derived from the sensor response subjected to BC methods could be used to differentiate infected onions placed among healthy onions. Based on the Wilk’s lambda and F values, the best feature and baseline correction method combination was selected. The selection criteria also included consistent increase in F values (or decrease in Wilk’s lambda value) indicating an increase in discrimination ability of the sensing device with the increase in number of dai. The consistency factor

is an indication of device potential to sense the changes in the volatiles released by the onions as the disease progresses. The significance in sensor response for discriminating the healthy and diseased onion was verified based on the p-value. The statistical test was performed to test the null hypothesis for lack of significant difference between the healthy and diseased onion. The MANOVA statistical test procedure was performed using SAS v9.1 software (Statistical Analysis Software, Cary, NC, USA).

Sensor selection was one of the crucial steps during data analysis. When multiple MOS sensors were used, sensor selection helped identify the MOS sensors that contributed the most to classification between the groups. The performance of the classification models can be improved by carrying out good sensor selection methods. Likewise, bad sensor selection can affect the classification model performance. In this study, all 7 MOS sensors need not necessarily contribute to the identification of important volatiles that differentiates between the control and diseased onion groups. There could be the possibility for some of the sensors to disrupt the classification performance. Considering data from the selected baseline corrected feature, MANOVA statistical test was used to identify the least performing MOS sensors based on the first batch onion data.

Principal component analysis (PCA), a multivariate method, was used to visualize the difference or similarities between the two groups (control and diseased). In this method, the dimensionality of a huge number of interrelated variables was reduced to a few important principal components (PC). These principal components were uncorrelated and were arranged in such a way that the first two or three components held most of the variation present in all of the variables. The number of components that

contributed the most variance was determined. The PCA score plot presents the qualitative information using the spatial distance similarities and differences between the treatments.

SMELLPRINT

A smellprint was obtained by taking the average of individual MOS sensor response for both the treatments on 7^{th} dai of first batch onion data. The infection in the onion samples is more prominent on seventh day after inoculation compared to other days. This was conducted to observe the difference in each MOS sensor's response for both the treatments.

VALIDATION AND CROSS-VALIDATION MODELS

Linear discriminant analysis (LDA)

The linear discriminant analysis (LDA) was performed using MATLAB (Math works Inc., Natick, MA, USA) to classify the two groups (control and diseased onion) based on their similarities. The closeness of the healthy onion and diseased onion groups was measured using the mahalanobis distance. A total of 446 data sets were considered which included both batches of onion data starting from 3^{rd} through 7^{th} dai. The results were validated by training the LDA model using the first batch of onion data and test using the second batch.

Support vector machine (SVM)

Support vector machine is a supervised pattern analysis method used for classifying control and diseased onions in this study. The training data was mapped onto a higher dimensional feature space where the radial basis function (RBF) kernel was be used to construct a separating hyper plane. The separating hyper plane was constructed

with a maximal margin to classify healthy and diseased onion samples. As a result of the training, the developed SVM model was used to classify the second batch of onion data which was completely unknown to the model (represented as B1 (train) & B2 (test)). Prior to training the model, two parameters namely 'C' which is a penalty parameter of the error term and 'γ' which is a RBF kernel parameter was optimized based on the training data. The optimization process was carried by using "grid-search" method in which various combination of 'C' and 'γ' values were used to identify the parameter that led to best classification accuracy. SVM was carried out using LIBSVM library(Chang and Lin, 2007) in Matlab.

Cross-validation models:

The repeatability and robustness of the model was tested by performing 4-fold and leave-one-out cross-validation methods. Using LDA, the four-fold method involved randomly splitting both the batches of onion data into four folds. Hence, the name 4–fold. The classification was carried out 4 times and each time, one of the folds was considered as a testing dataset and the rest of the three folds were used as a training dataset. The successful classification rate of all the four folds was averaged together to get the final cross-validation success rate. In case of SVM, 4-fold cross-validation method was carried by optimizing C and γ values every time each fold was assigned as training data. On the other hand, leave-one-out method involved using only 'n' data points (where n = 1) for testing and the rest (T-n, where n – testing data point and T – total number of data points) for training. The average of all 'T' independent runs was averaged together to obtain the cross validation result.

RESULTS AND DISCUSSION

Data Pre-processing

Noise Filter:

Usually, the raw data obtained as a result was subjected to various types of noises due to the high operating temperatures of the MOS sensors and on-board electronic components. Increase in SNR values for all the MOS sensors were observed as presented in table 3.2 after employing 5-point mean filter.

Table 3.2: Comparison of signal to noise ratio obtained for raw and filtered data for all 7 MOS sensor response curves.

MOS sensors	Raw	Filtered
Tgs 813	56.5	108.74
Tgs 822	44.88	108.24
Tgs 825	47.26	106.07
Tgs 826	36.9	86.8
Tgs 2620	75.74	159.30
SB 11A	52.62	114.29
SB-AQ8	30.91	70.52

Figure 3.2a and 3.2b shows the raw and filtered sensor response of individual MOS sensors. The difference in smoothness in the filtered sensor response (Figure 3.2b) was observed clearly. Complex filters could have been used for removing the noise, but the 5-point mean filter was simple and it effectively removed noise from the raw data.

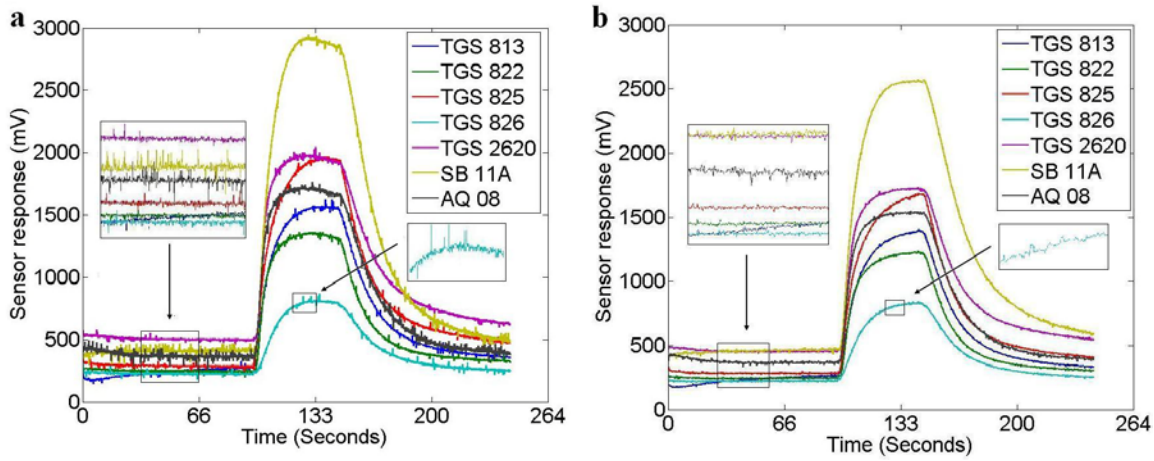

Figure 3.2: Raw sensor response of individual MOS sensors (a) and denoised sensor response (b). The inset figures show the close-up/zoom-in sensor response.

Smellprint:

Individual sensor response obtained by averaging the 7th dai of control and diseased onion data for all the three features (area, relative response and slope) are shown in Figure 3.3a, 3.3b and 3.3c respectively. The difference between the sensor response obtained for control and diseased onions by individual MOS sensor was clearly observed for area, relative response and slope features. By comparing the difference in response among the MOS sensors for all the three features, it was observed that Tgs 826 and SB-AQ8 sensors showed the most difference.

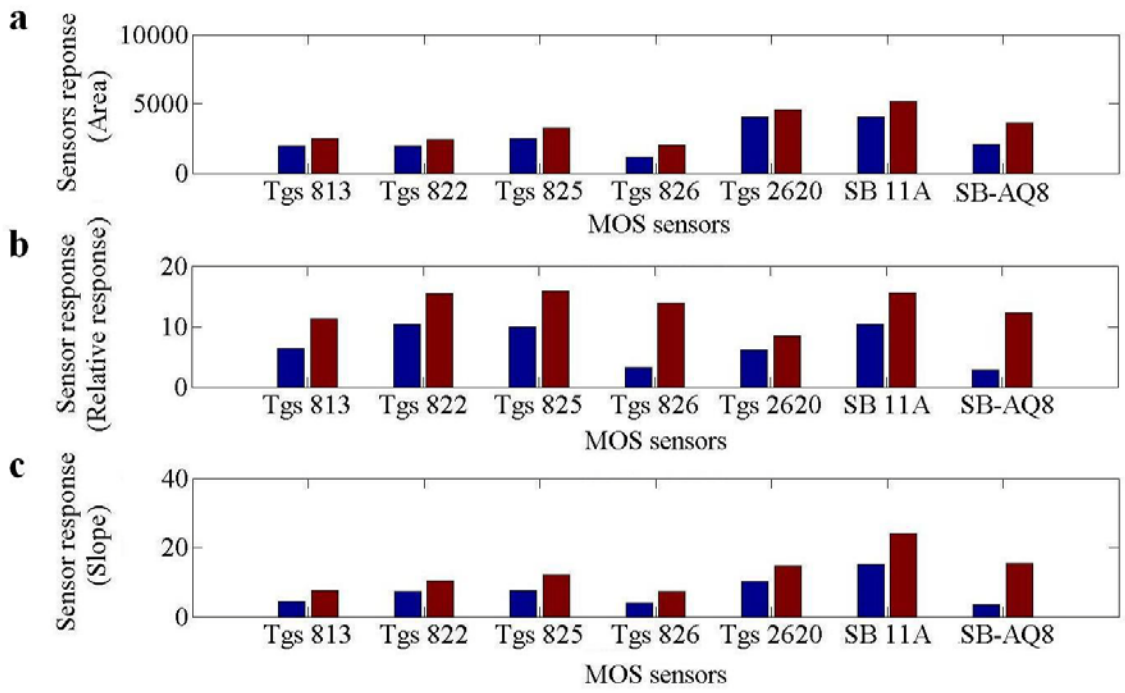

Figure 3.3: Difference in individual MOS sensor response obtained for control and diseased onion data presented for all the three features (area (a), relative response (b) and slope (c))

Baseline correction methods:

The filtered raw data was subjected to baseline correction methods to correct the sensor response with respect to its baseline. The result of employing three different baseline correction (BC) methods (differential, relative and fractional) to the filtered raw data is presented in Figures 3.4(a), 3.4(b) and 3.4(c) respectively. Firstly, the differential BC method subtracted mean baseline response of an individual MOS sensor from its respective response curve. Hence, each of the sensor response started from '0' (approx.). The approximate baseline value was a result of using the averaged sensor response obtained during the last 5 seconds of the baseline purge cycle. This case held

true for all the three BC methods. Secondly, the relative baseline correction method involved dividing the individual MOS sensor response with its respective mean baseline value. Hence, each of the MOS sensor response value started from '1' (approx.). Lastly, based on equation (3), applying the fractional baseline correction method resulted in sensor response starting from the value '0' (approx.).

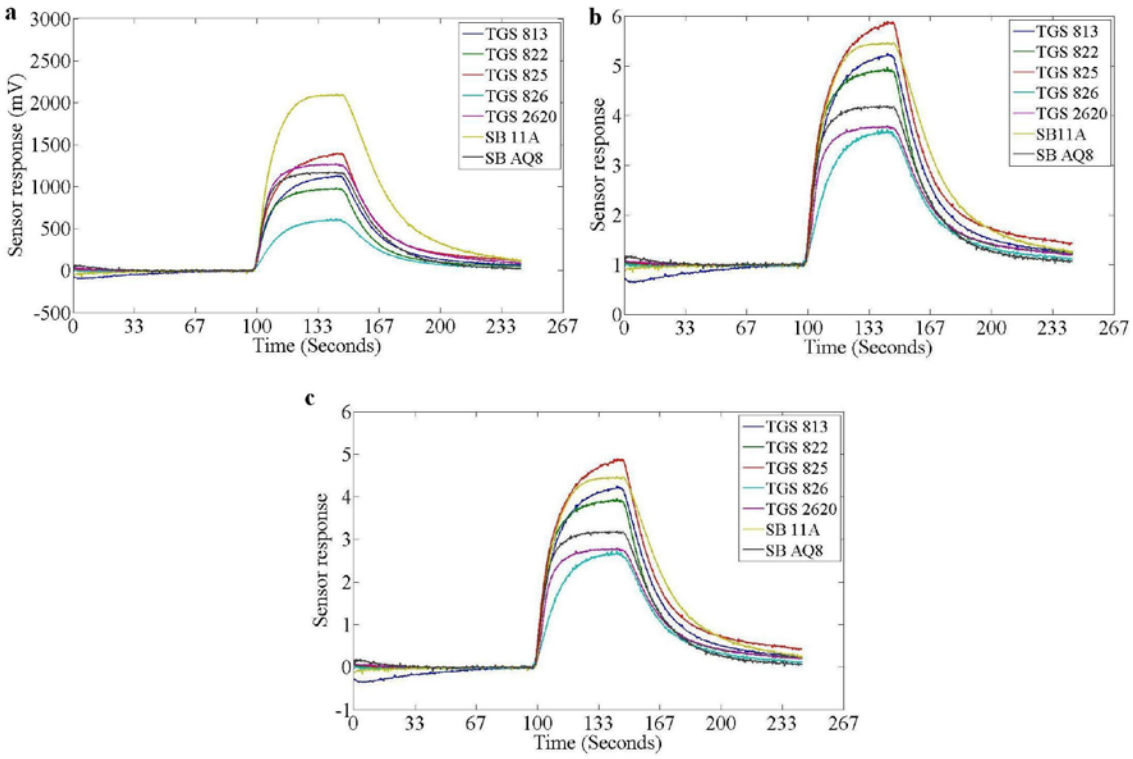

Figure 3.4: Raw data was subjected to three different baseline correction methods namely differential (a), relative (b) and fractional (c) methods.

Analysis based on MANOVA test

Table 3.3 shows three different features corrected with three baseline correction methods for individual days after inoculation and group of days (3-7th dai and 4th–7th dai) together. The relative response feature corrected with relative baseline correction method showed best discrimination ability based on Wilk's lambda and F value

compared to all other baseline corrected features followed by differential baseline corrected area feature. Relative response corrected with relative baseline correction method was the only method which showed significant difference between the control and sour skin infected onion on the 3rd dai.

When consistency was considered, all the features showed consistent increase in the F value with increase in dai except for the relative response feature corrected with differential and fractional methods. Within the relative response feature corrected with relative BC method, F values between the number of dai were compared with each other. A consistent increase in the F value was observed with the increase in the number of dai. This increase was an indication of the disease progressing over the number of days within the infected onion. The F values remained comparatively high after grouping the data from 4th through 7th dai. Grouping 3rd through 7th dai decreased the F value to more than half of the F value obtained by grouping 4th – 7th dai. The measure of variance between the groups reduced considerably with the addition of 3rd dai data in the grouped data. Compared to individual days the F-value obtained for grouped data was 77.4 which was much less compared to the F-value obtained for 7th dai, which was due to combining the data from 4th to 7th dai.

The statistics for the slope feature obtained for both relative and fractional baseline corrected methods were the same since the difference between the two methods was equal to ‘1’ (Fractional – Relative = 1, which can be proved by using equation 2 and 3 in place of relative and fractional, respectively).

Since the difference between the two equations is ‘1’, the parameters extracted from the sensor response subjected to these two methods differed by ‘1’ as well. The slope

feature was calculated using maximum, minimum and time taken to reach maximum values for each MOS sensor response.

This can be well understood with the following example. Let's assume that the maximum values for relative and fractional BC method were 6 and 5 respectively and minimum values were 1.2 and 0.2 respectively. Since the difference between the fractional and relative baseline corrected values are '1', the maximum and minimum values between the baseline correction methods also differed by '1'.

Slope feature was calculated by dividing the difference between the maximum and minimum values by the time taken to attain the maximum value for individual MOS sensors. For both the cases, the time taken to reach the peak was assumed to be 35 seconds.

Using the maximum, minimum and time taken to reach the maximum response resulted in the same value for both relative and fractional BC method which was 0.137. The relative response feature corrected with relative BC method was observed to show the highest difference between the control and diseased onions and hence was used for further analysis.

Table 3.3: Wilk's lambda, F and p values obtained as a result of MANOVA tests for three features (area, slope and relative response) baseline corrected with differential, relative and fractional methods.

		Baseline Correction Methods								
	DAI	Differential			Relative			Fractional		
		Wilk's	F-value	p-value	Wilk's	F-value	p-value	Wilk's	F-value	p-value
Area	3	0.918	0.91	0.505	0.85	1.81	0.099	0.863	1.62	0.1429
	4	0.495	10.46	<.0001	0.556	8.19	<.0001	0.578	7.49	<.0001
	5	0.361	21.74	<.0001	0.558	9.7	<.0001	0.432	16.23	<.0001
	6	0.231	41.68	<.0001	0.277	32.6	<.0001	0.31	26.9	<.0001
	7	0.164	63.91	<.0001	0.234	41.12	<.0001	0.19	53.39	<.0001
	3-7	0.705	26.13	<.0001	0.752	20.5	<.0001	0.766	19.08	<.0001
	4-7	0.438	65.57	<.0001	0.523	46.5	<.0001	0.507	49.55	<.0001
Relative response	3	0.89	1.16	0.337	0.76	3.09	0.006	0.921	0.88	0.525
	4	0.874	1.4	0.1877	0.527	9.2	<.0001	0.878	1.4	0.2119
	5	0.755	3.98	0.0008	0.327	25.2	<.0001	0.759	3.88	0.001
	6	0.88	1.63	0.138	0.19	52.1	<.0001	0.89	1.53	0.1672
	7	0.808	2.98	0.0075	0.110	100.	<.0001	0.82	2.75	0.0125
	3-7	0.902	6.74	<.0001	0.68	29.4	<.0001	0.912	5.99	<.0001
	4-7	0.895	5.94	<.0001	0.397	77.4	<.0001	0.905	5.31	<.0001
Slope	3	0.949	0.55	0.794	0.945	0.59	0.764	0.945	0.59	0.764
	4	0.77	2.94	0.0092	0.788	2.76	0.0134	0.788	2.76	0.0134
	5	0.549	10.08	<.0001	0.565	9.44	<.0001	0.565	9.44	<.0001
	6	0.40	18.71	<.0001	0.465	14.41	<.0001	0.465	14.41	<.0001
	7	0.304	28.7	<.0001	0.289	30.7	<.0001	0.289	30.79	<.0001
	3-7	0.807	14.93	<.0001	0.818	13.8	<.0001	0.818	13.86	<.0001
	4-7	0.583	36.47	<.0001	0.62	31.2	<.0001	0.62	31.29	<.0001

Principal Component Analysis

PRINCIPAL COMPONENT SELECTION AND PCA SCORE PLOT ANALYSIS:

PCA (principal component analysis) was used to investigate the different clusters formed within the multi-dimensional space. Seven principal components were

obtained, although the first few principal components generally held most of the variability in response for analysis. In this case, the first principal component held 91.97% and the second principal component held 3.4% of the response variability. Hence, the first two components were chosen for the following PCA score plots.

With the selection of first two components, the dimensionality was reduced from seven PCs to the two most important PCs. Using the selected two PC's, PCA score plot was obtained from 4^{th} through 7^{th} dai and is presented in the Figure 3.5. The score plot of the relative response feature selected based on the MANOVA results was considered for the analysis. The X-axis and Y-axis was labelled with the percentage of variance observed from the first two principal components. The positioning of the clusters over the period of 4 days indicated the progress of the infection within the diseased onion. In Figure 3.5a, the diseased and healthy onion scores were observed to be overlapping with each other. This could be because the injected pathogen did not infect the onions sufficiently enough for the volatile metabolites to be released, which can be sensed by the device. The healthy and diseased onion cluster was observed to be migrating away from each other starting from 5^{th} dai. With an increase in number of dai, the diseased and healthy onion scores were clustering distinctly from each other. On 7^{th} dai (shown in Figure 3.5d), distinct clusters were observed indicating the difference in the odor released by the healthy and diseased onions.

The lack of compressed clusters for control and sour skin infected onions could be due to the use of multiple onions in each replicate. With multiple diseased onions, the pathogen cannot be strictly controlled to the same level of infection. This could result in varying response from each IOB (Infected onion box) replicate. Understanding

various factors at the molecular level such as the relationship between volatiles produced and bacterial cells would be necessary for future studies.

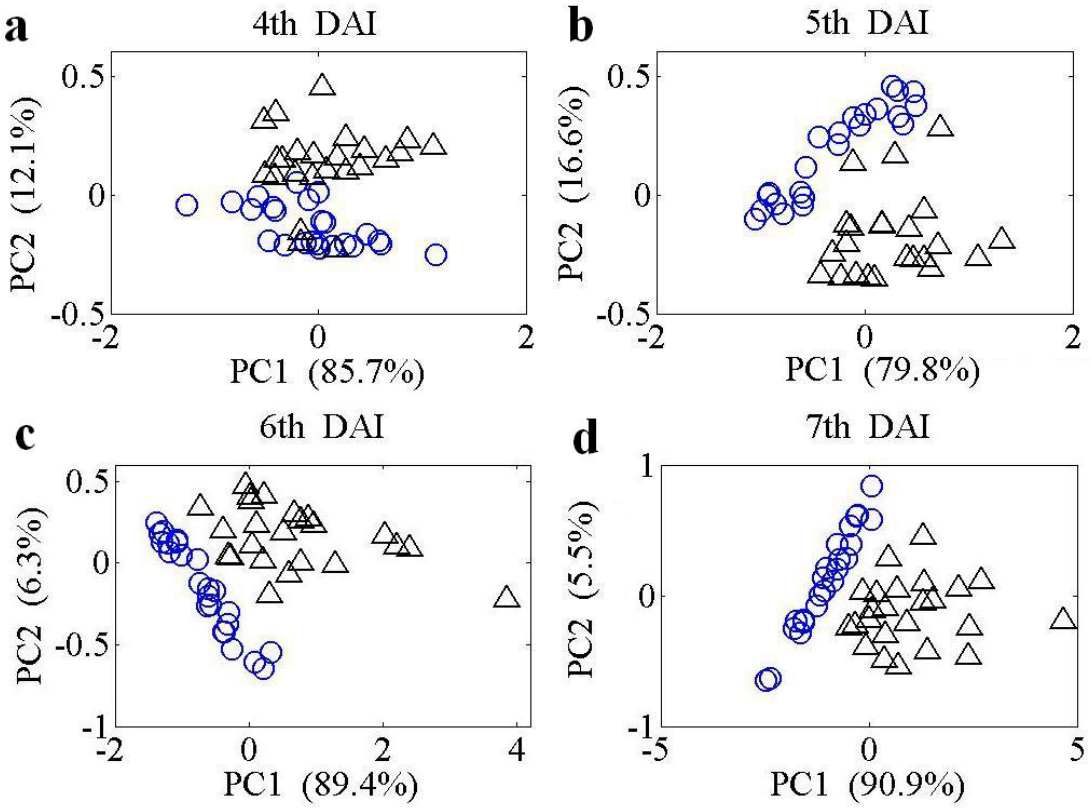

Figure 3.5: PCA score plot obtained for the relative response feature extracted from the sensor response to the onions in COB and IOB. The data was corrected using relative baseline correction method prior to PCA analysis. (a) 4th dai, (b) 5th dai, (c) 6th dai and (d) 7th dai. 'Δ' indicated sensor response to sour skin infected onions mixed with healthy onions and 'o'indicated the control onions mixed with healthy onions.

Selection of sensors

The F values obtained for the individual MOS sensors based on the MANOVA tests are presented in table 3.4. The F values show the significance of each MOS sensor in contributing to diseased onion detection when placed among the healthy onions. The

MANOVA statistical test was performed for individual batches (batch 1 and batch 2) of onion data and combined together from 4^{th} through 7^{th} dai. For all the three cases, Tgs 826 and SB-AQ8 contributed the most to the detection of diseased onions. On the other hand, Tgs 813 was observed to be the worst performer followed by Tgs 2620.

Three different combinations of MOS sensors were selected based on the results. The first combination being all the seven sensors and the second combination of sensors included all the MOS sensors except for the least contributing MOS sensor, which was Tgs 813. The third combination of MOS sensors was selected by removing the two least contributing sensors namely Tgs 813 and Tgs 2620. Tgs 813 was designed to be sensitive to methane, ethane compounds etc. Whereas, the Tgs 2620 sensor was designed to be sensitive to alcoholic and organic solvent vapors. Compared to various types of volatiles released by the sour skin infected onions and control onions, alcoholic volatile compounds are found the least (Li et al., 2011). Whereas, methane and ethane volatile compounds were not recorded in the volatile profile released by healthy and diseased onions. Although this is the case, MOS sensors responded to volatile compounds they are not designed to detect due to their lack of specificity. Hence, Tgs 2620 and Tgs 813 responded to both control and sour skin infected onions however the response was not sufficient enough to discriminate from each other.

The MOS sensors Tgs 826 and SB-AQ8 contributed the most to detecting diseased onions. SB-AQ8 sensor was developed to be sensitive to general volatile organic compounds. This indicated that the sensor was capable of sensing different kinds of volatile organic compounds released by healthy and infected onions. On the other hand, Tgs 826 was designed to be sensitive to ammonia which was not

specifically released by onions for both control and sour skin treatments. The high discrimination ability compared to other MOS sensors could be due to its lack of sensitivity to low thresholds of various types of volatiles released by healthy onions. This could have resulted in detecting the diseased onion odor without being masked by odors released by the group of healthy onions. In Figure 3.2b presented in the earlier sections, it was observed that Tgs 826 sensor response peak was lowest compared to all the other MOS sensors. This observation need not indicate that the lower the MOS sensor response the better it contributed to discriminating the two classes. This could be due to the sensor's lack of sensitivity to a wide range of volatiles released by the control onion. Whereas, in the case of sour skin infected onions, the concentration of volatiles released could have resulted in low response observed in Figure 3.2b. Table 3.4 showed that Tgs 826 and SB-AQ8 has the highest F values indicating their importance which was the same as the results observed in the relative response smellprint presented in Figure 3.3.

Table 3.4: F values obtained for individual MOS sensor response when tested for individual batches of onion data and combined together. The data was considered from 4^{th} dai- 7^{th} dai and the test was performed using MANOVA test.

MOS sensors	Batch 1	Batch 2	Both batches
Tgs 813	89.5	21	89
Tgs 822	112	29	119
Tgs 825	114	47	139
Tgs 826	125	105	225
Tgs 2620	109	28	104
SB 11A	112	29	112
SB-AQ8	138	206	154

Control and Diseased onion Classification

The two treatments consisting of control onions and sour skin infected onions were classified based on linear discriminant analysis and support vector machines. The classification was performed for the sensor responses obtained from 4^{th} through 7^{th} dai which accounts for a total of 366 datasets. Table 3.5 shows the successful classification percentages obtained for three different methods namely 4-fold, leave-one-out and testing the model trained with one batch of onion data with another.

All three sensor classification combinations (7S, 6S and 5S, where S stands for sensor) were presented and compared with each other. The model was trained using the

first batch of onion data and tested with the second batch further validated the repeatability of the results using 4-fold and leave-one-out methods.

Based on batch1 (train) and batch 2(test) LDA results, the sensor response from all the seven sensors contributed to 81.58% of successful classification with 35 misclassifications. All the 35 misclassifications fell under the diseased onion group (onions treated with sour skin disease). Further investigation revealed that out of 35 misclassifications, 21 belonged to 4^{th} dai, 9 to 5^{th} dai and 5 to 6^{th} dai as shown in table 3.6. This shows that the classification performance improved with the disease progressing within the sample. The number of mismatches observed on the 3^{rd} dai was much worse than 4^{th} dai. This was most probably because the infected onions were still in the earlier stages of infection. The 4-fold and leave-one-out methods further cross validated the repeatability of the device performance with 88.24% and 89.89% success rate respectively when all seven sensors were considered. The data for the 4-fold method was chosen randomly and minor changes in success rate with each run were observed.

Between the various combinations of sensors selected based on LDA, it was observed that with a decrease in the number of sensors the success rate was lowered compared to using all seven sensor's response. The cross-validation results obtained using leave-one-out method for 6 sensor (6S) and 5 sensor (5S) selections was observed to be 88.5% and 88.52% respectively. Consistent decrease or increase in successful classification rate was not observed in case of leave-one-out method. However, the difference in success rate between 6S and 5S was observed to be very small.

A similar pattern was observed with SVM classification results when compared to LDA considering the batch1 (test) and batch2 (train) scenario. The optimal values for the parameters 'C' and 'γ' were 16384 and 0.0078 respectively when first batch was selected for training and the second one for testing. A success rate of 85.26%, 81.05 and 75.78% was observed for 7S, 6S and 5S selections respectively showing the importance of using all the seven MOS sensors for this application. The seven sensor selection category had a total of 28 misclassifications which was much smaller compared to LDA. Out of 28 misclassifications, the maximum number of mismatches belonged to 4^{th} dai and a similar pattern was observed in the case of LDA as well. As presented in table 3.6, 19 misclassifications were observed for 4^{th} dai, 6 for 5^{th} dai and 2 for 6^{th} dai.

Cross-validation using 4–fold and leave-one-out methods resulted in more than 90% success rate for all the three sensor selections. 5S selections performed better than the 7S and were almost the same when compared with the 6S selection. However, it was observed that for 4-fold cross validation method, the difference in success rate obtained between the three different sensor selections was less than 0.6%. Similarly, in case of the leave-one-out cross validation method, 5S selections yielded the best classification percentage compared to other two sensor combinations. The difference in success rate in classifying the control and diseased onion was observed to be less than 1%.

For both SVM and LDA methods, based on the number of misclassifications, the maximum classification rate was observed from 5^{th} dai based on the results tabulated in table 3.6. Also, cross-validation tests performed better than individual tests. This was mainly due to the size of the training data allocated to train the model.

Table 3.5: Successful classification percentages obtained for three different sensor groups using LDA and SVM classifiers. 4 fold, leave-one-out method and testing second batch of onion data based on the model trained with first batch validated the results.

Cross - validation	LDA			SVM		
	All Sensors	S2,S3,S4, S5, S6,S7	S2,S3,S4, S6, S7	All Sensors	S2,S3,S4, S5, S6,S7	S2,S3,S, S6, S7
	7 S	6 S	5 S	7 S	6 S	5 S
B1 (train) & B2(Test)	81.58	56.3	43.15	85.26	81.05	75.78
1st fold	91.30	88.04	89.13	90.21	93.47	89.13
2nd fold	85.86	89.13	90.21	91.30	92.39	91.30
3rd fold	93.40	82.41	84.61	92.30	92.3	93.40
4th fold	82.41	82.41	83.51	93.40	91.20	95.60
Average	88.24	86.26	87.63	91.80	92.34	92.36
Leave-one-out	89.89	88.5	88.52	92.35	91.53	92.62

S1 – TGS 813, S2 – TGS 822, S3 – TGS 825, S4 –TGS 826, S5 – TGS 2620, S6 – SB11A AND S7 – SB-AQ8

Table 3.6: Classification model was developed by training it with first batch of onion data and testing it with second batch. The number of incorrect estimations over a period of 4 days (4th – 7th dai) was presented for both the treatments based on LDA and SVM classifiers.

Number of dai	LDA - Mismatches		SVM - Mismatches	
	Control	Diseased (sour skin)	Control	Diseased (sour skin)
4th dai	0	21	0	19
5th dai	0	9	0	6
6th dai	0	5	1	2
7th dai	0	0	0	0
Total Mismatches	0	35	1	27

A relevant study was carried out using a commercially available Cyranose 320 electronic nose (Smith Detection Inc., Pasadena, CA) to detect the presence of sour skin infection within individual sweet onion bulbs (Li et al., 2009). This study will be represented as Cyranose study for convenience. Out of 32 conducting polymer sensors, 6 sensors were selected based on the PC loading values and SVM was carried out from 3^{rd} – 6^{th} dai yielding successful classification accuracy of 85%. In contrast, this study, collected headspace gas from 3 healthy onions and 2 diseased onions was used to detect the presence of infected onions among healthy onions. Seven MOS sensor responses were considered from 4^{th} – 7^{th} dai and achieved a success rate of 85.26%, very close to the Cyranose study when examining only individual diseased and healthy onions. Based on the successful classification rate, the designed device showed promise in identifying the presence of sour skin infected onion when placed among healthy onions.

The difference between both the studies included type of onion used, inoculation procedures, type and number of sensors used. Inoculation in the Cyranose study was conducted by stabbing a sterile wooden toothpick containing a small glob of bacteria at its tip. Whereas in this study, the bacterial innoculum was injected into the onion using a sterile syringe. There are several factors that prohibits from conducting a thorough comparison of studies. Further study with the Cyranose 320 and the customized gas sensing device will be required to compare the performance and potential for diseased onions.

CONCLUSIONS

The noise in each MOS sensor's response to the samples was effectively filtered using 5 point mean filter. Difference in averaged response of individual MOS sensor

was observed for all the features based on the obtained smellprint. MANOVA statistical test confirmed that the relative response feature when corrected with relative baseline correction method performed best compared to combinations of three features and 3 baseline correction methods. Individual day analysis showed significant difference between the volatiles released by control onions and sour skin diseased onions for all the days from 4^{th} dai through 7^{th} dai. A PCA score plot demonstrated distinct sensor response for the volatiles analyzed from COB's (control onion box) and IOB's (infected onion box). The results obtained from 4^{th} and 7^{th} dai showed the progress of bacterial infection in the onions over a period of 4 days. Based on the analysis conducted using MANOVA, Tgs 813 was observed to provide the least significant response followed by Tgs 2620 MOS sensor; whereas, Tgs 826 and SB-AQ8 contributed the most to detection of diseased onions. Based on LDA validation and cross-validation methods, all 7 sensors performed well. On the other hand, picking the 5 sensors with the highest F value performed better compared to other sensor combinations based on SVM validation scenarios except the batch1(train) and batch 2 (test).Classification models were developed using Linear discriminant analysis (LDA) and Support vector machine (SVM) methods for three different sensor selections (7-sensor, 6-sensor and 5-sensor). Based on the classification results, all seven MOS sensors (7-sensor) were observed to be important in classifying diseased and control onions. Validation based on the LDA and SVM models resulted in a 81.58% and 85.26% success rate, respectively. Based on LDA validation and cross-validation models, all the seven MOS sensors discriminated better than the other sensor combinations. However, based on SVM cross validation methods, 5-sensor selection performed slightly better than the other sensor

combinations. Based on the number of incorrect classification estimates for individual day, it was observed that the 4^{th} dai had the maximum number of mismatches. The number of mismatches was observed to be reduced from 5^{th} through 7^{th} dai.

The study proved the reliability and the potential of the customized gas sensor array to detect sour skin infected onions when placed among healthy ones. This is important because the volatiles from the healthy onions did not mask the volatiles from the sour skin infected onions. This unfolds a wide range of applications displaying future potential for non-destructive use of automated post-harvest disease detection in onions during storage.

REFERENCES

Abbey, L., J. Aked, and D. Joyce. 2001a. Discrimination amongst Alliums using an electronic nose. Annals of Applied Biology 139(3):337-342.

Abbey, L., J. Aked, and D. C. Joyce. 2001b. Discrimination amongst Alliums using an electronic nose. Annals of Applied Biology 139(3):337-342.

Abbey, L., D. C. Joyce, J. Aked, B. Smith, and C. Marshall. 2004. Electronic nose evaluation of onion headspace volatiles and bulb quality as affected by nitrogen, sulphur and soil type. Annals of Applied Biology 145(1):41-50.

Aishima, T. 1991. Aroma discrimination by pattern recognition analysis of responses from semiconductor gas sensor array. Journal of Agricultural and Food Chemistry 39(4):752-756.

Arshak, K., E. Moore, G. Lyons, J. Harris, and S. Clifford. 2004. A review of gas sensors employed in electronic nose applications. Sensor Review 24(2):181-198.

Barbara, H. 1996. Solid state, resistive gas sensors. In Handbook of Chemical and Biological Sensors. Taylor & Francis.

Benady, M. 1995. Fruit ripeness determination by electronic sensing of aromatic volatiles. Transactions of the ASABE 38(1):251.

Boothe, D. D. H., and J. W. Arnold. 2002. Electronic nose analysis of volatile compounds from poultry meat samples, fresh and after refrigerated storage. Journal of the Science of Food and Agriculture 82(3):315-322.

Brezmes, J., E. Llobet, X. Vilanova, J. Orts, G. Saiz, and X. Correig. 2001. Correlation between electronic nose signals and fruit quality indicators on shelf-life measurements with pinklady apples. Sensors and Actuators B: Chemical 80(1):41-50.

Chang, C.-C., and C.-J. Lin. 2007. LIBSVM: a library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST) 2(3):27.

de Lacy Costello, B. P. J. 2000. The development of a sensor system for the early detection of soft rot in stored potato tubers. Measurement science & technology 11(12):1685.

Di Natale, C., A. Macagnano, E. Martinelli, R. Paolesse, E. Proietti, and A. D'Amico. 2001. The evaluation of quality of post-harvest oranges and apples by means of an electronic nose. Sensors and Actuators B: Chemical 78(1):26-31.

Gitaitis, R., and E. Tollner. 2005. Experiences with a food product X-ray inspection system for classifying onions. Applied engineering in agriculture 21(5):907-914.

Gómez, A. H. 2006. Evaluation of tomato maturity by electronic nose. Computers and electronics in agriculture 54(1):44.

Gonzalez-Jimenez, J., J. G. Monroy, and J. L. Blanco. 2011. The Multi-Chamber Electronic Nose - An Improved Olfaction Sensor for Mobile Robotics. Sensors 11(6):6145-6164.

Green, G. C., A. D. C. Chan, H. Dan, and M. Lin. 2011. Using a metal oxide sensor (MOS)-based electronic nose for discrimination of bacteria based on individual colonies in suspension. Sensors and Actuators B: Chemical 152(1):21-28.

Gutierrez-Osuna, R. 2002. Pattern analysis for machine olfaction: a review. Sensors Journal, IEEE 2(3):189-202.

Gutierrez, A., J. A. Burgos, C. Garcera, A. I. Padilla, M. Zarzo, C. Chirivella, M. L. Ruiz, and E. Molto. 2007. Optimization of an aroma sensor for assessing grape

quality for wine making. Spanish Journal of Agricultural Research 5(2):157-163.

Haddad, R., A. Medhanie, Y. Roth, D. Harel, and N. Sobel. 2010. Predicting Odor Pleasantness with an Electronic Nose. PLoS Comput Biol 6(4):e1000740.

Kea-Tiong, T., C. Shih-Wen, P. Chih-Heng, H. Hung-Yi, L. Yao-Sheng, and L. Ssu-Chieh. 2010. Development of a Portable Electronic Nose System for the Detection and Classification of Fruity Odors. Sensors (14248220) 10(10):9179-9193.

Klinbumrung, A., K. Jaroensutasinee, S. Pratontep, and T. Kerdcharoen. 2009. Low - cost Electronic nose evaluated on Thai-herb of Northern Thailand samples using multivariate analysis methods. In AIP Conference Proceedings.

Li, C., R. Gitaitis, B. Tollner, P. Sumner, and D. MacLean. 2009. Onion sour skin detection using a gas sensor array and support vector machine. Sensing and Instrumentation for Food Quality and Safety 3(4):193-202.

Li, C., P. Heinemann, and J. Irudayaraj. 2007. Detection of apple deterioration using an electronic nose and znose. Transactions of the ASABE 50(4):1417-1425.

Li, C., G. W. Krewer, P. Ji, H. Scherm, and S. J. Kays. 2010. Gas sensor array for blueberry fruit disease detection and classification. Postharvest Biology and Technology 55(3):144-149.

Li, C., N. E. Schmidt, and R. Gitaitis. 2011. Detection of onion postharvest diseases by analyses of headspace volatiles using a gas sensor array and GC-MS. LWT - Food Science & Technology 44(4):1019-1025.

Mamat, M., S. A. Samad, and M. Hannan. 2011. An Electronic Nose for Reliable Measurement and Correct Classification of Beverages. Sensors 11(6):6435-6453.

Nagle, H. T., and R. Gutierrez-Osuna. 1998. The how and why of electronic noses. IEEE Spectrum 35(9):22.

Panigrahi, S., S. Balasubramanian, H. Gu, C. M. Logue, and M. Marchello. 2006. Design and development of a metal oxide based electronic nose for spoilage classification of beef. Sensors and Actuators B: Chemical 119(1):2-14.

Pearce, T. C. 2003. Handbook of machine olfaction : electronic nose technology / [edited by] T.C. Pearce ... [et al.]. Weinheim [Germany] : Wiley-VCH, c2003.

Rains, G. C., J. Tomberlin, M. D'Alessandro, and W. J. Lewis. 2004. Limits of volatile chemical detection of a parasitoid wasp, Microplitis croceipes, and an electronic nose: a comparative study. Transact Am Soc Agri Engin 47:2145-2152.

Rossi, V., R. Talon, and J.-L. Berdague. 1995. Rapid discrimination of Micrococcaceae species using semiconductor gas sensors. Journal of microbiological methods 24(2):183-190.

Rouseff, R. L., and K. R. Cadwallader. 2001a. Headspace analysis of foods and flavors : theory and practice / edited by Russell L. Rouseff and Keith R. Cadwallader. Advances in experimental medicine and biology: 488. New York : Kluwer Academic/Plenum Publishers, 2001.

Rouseff, R. L., and K. R. Cadwallader. 2001b. Headspace Analysis of Foods and Flavors: Theory and Practice. Springer.

Schwartz, H. F., and S. K. Mohan. 2008. Compendium Of Onion and Garlic Diseases And Pests. Amer Phytopathological Society.

Simon, J. E., A. Hetzroni, B. Bordelon, G. E. MILES, and D. J. CHARLES. 1996. Electronic sensing of aromatic volatiles for quality sorting of blueberries. Journal of food Science 61(5):967-970.

Tang, K.-T., S.-W. Chiu, C.-H. Pan, H.-Y. Hsieh, Y.-S. Liang, and S.-C. Liu. 2010. Development of a Portable Electronic Nose System for the Detection and Classification of Fruity Odors. Sensors 10(10):9179-9193.

Taylor, R. F., and J. S. Schultz. 1996. Handbook of chemical and biological sensors / edited by Richard F. Taylor, Jerome S. Schultz. Bristol ; Philadelphia : Institute of Physics Pub., c1996.

Tollner, E. W., Y.-C. Hung, B. W. Maw, D. R. Sumner, and R. D. Gitaitis. 1995. Nondestructive testing for identifying poor-quality onions.392-402.

USDA-NASS. 2012. 2010-2012 Annual marketing year average prices and value of production of principal costs, Retrieved June 2013, from , http://usda.mannlib.cornell.edu/MannUsda/viewDocumentInfo.do?documentID=1050.

Vernat-Rossi, V., C. Garcia, R. Talon, C. Denoyer, and J.-L. Berdague. 1996. Rapid discrimination of meat products and bacterial strains using semiconductor gas sensors. Sensors and Actuators B: Chemical 37(1):43-48.

Vikram, A., H. Hamzehzarghani, and A. C. Kushalappa. 2005. Volatile metabolites from the headspace of onion bulbs inoculated with postharvest pathogens as a

tool for disease discrimination. Canadian Journal of Plant Pathology 27(2):194-203.

Zampolli, S., I. Elmi, F. Ahmed, M. Passini, G. C. Cardinali, S. Nicoletti, and L. Dori. 2004. An electronic nose based on solid state sensor arrays for low-cost indoor air quality monitoring applications. Sensors and Actuators B: Chemical 101(1-2):39-46.

CHAPTER 4

CONCLUSIONS

A customized gas sensor array was designed to detect the difference in volatiles released by healthy and sour skin diseased onions. Hardware of the sensor was designed to facilitate gas exposure to the MOS sensors and the electronic circuit was designed for automation and data acquisition. The software of the sensor was designed at the microcontroller level to operate the sensor in timely fashion and at computer level for the user to configure and interface with the sensor. Upon testing the device, it showed the potential to detect the presence of diseased onions among healthy ones.

The entire study can be summarized as follows:

1. A stand alone gas sensor array was designed, characterized and tested. The system consisted of mechanical, electronic and software program components. The mechanical component had a gas delivery system to transport the volatiles from the headspace to the MOS sensors docked inside the sensor chamber. The electronic component included a circuit board with a microcontroller, memory chip, real time clock and other peripheral devices. The software programs were at two levels. At the microcontroller level, the program was developed for automation. It included operating the pump, valve in a timely manner, extract features, save data, transfer data to PC and display the status of the device. At the computer level, a graphical user interface (GUI) was developed to configure the sensor, download and analyze the data. The device operated well at high

pump speeds and the location of the MOS sensors within the chamber did not have any effect on respective sensor responses. Sensor was able to discriminate minor concentration differences of four basic chemicals (ethanol, acetone, ethyl acetate and Acetonitrile) which showed the potential of the sensor to discriminate differences in the concentration of the volatiles. Tgs 826 and SB AQ8 MOS sensors responded very well to the 0.5 µL concentrations of volatiles released by the sour skin infected onions (methylpropyl-di-sulfide and 2-nonanone).

2. The device was tested for its capability to detect the presence of diseased onions among healthy ones. Sensor response was recorded for five days in a row to observe the infection progressing within the onions. Three different features namely area, slope and relative response were extracted for every sensor response. Relative response feature corrected with relative baseline correction method was selected based on the MANOVA statistical test procedure. PCA analysis demonstrated distinct clusters for healthy and sour skin infected onions. Moreover, disease progression within the onion samples was observed with the increase in number of days. Tgs 813 and Tgs 2620 MOS sensors were identified to be least contributing sensors whereas, Tgs 826 and SB-AQ8 contributed the most in discriminating the healthy and diseased onions. All the seven MOS sensors were observed to be important for this application based on the linear discriminant analysis (LDA) and support vector machines (SVM) validation models with 81.58% and 85.26% success rate, respectively.

The study proved the reliability and the potential of the fabricated gas sensor array to detect sour skin infected onions when placed among the healthy ones. This is important because the volatiles from the healthy onions did not mask the volatiles from the sour skin infected onions. This unfolds a wide range of applications displaying future potential for non destructive use for automated post harvest disease detection in onions during storage.

FUTURE RESEARCH

Multiple customized gas sensor arrays will be designed and its potential to detect sour skin diseased onions will be tested in a small storage room. The study will be further extended to observe if the sensor can detect *Botrytis alli* infected onions at lab scale and during storage. Additionally, drift effect on the MOS sensors for this application will be studied and appropriate calibration models will be developed.

APPENDICES

Appendix A

Source code of gas sensor

1. Gas sensor source code

```
'*****************************************************************
'*  Name    : Gas sensor program.BAS                                 *
'*  Author  : [Changying Li, Tharun Konduru]                         *
'*  Notice  : Copyright (c) 2010                                     *
'*          : All Rights Reserved                                    *
'*  Date    : 5/17/2012                                              *
'*  Version : 2.8                                                    *
'*  Notes   : Enose PBP program;                                     *
'*          : for 18F4550                                             *
'*****************************************************************

'---'INITIALIZATION SETUP FOR SYSTEM VAIRABLES, REGISTERS,
SYMBOLS--------------
CLEAR
INCLUDE "modedefs.bas"
define OSC 48      'Define the Cystal Speed, 20MHz

' RS232 communication
' ---------initialize USART
define HSER_RCSTA 90h '%10010000    '90h
DEFINE HSER_TXSTA 24h  '%00100000      '20h   ' for high speed, use 24h; low use
20
BAUDCON.3 = 1          'BRG16=1
DEFINE HSER_BAUD 9600
DEFINE HSER_SPBRG 1249   ' sync=0; BRGH=0; BRG16=1
DEFINE HSER_SPBRGH 4     ' sync=0; BRGH=0; BRG16=1

'define HSER_CLROERR
Define LOADER_USED       1

'---------------- pump registers -------------
Define CCP1_REG     PORTC
DEFINE CCP1_BIT     2
'------------Definition of Flags and Threshold Variables'----------------
```

```
flag_0          var bit     ' FOR BASELINE
flag_1          var bit     ' FOR SAMPLING
flag_2          var bit     ' FOR PURGING
flag_3          var bit
'set_time_ok    var bit

purging_t       var byte     ' Define variable for forced purging
purging1_t      var byte    ' Define time variable for baseline purge cycle
purging2_t      var byte    ' Define time variable for purging cycle
sampling_t      var byte    ' Define time variable for Sampling

autorun_min     var byte

a_m             var byte
a_s             var byte

flag_0 = 0   'Initiate baseline purge flag
flag_1 = 0   'Initiate sampling flag
flag_2 = 0   'Initiate purging flag
flag_3 = 0   'Initiate Autorun flag

a_m = 0     ' Initiate value for minute
a_s = 0     ' Initiate value for second

purging_t = 0   ' Initiate value for forced purging
purging1_t = 0  ' Purging time initiate value for baseline purge cycle
purging2_t = 0  ' Purging time initiate value for purging cycle
sampling_t = 0  ' Sampling time value iniating value

' set all pins as output, but RB0 as input for RTC data in. to control pump and valve

TRISB = %00011001
pump var PORTC.2
valve VAR PORTB.5

auto_flag var byte          ' Flag variable for autorun
auto_time_counter var byte    ' Autorun time counter variable
Autorun_switch_1 VAR PORTD.7  ' Autorun switch for 6 hr
Autorun_switch_2 VAR PORTD.6  ' Autorun switch for 12 hr

auto_flag = 0
pump = 0
valve = 0
J var byte
A var word  ' Autorun feature loop variable
```

```
'------------Control word received from interface board--------------------

' I2C Master-Slave communication contains the setup information for the
' sensor.It contains ten bytes.With last byte to determine the type of parameter.
' With certain number of other bytes(LSB)to show the actual control data. See command
' list for detail list.
CON_P            VAR byte       ' Determines what type of control information(The first
byte of CON_BUFF)
'Command List for PC-Interface-Sensor communications: 12 bits
'    11 10 9  8  7  6  5  4  3  2  1  0    Digits
'    Y  Y  M  M  D  D  H  H  M  M  S  S    sync time
'    X  X  X  X  X  X  X  X  X  X  X  3    Download data to PC
'    B  B   B   S   S   S   P   P   P   F   P  4    start sampling and recording data
'   (B: baseline time; S: sampling; P: purging; F: feature; P: pump speed)
'    X  X   X  X  X  X  X  X  X  X  X  5    force sampling
'    X  X   X  X  X  X  X  X  X  X  X  6    force purging
'    X  X   X  X  X  X  X  X  X  X  X  9    Erase Memory

' received data in string from RS232
Rx_Data                  var Byte[12]
Rx_Time                  var byte[12]
I_AUTO                   var byte
I VAR WORD
'Rx_Date var Byte[6]
FOR I = 0 TO 11 STEP 1
   RX_DATA[I] = 0
   RX_TIME[I] = 0
   'hserout [ dec Rx_Data[I]," : ", dec Rx_Time[I],10,13]
NEXT I

'--------------------------Erase VARIABLES-----------------------------------
CLC                      var  byte[64]
COUNTER                  var  word
while_count              var byte
while_count = 0
'------------------------------------------------------------------------------
FOR COUNTER = 0 TO 63 STEP 1
   CLC[COUNTER] = 0
   'hserout [dec clc[counter],10,13]
NEXT COUNTER

' ----------------------I2C port definition-----------------------------
'DPIN                     VAR PORTC.1    'I2C date PIN
'CPIN                      VAR PORTC.0    'I2C Clock PIN
```

```
SCL                 Var PORTC.0         ‘ Clock pin
SDA                 Var PORTC.1         ' Data pin

'ADD                VAR WORD     ' Address for SPI-Memory address operations
ADDR                VAR WORD     ' Address for Array operations
ADDR_LAST           VAR WORD
ADDR_0              VAR WORD
ADDR_INC            CON 27       ' address increment after each data set:  2+1
(feature) +6+9*2.
ADDR_INC5         CON 51       ' IF USE ALL THREE FEATURES WITH 7
SENSORS: 2+1(feature)+6+7*3*2
Index               var word     ' index of the smellprint.
Index_last          var word     ' latest index after sampling

BLOCK             VAR BYTE
BLOCK0            CON %10100000  ' Accessing EEPROM's first block
BLOCK1            CON %10101000  ' Accessing EEPROM's Second Blcok

ADDR = 0                  ' Address, pointers intialize
ADDR_LAST = 0
ADDR_0 = 0
'ADDR_INC = 24               ' address increment after each data set: 24.
Index  = 0
Index_last  = 0

'TRISC = %00000000; set all ports as output;   be careful: RS232 also uses PORTC

'-----------------------sensor data variables------------------------------

S                   VAR WORD[9]   ' SENSOR DATA
S_tym               var word[9]
S_BASE_MEAN         VAR WORD[9]
S_BASE_LAST         VAR WORD[9]
S_MAX               VAR WORD[9]
A0                  VAR WORD[9]
A_DELTA             VAR WORD[9]
A_ALL               VAR WORD[9]
S_RC                VAR WORD[9]
S_AR                VAR WORD[9]
S_SL                VAR WORD[9]
base_manip          var word[9]
True_RH1            var word
True_RH2            var word
True_RH3            var word
```

```
BASE_LOOP       VAR WORD
DELTA_T         var byte   ' 2 SECOND INTERVAL FOR AREA INTEGRATION
DURATION        VAR WORD   ' DURATION FOR BASELINE, SAMPLING,
PURGING

feature var byte           ' feature: 1-4;
pump_spd var byte       ' pump speed 1-2
feature = 2
pump_spd = 1

AUTORUN_N CON 2         ' SAMPLE 1 TIMES FOR EACH AUTORUN

'----------- Sensor electrical configuration --------------------
Vc  var word        ' Vc (Circuit voltage (5V)) for all figaro sensors
Vc = 5000
Sr  var word[9]     ' Sr (Sensor resistance calculated using equation in figaro datasheet)

'Vlr var word       ' Voltage of load resistace. "S" is the variable used in this code
Rl  var word        ' Rl (Load resistance (0.48KOhms) for all figaro sensors
Rl = 1200           '1.2 Kohms (test)

'------------------------format-----------------------------
CR CON 13
LF con 10
'-------------------LCD definition and setup ----------------------
TRISD = %00000000       ' set all pins as output.

' Define LCD registers and bits; current use PORTD
Define  LCD_DREG        PORTD     ' set LCD data port;
Define  LCD_DBIT            0        ' set data starting pin to 4
Define  LCD_RSREG       PORTD     ' set RS register port
Define  LCD_RSBIT           4        ' set RS register bit to 4
Define  LCD_EREG        PORTD     ' set E register port
Define  LCD_EBIT            5        ' set E register bit to 5
define  LCD_BITs            4        ' 4 bit operation;
define  LCD_LINES           2        ' set number of LCD rows
'disp var byte                       ' disp =1 to update display.
Pause 500                            ' Wait for LCD to start up

'-----------------------A/D setups -------------------------------------------
define  ADC_BITS            10      ' Set number of bits in result
Define ADC_CLOCK             3      ' Set clock source (3=rc)
Define ADC_SAMPLEUS     50          ' Set sampling time in uS
'TRISA  = %00001111                 ' Set PORTA.1 as input
TRISA = %11111111                   ' Set all PORTA pins to input
TRISE = %00000111
```

```
'ADCON0 = %11001001        ' SET UP THE CONVERSION CLOCK AND RIGHT
                                JUSTIFED
'ADCON1 = %10000000             ' Set PORTA all analog and right justify result;;
reference voltage Vdd

' try this for 18F
'ADCON0 = %00000111        ' select channel; ADC on.
ADCON1 = %00000111         ' 0111 for 8 analog channels

'ADCON1 = 0                ' 0111 for 8 analog channels
ADCON2 = %10000000         ' right justified.
ADCON0.0 = 1               ' ADC on

"---------------------RTC DS1302 setup-----------------------------------

' -----[ I/O Definitions ]---------------------------------------------

symbol DataIO = PORTB.0            ' data pin
symbol Clock = PORTB.1             ' clock pin
symbol CS1302 = PORTB.2            ' reset pin

' -----[ Constants ]---------------------------------------------------

CWPr          CON     $8E          ' Write Protect Register   10001110
WPr1          CON     $80          ' Set Write Protect        10000000
WPr0          CON     $00          ' Clear Write Protect
WrBurst       CON     $BE          ' Write Burst Of Data    %1011 1110
RdBurst       CON     $BF          ' Read Burst Of Data     %1011 1111
Hr24          CON     0            ' 24 Hour Mode
Hr12          CON     1            ' 12 Hour Mode

' -----[ Variables ]---------------------------------------------------

reg               VAR     Byte     ' Read/Write Address
ioByte            VAR     Byte     ' Data To/From DS1302

'Ticks            VAR WORD         ' Tick count (61 ticks = 1 sec)
Hour              VAR byte         ' Hour variable
Minute            VAR byte         ' Minute variable
Second            VAR byte         ' Second variable
year              var byte         ' year
month             var byte         ' month
date              var byte         ' date
day               VAR byte         ' Day: S-F

'Ticks_1           VAR WORD        ' Tick count (61 ticks = 1 sec)
```

```
Hour_1              VAR byte        ' Hour variable
Minute_1            VAR byte        ' Minute variable
Second_1            VAR byte        ' Second variable
year_1              var byte        ' year
month_1             var byte        ' month
date_1              var byte        ' date

minute0             VAR byte        ' Minute variable
second0             VAR byte        ' Second variable, also used for area feature
second1             var byte        ' second variable used for slope feature
slope_tym           var word        ' dx value obtained for calculating slope
second0_b           var byte
second1_b           var byte
base_tym            var word        ' used for calculating the time in baseline phase to
obtaine base mean values

C_sec var word
k                   var byte
x                   var byte
'------------------- Initiate time variables ----------------------------
Hour = 0
Minute = 0
Second = 0
year = 0
month = 0
date = 0
day = 0

'Ticks_1 = 0
Hour_1 = 0
Minute_1 = 0
Second_1 = 0
year_1 = 0
month_1 = 0
date_1 = 0
base_tym = 0
ampm            VAR     Hour.BIT5       ' AM/PM Flag Bit
clockMode       VAR     Hour.BIT7       ' 12/24 Hour Mode Bit
ampmFlag        VAR     Bit             ' 0 = AM, 1 = PM
modeFlag        VAR     Bit             ' 0 = 24, 1 = 12 (Hours)


   ' register: RTC; PUMP, VALVE.
'TRISB = %00000001  ' RB0=input, RB1=output
'TRISB.3  = 1
'TRISB.4  = 1
```

```
modeFlag = 0    ' 0= 24 h mode; 1 = 12h mode
clockMode = modeFlag  ' 24 h mode    ' this is real bit value in Hour

'******************************  Main
****************************************
'*************************************************************************
*********
 purg_t var byte
 sample_t var byte
 purgin_t var byte
 feat_t var byte
 psp var byte
'---------------------- Start ----------------------------------------
START:

   gosub Get_time         ' obtain current time from DS1302

   if year = 0 then
       " -----[ Initialization ]
      reg = CWPr                      ' Initialize DS1302
      ioByte = WPr0                    ' Clear Write Protect
      GOSUB RTC_Out                        ' Send Command
      gosub Set_Time
   else
   endif
 '---------'Read config into ADDR_LAST; Read from FIRST memory segment AT
LOCATION 1 ----------
      I2CREAD SDA,SCL,BLOCK0,ADDR_0,[ADDR_LAST, Index_last, purging1_t,
      sampling_t, purging2_t, _
                  feature, pump_spd]

   IF ADDR_LAST = 0 THEN                   ' if EEPROM is empty
      ADDR = ADDR_LAST + ADDR_INC          ' AFTER ERASING, ADVANCE
      ADDR_INC BYTES ADDRESS TO WRITE DATA
      Index  = Index_last

      ' write default setup parameters into EEPROM; can be used for AUTORUN
      purging1_t = 50        ' Default time value for baseline purge
      sampling_t = 35        ' Default time value for sampling
      purging2_t = 50        ' Default time value for purging
      feature = 5           ' feature
      pump_spd = 1           ' Pump speed

      ' write Config into EEPROM
      I2Cwrite SDA,SCL,BLOCK0,ADDR_0,[ADDR_LAST, Index_last, purging1_t,
      sampling_t, purging2_t, _
```

```
                feature, pump_spd]
      Pause 10
      I2CREAD SDA,SCL,BLOCK0,ADDR_0,[ADDR_LAST, Index_last, purg_t,
      sample_t, purgin_t,feat_t, psp]
      pause 10
      HSEROUT ["Conf in EEPROM1:", DEC4 ADDR_LAST, " ", DEC4 Index_last, "
       ",DEC3 purg_t, " ",DEC3 sample_t," ", DEC3 purgin_t, " ",_
                DEC feat_t, " ",DEC psp]
   ELSE
      ADDR = ADDR_LAST                  ' IF LAST ADDRSS IS NOT ZERO, JUST
                                         USE THAT ADDRESS
      Index = Index_last
   ENDIF

'==================user input to setup teh Enose ===========
   HSEROUT [lf, cr, "Start Sampling?"]
   LCDOUT $FE,1, "Initialize!"

  '%%%%%%%%%%%%%%%%%% New feature for autorun switch selection
%%%%%%%%%%%%%%%%%%%%%%%%

   LCDOUT $FE,1, " Press Autorun switch"
   HSEROUT [lf, cr, " Press Autorun switch" ]

C_sec = Second
A = 1

TRISD.7 = 1
TRISD.6 = 1

 DURATION = 5
 GOSUB GET_M_S
 GOSUB GET_TIME
 GOSUB CONVERT_DEC

while  flag_3 = 0
         hserout [LF, CR, dec auto_flag, " ","Pausing for selection"]
         pause 1000

         if Autorun_switch_1 = 0 and Autorun_switch_2 = 0 then    ' Autorun the device
                                                                   every one hr
            pause 10
            auto_flag = 1
            auto_time_counter =  0
            hserout [LF, CR, dec auto_flag, " ","1 hr selection detected"]
            GOSUB GET_TIME
```

```
   GOSUB CONVERT_DEC

   autorun_min =  Minute +  1

   if autorun_min = 60  then
      autorun_min = 0
   endif

   Hserout [LF,CR, dec2 Minute," ", dec2 autorun_min]
   gosub Auto_run
else
if Autorun_switch_1 = 1 and Autorun_switch_2 = 0  then   ' Autorun the device
                                                          every 6 hrs
   pause 10
   auto_flag = 6
   auto_time_counter =  0
   hserout [LF, CR, dec auto_flag, " ","6 hr Selection detected"]
   GOSUB GET_TIME
   GOSUB CONVERT_DEC

   autorun_min =  Minute +  1
   if autorun_min = 60     then
      autorun_min = 0
   endif
   Hserout [LF,CR, dec2 Minute," ", dec2 autorun_min]
   gosub Auto_run
else
if Autorun_switch_2 = 1 and Autorun_switch_1 = 0   then     ' Autorun the
                                                      device every 12 hrs
   pause 10
   auto_flag = 12
   auto_time_counter =  0
   hserout [LF, CR, dec auto_flag, " ","12 hr Selection detected"]
   GOSUB GET_TIME
   GOSUB CONVERT_DEC

   autorun_min =  Minute +  1
   if autorun_min = 60   then
      autorun_min = 0
   endif
   Hserout [LF,CR, dec2 Minute," ", dec2 autorun_min]
   gosub Auto_run
endif
endif
endif
```

```
            gosub Get_time  ' get current time from DS1302
            Gosub Convert_DEC
       while_count = while_count + 1     ' User should input time value in 10 seconds
          if while_count = 10  then
            flag_3 = 1
          endif
wend

    HSEROUT [lf, cr, "Start Sampling"]
    LCDOUT $FE,1, " Start Sampling"
    hserin [ str Rx_Data\12]

   ' convert ASCII to DEC
        FOR I = 0 TO 11 STEP 1
             Rx_data[I] = Rx_data[I]-48

             if Rx_data[I] > 9  then '  convert space into 0 in decimal
                 Rx_data[I] = 0
             else
                 Rx_data[I] = Rx_data[I]
             endif
        NEXT I

   CON_P = Rx_data[11]          ' extract control byte; Update the recieved control
                                byte,update CON_P and determine the control data's type
   LCDOUT $FE,1, "control code:", DEC2 CON_P
   Pause 500

'------------------------------ Command execution -----------------------------
    Select Case CON_P
        Case  1                          ' auto run
            autorun_min = Rx_data[0]*10+ Rx_data[1]
            Gosub Auto_run

        Case  3                           ' download from memory

            LCDOUT $FE,1,          "Downloading 4m memory"
            GOSUB Download

        Case  4              ' start sample
            base_tym = 0
            '-------------------reset flags: baseline, sample, purge; 0=run; 1=not run-------
-------------
            flag_0 = 0
            flag_1 = 0
            flag_2 = 0
```

```
        purging1_t = Rx_data[0]*100+ 10*Rx_data[1]+Rx_data[2]
        sampling_t = Rx_data[3]*100+ 10*Rx_data[4]+Rx_data[5]
        purging2_t = Rx_data[6]*100+ 10*Rx_data[7]+Rx_data[8]

        feature = Rx_data[9]        ' feature selection: 1: real time; 2: relative change;
                                      3. area; 4. slope

        pump_spd = Rx_data[10]      ' pump speed

        'gosub cleaning             ' clean the chamber
        gosub pump_on               ' Switch on pump

        gosub valve_fresh           ' Switch valve to allow fresh air
        gosub baseline

        gosub valve_sample          ' Switch valve to allow sample
        gosub sampling

        gosub valve_fresh           ' Switch valve to allow fresh air
        gosub purging

        gosub pump_off              ' Switch off the pump

        hserout ["Entering idle",10,13]
        gosub idle

        if feature = 1 then
           Index = Index       ' do not increment index for real time
        else
           Index = Index + 1   ' increment index number only for
        endif

        gosub Update_ADDR
        hserout ["Done ",10,13]
           'sample_ok = 0


    Case  5              ' set forced sampling time
        sampling_t = Rx_data[0]*100+ 10*Rx_data[1]+Rx_data[2]
        'sampling_t = sampling_t

        HSEROUT [LF, CR]
        hserout ["Forced Sampling... ", DEC3 sampling_t, LF,CR]

        flag_1 = 0
```

```
            gosub pump_on              ' Switch on pump
            gosub valve_sample         ' Switch valve to allow sample
            gosub forced_sampling
            gosub pump_off
        case 6              ' set forced purging time

            purging_t = Rx_data[0]*100+ 10*Rx_data[1]+Rx_data[2]
            HSEROUT [LF, CR]
            'purging_t = purging1_t
            hserout ["Forced Purging... ", DEC3 purging_t, LF,CR]

            flag_0 = 0
            gosub pump_on              ' Switch pump on
            gosub valve_fresh          ' Switch valve to allow fresh air
            gosub forced_purging
            gosub pump_off


        Case  9              ' erase  memory
            hserout ["Erasing...", LF,CR]
            LCDOUT $FE,1, "Erasing..."
            pause 100
            GOSUB Erase                          'Remove Flag for Synchronization
            gosub Update_ADDR

    End Select

GOTO START

' -----------------------------------------Subroutines-----------------------------------------
--------------------------
pump_on:
   HPWm 1, 254, 1221           'Pump speed control - User set
   return

pump_off:
   HPWm 1, 0, 1221             'Pump speed control - User set
   return

valve_fresh:
   low PORTB.5
   return

valve_sample:
   high PORTB.5
   return
```

```
'cleaning:
'   LCDOUt $fe, 1, "Cleaning purge"
'   gosub pump_on
'   gosub baseline
'   gosub sampling
'   gosub pump_off
'   LCDOUt $fe, 1, "Cleaning done"
'   return
"--------------------------auto run----------------------

Auto_run:

   GOSUB Get_Time                    ' Get The Current Date/Time
   gosub CONVERT_dec

   if minute = autorun_min && second = 0 then ' for round hour
 '****************** Autorun TIME SELECTION *************************
          Select case auto_flag
              case 6
                    auto_time_counter = auto_time_counter + 1
                    Hserout [LF,CR, dec auto_time_counter," ",dec autorun_min]
                     if auto_time_counter = 6 or auto_time_counter = 1 then

                       hserout [LF,CR, " 6 hr Autorun"]
                       auto_time_counter =  1
                     else
                       goto Auto_run             ' WAIT
                     endif

              case 12
                    auto_time_counter = auto_time_counter + 1
                    Hserout [LF,CR, dec auto_time_counter," ",dec autorun_min]
                     if auto_time_counter = 12 or auto_time_counter = 1 then

                       hserout [LF,CR, " 12 hr Autorun"]
                       auto_time_counter =  1
                     else
                       hserout [LF,CR, " 12 hr Autorun"," ", dec auto_time_counter]
                       goto Auto_run             'WAIT
                     endif

              CASE 1
                       hserout [LF,CR, " 1 hr Autorun"]
          end SELECT
```

```
      ' be wary of index: local variable vs. global variable (risky: it may be changed in
Subroutines).
      for I_AUTO = 1 to AUTORUN_N Step 1
           '-------------------reset flags: baseline, sample, purge; 0=run; 1=not run------------
--------
           'hserout ["step 3", LF,CR]
          flag_0 = 0
          flag_1 = 0
          flag_2 = 0

          gosub pump_on              ' Switch ON the pump
          gosub valve_fresh          ' Switch the channel to allow fresh air
          gosub baseline
          gosub valve_sample          ' Switch valve to allow sample
          gosub sampling
          gosub valve_fresh          ' Switch valve to allow fresh air
          gosub purging
          gosub pump_off             ' Switch OFF the pump

          gosub idle
          Index = Index + 1   ' increment index number
          gosub Update_ADDR

      next I_AUTO
   else
      GOTO Auto_run            ' wait
   endif

   Goto Auto_run       '  wait
return


"------------' UPDATE LAST MEMORY ADDRESS and sample INDEX--------------
' update address and index only during sampling, not downloading; erasing.
Update_ADDR:
I2CWRITE SDA,SCL,BLOCK0,ADDR_0,[ADDR, Index] ' Write lastest address to
                                                        location ADDR_0 ;
 ' wait for 10 ms after I2C write
for counter = 0 to 9
            pause 1
 next counter
             Counter = 0
return

'----------------------------baseline purging subroutine----------------------
```

```
Baseline:
   DURATION = purging1_t
   GOSUB GET_M_S

  FOR I = 0 TO 8 STEP 1
    S_BASE_MEAN[I] = 0
  NEXT I

  BASE_LOOP = 0   ' Calculate no. of data calculated in last 5 seconds
    'Time information for calculating baseline mean value
 second1_b = (second - 1)

  ' -------------------baseline purge
while flag_0 = 0
     GOSUB ADC_CONV

 ' Slope feature
 if second1_b = 59 then        'if seconds reach 59 do...
    second1_b = 0
 endif

  ' for slope
if second = (second1_b + 1) then
     second1_b = second
     base_tym = base_tym + 1
 else
 endif

  ' different operation for four features
   Select Case feature
      CASE 1
      CASE 2        ' RELATIVE CHANGE

         if  base_tym > (purging1_t - 5)  then   ' Calculate last 5 seconds of the baseline
                                                    to get mean value
                  for I = 0 to 8 step 1
                      S_BASE_MEAN[I] = S_BASE_MEAN[I] + S[I]
                  NEXT I
                     BASE_LOOP = BASE_LOOP + 1
                  endif
      CASE 3  ' area; do nothing
      CASE 4   ' slope

                  FOR I = 0 TO 8 STEP 1
                        S_BASE_LAST[I] = S[I]
                  NEXT I
```

```
        CASE 5      ' USE ALL FEATURES

 if  base_tym > (purging1_t - 5)  then   ' Calculate last 5 seconds of the baseline to get
                                                    mean value
          FOR I = 0 TO 8 STEP 1
                    S_BASE_MEAN[I] = S_BASE_MEAN[I] + S[I]
          NEXT I
                    BASE_LOOP = BASE_LOOP + 1
                    Hserout [ dec BASE_LOOP,10,13]
          endif

          for I = 0 to 8 step 1
                    S_BASE_LAST[I] = S[I]      ' FOR SLOPE
          next I
          end select

    ' display
    GOSUB Display

    ' first judge in case purging time is 0.
          gosub Get_time  ' get current time from DS1302
          Gosub Convert_DEC

           if minute = (minute0 + a_m) && second = a_s  then
                    Flag_0 = 1

    ' different operation for four features

Select Case feature
          CASE 1, 3, 4   ' do nothing

          CASE 2        ' relative change
             FOR I = 0 TO 8 STEP 1
                S_BASE_MEAN[I] = S_BASE_MEAN[I]/BASE_LOOP
             NEXT I
          CASE 5  ' USE ALL FEATURES
             FOR I = 0 TO 8 STEP 1
                S_BASE_MEAN[I] = S_BASE_MEAN[I]/BASE_LOOP
             NEXT I
       end select
    endif
wend
return

'----------------------Sampling, collect data---------------------------
Sampling:
```

```
hserout ["Sampling",10,13]
second0 = 0           ' for area
second1 = 0           ' for slope
slope_tym = 0         ' storing time for calculating slope

   DURATION = SAMPLING_T
   GOSUB GET_M_S


   ' INITIALIZE VARIALBES.
   FOR I = 0 TO 8 STEP 1
      S_MAX[I] = 0          ' FOR RELATIVE CHANGE AND SLOPE
      A0[I] = 0
      A_ALL[I] = 0
      A_DELTA[I] = 0
   NEXT I

  GOSUB GET_TIME
  GOSUB CONVERT_DEC
  SECOND0 = SECOND
  second1 = (second - 1) ' to record time for slope
  J = 0

 while flag_1 = 0
   ' ADC, collect data
   GOSUB ADC_CONV

' Slope feature
 if second1 = 59 then        'if seconds reach 59 do...
   second1 = 0
 endif
   ' for slope
  if second = (second1 + 1) then
     second1 = second
     slope_tym = slope_tym + 1
  else
  endif

-------------------------------------------------------------------------------------------------
 'baseline manipulation (Use S_BASE_LAST[I] value and subtract from the obtained
values)

'     For I = 0 to 7 STEP 1
'       if S[I] > S_base_last[I] then     ' S_BASE_LAST collects the final value in purge
cycle
'       S[I] = S[I]/S_base_last[I]
```

```
'       'hserout ["F ",dec4 S[I],10,13]
'       else
'       S[I] = S_base_last[I] - S[I]
'       'hserout ["L ",dec4 S[I],10,13]
'       endif

'     select case I
'           case 2
'             I = I + 1
'     end select

'    NEXT I
     'hserout ["dur", dec duration,10,13]

  ' initial area values
    While J = 0
    for I = 0 to 8 step 1
       A0[I] = S[I]
    next I
    J = 2
    wend

    Select Case feature
          CASE 1        ' real time response
          ' do nothing

          CASE 2, 4  ' relative change and slope

              ' FIND THE MAXIMUM VALUE
              FOR I = 0 TO 8 STEP 1
                IF S_MAX[I] < S[I] THEN
                   S_MAX[I] = S[I]
                   S_tym[I] = slope_tym
                ENDIF
              NEXT I

          CASE 3      ' area

             GOSUB GET_TIME
             GOSUB CONVERT_DEC

           if second0 = 58 then
             second0 = 0
             DELTA_T = 0
           else
            if second0 = 59   then
```

```
   second0 =0
   DELTA_T = 1
endif
endif

  IF SECOND = SECOND0 + DELTA_T THEN

     FOR I = 0 TO 8 STEP 1
        A_delta[I] = (A0[I] + S[I]) * DELTA_T / 2   '  delta_a = 1/2*(a+b)*t; 2
                                                       seconds interval.
        A_delta[I] = A_delta[I]/10  ' reduce resolution by 10
        A_all[I] = A_all[I] + A_delta[I]

        A0[I] = S[I]
     NEXT I
      x = 1
     GOSUB GET_TIME      ' RENEW SECOND0
     GOSUB CONVERT_DEC
     SECOND0 = SECOND
     delta_t = 2
  ENDIF

  CASE 5  ' USE ALL FEATURES
     GOSUB GET_TIME
     GOSUB CONVERT_DEC

     if second0 = 58 then
        second0 = 0
        DELTA_T = 0
     else
     if second0 = 59   then
        second0 =0
        DELTA_T = 1
     endif
     endif

     IF SECOND = SECOND0 + (DELTA_T * x) THEN

        FOR I = 0 TO 8 STEP 1
           A_delta[I] = (A0[I] + S[I]) * DELTA_T / 2   '  delta_a = 1/2*(a+b)*t;
                                                          2 seconds interval.
           A_delta[I] = A_delta[I]/10  ' reduce resolution by 10
           A_all[I] = A_all[I] + A_delta[I]

           A0[I] = S[I]
        NEXT I
```

```
                x = 1
                GOSUB GET_TIME      ' RENEW SECOND0
                GOSUB CONVERT_DEC
                SECOND0 = SECONd
                delta_t = 2
              ENDIF

              ' FIND THE MAXIMUM VALUE FOR SLOPE AND RELATIVE
            FOR I = 0 TO 8 STEP 1
              IF S_MAX[I] < S[I] THEN
                S_MAX[I] = S[I]
                S_tym[I] = slope_tym
              ENDIF
            NEXT I
    end select

    Gosub Display   ' get time and display data
    Gosub Get_time  ' get current time from DS1302 and convert HEX to DEC
    Gosub Convert_DEC

if minute = (minute0 + a_m) && second = a_s then
        for I = 0 to 8 step 1
            A_delta[I] = (A0[I] + S[I]) * DELTA_T / 2   '  delta_a = 1/2*(a+b)*t; 2
                                                            seconds interval.
            A_delta[I] = A_delta[I]/10  ' reduce resolution by 10
            A_all[I] = A_all[I] + A_delta[I]
        next I
          flag_1 = 1        ' 1 min sampling
    ENDIF
wend
return
'------------------Purging subroutine--------------------
Purging:

    DURATION = PURGING2_T
    GOSUB GET_M_S
    while flag_2 = 0
      ' ADC, collect data
       GOSUB ADC_CONV
       gosub Get_time  ' get current time from DS1302, and convert HEX to DEC
       gosub Convert_DEC

       if minute = (minute0 + a_m) && second = a_s then
          flag_2 = 1        ' 1 min sampling
       endif
```

```
      Gosub Display
   wend
return

' -----------------Idle pump and valve, compute features, save data----------------:
Idle:
   Select Case feature
         CASE 1   ' real time
                  RETURN ' GOBACK TO MAIN PROGRAM

         CASE 2  ' relative change
         '    Calculate relative change of V
            FOR I = 0 TO 8 STEP 1
                  S[I] = (S_MAX[I] - S_BASE_MEAN[I])/S_BASE_MEAN[I]
                  hserout ["RR", dec S[I]," ", dec S_MAX[I]," ", Dec
                  S_BASE_MEAN[I],10,13]
            NEXT I

         CASE 3       ' area
            FOR I = 0 TO 8 STEP 1
                  S[I] = A_all[I]
                  Hserout ["Area ", Dec I, " ", Dec5 S[I],10,13]
            NEXT I

         CASE 4     ' slope
            '    Calculate slope of V
            FOR I = 0 TO 8 STEP 1
                  S_SL[I] = (S_MAX[I] - S_BASE_LAST[I])/s_tym[I]
                   hserout[ dec I, " Slope ", dec5 S_SL[I]," ", dec5 S_MAx[I]," ", dec5
                  S_BASE_last[I]," ", dec slope_tym," ", dec S_tym[I],10,13]
            NEXT I

         CASE 5
            '    Calculate relative change of V
            FOR I = 0 TO 8 STEP 1
                   S_RC[I] = (S_MAX[I] - S_BASE_MEAN[I])/S_BASE_MEAN[I] '
                                                          RELATIVE
                   hserout [dec I, " RC ",dec5 S_RC[I],10,13]
                   S_AR[I] = A_all[I]   ' AREA
                  hserout [ dec I , " area  ", Dec5 A_ALL[I],10,13]
                  S_SL[I] = (S_MAX[I] - S_BASE_LAST[I])/S_tym[I]  ' SLOPE
                  hserout[ dec I, " Slope ", dec5 S_SL[I]," ", dec5 S_MAx[I]," ", dec5
                  S_BASE_last[I]," ",dec5 S_BASE_mean[I]," ", dec S_tym[I],10,13]
            NEXT I
                   hserout ["save data ", dec4 Index," ",hex2 Hour," ", hex2 Minute,"
                  ",hex2 Second,_
```

```
                dec4 S_RC[0]," ",dec4 S_RC[1]," ",dec4 S_RC[2]," ",dec4 S_RC[3],"
                ",dec4
                S_RC[4]," ",dec4 S_RC[5]," ",dec4 S_RC[6], _
                dec5 S_AR[0]," ",dec5 S_AR[1]," ",dec5 S_AR[2]," ",dec5 S_AR[3],"
                ",dec5 S_AR[4]," ",dec5 S_AR[5]," ",dec5 S_AR[6], _
                 dec4 S_SL[0]," ",dec4 S_SL[1]," ",dec4 S_SL[2]," ",dec4 S_SL[3],"
                ",dec4 S_SL[4]," ",dec4 S_SL[5]," ",dec4 S_SL[6], 10,13]
  end select
‘write to memory
                gosub Get_time  ' get current time from DS1302
                Gosub Save_data

        Lcdout $fe, 1,"Done" , HEX2 Hour, ":",HEX2 minute, ":", HEX2 second

return

'----------------------write data into EEPROM----------------------
Save_data:

  IF FEATURE = 5 THEN
      if ADDR < 64000 then
        I2CWRITE SDA,SCL,BLOCK0,ADDR,[Index, feature, year, month, date,
            Hour, Minute, Second,_
          S_RC[0], S_RC[1],S_RC[2], S_RC[3],S_RC[4],S_RC[5], S_RC[6], _
          S_AR[0], S_AR[1],S_AR[2],S_AR[3], S_AR[4],S_AR[5], S_AR[6], _
          S_SL[0], S_SL[1],S_SL[2],S_SL[3], S_SL[4],S_SL[5], S_SL[6]] ' Write
                each location ; access teh second segment

        ' wait for 10 ms after I2C write
        for counter = 0 to 9
          pause 1
        next counter
        ADDR = ADDR + ADDR_INC5

      else    ' use block 1
        ADDR = 0
        I2CWRITE SDA,SCL,BLOCK1,ADDR,[Index, feature, year, month, date,
                Hour, Minute, Second,_
          S_RC[0], S_RC[1],S_RC[2], S_RC[3],S_RC[4],S_RC[5], S_RC[6], _
          S_AR[0], S_AR[1],S_AR[2],S_AR[3], S_AR[4],S_AR[5], S_AR[6], _
          S_SL[0], S_SL[1],S_SL[2],S_SL[3], S_SL[4],S_SL[5], S_SL[6]] ' Write
            each location ; access teh second segment

        ' wait for 10 ms after I2C write
        for counter = 0 to 9
          pause 1
```

```
            next counter
            counter = 0

            ADDR = ADDR + ADDR_INC5
          endif
   ELSE    ' FOR FEATURES  2, 3, 4
      if ADDR < 64000 then

         I2CWRITE SDA,SCL,BLOCK0,ADDR,[Index, feature, year, month, date,
            Hour, Minute, Second,_
            S[0], S[1],S[2],S[3], S[4],S[5], S[6]] ' Write each location ; access teh second
            segment

         ' wait for 10 ms after I2C write
         for counter = 0 to 9
                  pause 1
         next counter
                  counter = 0

                  ADDR = ADDR + ADDR_INC
      else  ' use block 1
                  ADDR = 0
         I2CWRITE SDA,SCL,BLOCK1,ADDR,[Index, feature, year, month, date,
         Hour, Minute, Second,_
            S[0], S[1],S[2],S[3], S[4],S[5], S[6]] ' Write each location ; access teh second
segment

         ' wait for 10 ms after I2C write
         for counter = 0 to 9
                  pause 1
         next counter
                  counter = 0

         ADDR = ADDR + ADDR_INC
      endif
   ENDIF

RETURN

'------------------------Display data
Display:
   ' first judge in case purging time is 0.
   gosub Get_time  ' get current time from DS1302
   hserout [DEC4 Index, " ", HEX2 Hour," ", HEX2 Minute," ", HEX2 second, " ",_
         Dec4 S[0], " ", DEC4 S[1]," ", Dec4 S[2], " ", Dec4 S[3], " ", DEC4 S[4]," ",_
         Dec4 S[5], " ", DEC4 S[6]," ",Dec4 S[7]," ", DEC4 S[8], LF, CR]
```

```
   LCDout $FE,1,HEX2 Hour," ", HEX2 Minute," ", HEX2 second, LF,CR

return

'----------------------convert to DECIMAL
   ‘Convert HEX to DEC
CONVERT_DEC:

   minute = minute/16*10 + minute//16
   second = second/16*10 + second//16

return

'----------------------- ADC DATA COLLECTION AND CONVERSION TO
VOLTAGE----
ADC_CONV:

‘Calculating load resistance (0.48 Kohms)

   For I = 0 to 7 STEP 1
   ADCIN I, S[I]
      S[I] = S[I] * 49
      S[I] = S[I] / 10

    select CASE I
         CASE 2
              I = I + 1
         CASE is >5
         'hserout [ dec S[I],10,13]
            S[I] = 5000 - S[I]
    end select

    NEXT I

   I = 3
   J = 4
    for  I = 3 to 6
         s[I] = S[J]
         J = J + 1
    next I
         s[7] = 0
      ‘Temperature sensor
         Adcin 9, S[7]
         S[7] = (S[7] * 49)
         s[7] = s[7] / 100
```

```
      'Humidity sensor
        Adcin 11, S[8]
       '        S[7] = (S[7] */ 500) >> 2
                    S[8] = (S[8] * 49)
                    s[8] = s[8] / 10
                     s[8] = (s[8] - 826)/ 32

RETURN

'-----------------------
GET_M_S:

       ' get current time
   gosub Get_time  ' get current time from DS1302,  convert HEX to DEC
   Gosub Convert_DEC
   minute0 = minute
   second0 = second
   DURATION = DURATION + second0

   a_m = DURATION/60
   a_s = DURATION//60
RETURN

'----------------------Forced purging-------------------------------
Forced_purging:
   DURATION = purging_t
   GOSUB GET_M_S
' -------------------Baseline purge
       while flag_0 = 0

          gosub Get_time  ' get current time from DS1302
          HSEROUT [HEX2 Hour," ", HEX2 Minute," ", HEX2 second, LF,CR]
          Lcdout $fe,1,HEX2 Hour," ", HEX2 Minute," ", HEX2 second ' Display

          ' first judge in case purging time is 0.
          gosub Get_time  ' get current time from DS1302

          ' convert HEX to DEC
          minute = minute/16*10 + minute//16
          second = second/16*10 + second//16

          if minute = (minute0 + a_m) && second = a_s  then
             flag_0 = 1
             pump = 0   ' turn off pump

          endif
```

```
        wend
return

'----------------------Forced sampling-------------------------------
Forced_sampling:
   DURATION = sampling_t
   GOSUB GET_M_S
' -------------------baseline purge
        while flag_1 = 0

           gosub Get_time  ' get current time from DS1302
           HSEROUT [HEX2 Hour," ", HEX2 Minute," ", HEX2 second,LF,CR]
           Lcdout $fe,1,HEX2 Hour," ", HEX2 Minute," ", HEX2 second ' Display
           ' first judge in case purging time is 0.
           gosub Get_time  ' get current time from DS1302

              ' convert HEX to DEC
           minute = minute/16*10 + minute//16
           second = second/16*10 + second//16

           if minute = (minute0 + a_m) && second = a_s  then
              flag_1 = 1
              pump = 0   ' turn off pump
              valve = 0
           endif
        wend
return

"----------------------write memory---------------------------------

Download:            ' 18cLOCKS

   LCDOUT $FE, 1
   LCDOUT $FE, $80,"Downloading..."

     ‘Download configuration data from EEPROM
    ADDR = 0
    I2CREAD SDA,SCL,BLOCK0,ADDR,[ADDR_LAST, Index_last, purging1_t,
          sampling_t, purging2_t, _
             feature, pump_spd]

  HSEROUT [DEC5 ADDR_LAST, " ", DEC4 Index_last, " ", DEC3 purging1_t, " ",
          DEC3 sampling_t, _
       " ", DEC3 purging2_t, " ", DEC feature, " ", DEC pump_spd, LF,CR]

      ‘Download data   : FIRST BLOCK
```

```
   ADDR = ADDR_INC     ' JUMP OVER RESEARVED SPACE TO SAVE LAST
     ADDRESS
   BLOCK = BLOCK0
   HSEROUT ["Bank 1...", LF, CR]
   gosub READ_MEMORY


   ‘DOWNLOAD SECOND BLOCK DATA
   ADDR = 0
   BLOCK = BLOCK1
   HSEROUT ["Bank 2...",LF, CR]
   GOSUB READ_MEMORY

   ADDR = 0

   Lcdout $fe,1,"Download done"

RETURN

'---------------READ EEPROM MEMORY AND DISPLAY----
READ_MEMORY:

      ‘For other features: to read in one format(27); to enable the following scripts
      For counter = 0 To 2370    ' 2462 x 27 = 64,000 bytes. 64k bytes, first segment.
        I2CREAD SDA,SCL,BLOCK,ADDR.byte1, ADDR.Byte0,[Index, feature,
         year_1, month_1, date_1, Hour_1, minute_1,_
          second_1, S[0], S[1],S[2],S[3], S[4],S[5], S[6]]    ' Read data into B2; read from
         second memory segment

        Lcdout $fe,1,"ADDR:", DEC5 ADDR
        HSEROUT [DEC5 ADDR, " ", DEC4 Index, " ",DEC1 feature, " ", HEX2
         year_1, _
            " ", HEX2 month_1, " ", HEX2 date_1, _
            " ", HEX2 Hour_1, " ", HEX2 minute_1, " ", HEX2 second_1,_
                " ", DEC4 S[0], " ",Dec4 S[1]," ", DEC4 S[2], " ",Dec4 S[3]," ",_
                DEC4 S[4], " ",Dec4 S[5]," ",DEC4 S[6], LF,CR]
        ADDR = ADDR + ADDR_INC
      Next counter
      Counter = 0

RETURN
'---------------------- Erase memory---------------------------------
'---------------Erase everything (including config) or just data? --just data.
Erase:                              ‘Erase the EEPROM

         ‘Erase everything including config data in EEPROM
```

```
        LCDOUT $FE, 1
        LCDOUT $FE, $80,"Erasing 1st block memory..."
        ADDR = 0
        FOR COUNTER = 0 TO 999              ; Erase the first section: 1000 x 64 = 64k.
           I2CWRITE SDA,SCL,BLOCK0,ADDR,[STR CLC\64] ' Write each location ;
           access the second segment
           Pause 10       ' Delay 10ms after each write
                 ADDR = ADDR+64
        NEXT COUNTER
        hserout [" 1st block done",10,13]
        ' erase second block memory
        LCDOUT $FE, 1
        LCDOUT $FE, $80,"Erasing 2nd block memory..."
        ADDR = 0
        FOR COUNTER = 0 TO 999              ; Erase the second section: 1000 x 64 =
                                                    64k.
           I2CWRITE SDA,SCL,BLOCK1,ADDR,[STR CLC\64] ' Write each location ;
           access the second segment
           Pause 10       ' Delay 10ms after each write
                 ADDR = ADDR+64
        NEXT COUNTER
        hserout [" 2nd block done",10,13]
        counter = 0

        LCDOUT $FE, 1
        LCDOUT $FE, $80,"Erase Completed"
        hserout ["Complete",10,13]
        pause 500

        ADDR = 0  ' RESUME STARTING ADDRSS!
        Index = 1
RETURN

'---------------------- Do nothing, Exit---------------------------------
Exit:

RETURN

"--------------------RTC DS1302 routine-------------------------------------------------------
-
' ---- initialize RTC: clear write protection
RTC_Out:
  HIGH CS1302                       ' Select DS1302
  SHIFTOUT DataIO, Clock, LSBFIRST, [reg, ioByte]     'Write Protect Register, clear
write protection
  LOW CS1302                        ' Deselect DS1302
```

```
  RETURN

                        '
Set_Time:   ' set current time info to DS1302 chip.

     ' '==================user input for date info==========
  HSEROUT [lf, cr, "Please input current date in yymmdd:"]
  HSEROUT [LF, CR]
  HSERIN [HEX2 year, HEX2 month, HEX2 date,HEX2 Hour, HEX2 Minute, HEX2
Second]
  'HSEROUT [lf, cr, "please input current hours in hhmmss:"]
  'HSERIN [HEX2 Hour, HEX2 Minute, HEX2 Second]

  HIGH CS1302                   ' Select DS1302
  SHIFTOUT DataIO, Clock, LSBFIRST, [WrBurst]   ' burst write mode
  SHIFTOUT DataIO, Clock, LSBFIRST, [Second, Minute, Hour, date, month, day,
year, 0]
  LOW CS1302                    ' Deselect DS1302
  RETURN


Get_Time:                       ' DS1302 Burst Read
  HIGH CS1302                   ' Select DS1302
  SHIFTOUT DataIO, Clock, LSBFIRST, [RdBurst]
  'SHIFTIN DataIO, Clock, LSBPRE, [HEX Second]
  SHIFTIN DataIO, Clock, LSBPRE, [Second, Minute, Hour, date, month, day, year]
  LOW CS1302                    ' Deselect DS1302
  RETURN

END    'End of the program
```

2. MATLAB Post processing code

```
 %%%%%%%%%%%%%%% POST PROCESSING CODE%%%%%%%%%%%%%
%%%% Enter the batch number %%%%%%%%%%

batch = 1;                  % Enter '1' for first batch and '2' for second batch
filter = 1 ;                % '1' to switch on filter and '2'to switch off filter
baseline = 1;               % '1' to activate baseline manipulation, '0' for no baseline
manipulation
%baseline_select =2;           % '1' for differential, '2' for relative and '3' for fractional
write_text = 0;             % '0' to deactivate writing the filtered data on to the text file
and '1' to do otherwise. ' select path inside the program before switching it on
Manual_baseline = 0;           % enter 0 if you want exact baseline or '1' if you want to
set baseline as '1' after subtracting the minimum value.
%After baseline manipulation, baseline values might tend to be zero.
%Hence instead of setting basevalue as '0', we set a
%common value as 1 to avoid results showing infinite values and to set
%common baseline.
display(' Retreiving data might take from 30-40 seconds')
RR_CL1 = 0;
Slope_CL1 = 0;
Area_CL1 = 0;
RR_CL2 = 0;
Slope_CL2 = 0;
Area_CL2 = 0;
RR_CL3 = 0;
Slope_CL3 = 0;
Area_CL3 = 0;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
for baseline_select = 1:3;
   for batch =  1:2

      clearvars -except Area_b1 Slope_b1 RR_b1 Slope_tym_b1 max_b1 max_b2
max_CL1 max_CL2 max_CL3 mean_CL1 mean_CL2 mean_CL3 mean_b1 mean_b2
batch filter baseline baseline_select write_text Manual_baseline sen raw_sen
filtered_slope_tym Ctym Stym Cbm Sbm Cb Sb RR_CL1 Slope_CL1 Area_CL1
RR_CL2 Slope_CL2 Area_CL2 RR_CL3 Slope_CL3 Area_CL3 c_diff_graph1
s_diff_graph1 c_rel_graph1 s_rel_graph1 c_frac_graph1 s_frac_graph1 c_diff_graph2
s_diff_graph2 c_rel_graph2 s_rel_graph2 c_frac_graph2 s_frac_graph2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% Inititate variables %%%%
C_count=0;                  % Control onion repetition data (Each replicate was
sampled three timees (3 repetitions)
S_count=0;                  % Diseased onion repetition data (Each replicate was
sampled three timees (3 repetitions)
base = 100;                 % Baseline purge value (in seconds)
sample = 50;                % Sample purge value (in seconds)
purge = 100;                % Purge value (in seconds)
S_freq = 14;                % Sampling frequency (varies between 14 and 15 , we are
not using this in the code)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if batch == 1
   A_freq = 0.066;
elseif batch == 2
   A_freq = 0.066;          % Area was calculated with 2 seconds interval ( 1 / (14
(S_freq) * 2))
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
C_F_count = 1;              % Initiate - number of data files for first batch control
onion data
S_F_count = 1;              % Initiate - number of data files for first batch diseased
onion data
C_D_sec = zeros(120,7);
S_D_sec = zeros(120,7);
C_base_max = zeros(120,7);
C_base_mean = zeros(120,7);
C_slope_tym = zeros(120,7);
tx1 = 0;
x_end = 3500;               % final row of each data file.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%% Declare dates for reading onion batch data%%%%%

if batch == 1
   start_date = 5;          % First batch data: Start date Sept '05'th,2012
   end_date = 9;            % First batch data: Start date Sept '09'th,2012
elseif batch == 2
   start_date = 22;         % Second batch data: Start date Sept '22'th,2012
   end_date = 26;           % Second batch data: Start date Sept '26'th,2012
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for selection = 1:2         % '1' indicates control onion data '2' indicates diseased
onion data
```

```
        if selection == 2
           i = 1;
           x = 1;
           sen = 1;
        else
        end
        for i = start_date:end_date        % dates for First batch onion data (05 - 09th day
of September 2012)
           if selection == 1 && batch == 1
              display ('Control data for first batch');
              display(i);
           elseif selection == 1 && batch == 2
              display ('Control data for second batch');
              display(i);
           elseif selection == 2 && batch == 1
              display ('Diseasd onion data for first batch');
              display(i);
           elseif selection == 2 && batch == 2
              display ('Diseasd onion data for second batch');
              display(i);
           end
           for j = 1:8 % 1 to 8 replicates

              if batch == 1
%%%%%%%%%% Open files and extract data %%%%%%%%%%%%%%%%%%
                 if selection == 1
                    a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09052012_Processed_data\Tharunk_Sourskin_box_090';
                    b='2012_3cycles__control_rep';
                    c='_processed.txt';
                    f_name = strcat(a,num2str(i),b,num2str(j),c);
                 elseif selection == 2
                    a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09052012_Processed_data\Tharunk_Sourskin_box_090';
                    b='2012_3cycles__sample_rep';
                    c='_processed.txt';
                    f_name = strcat(a,num2str(i),b,num2str(j),c);
                 end
              elseif batch == 2

%%%%%%%%%% Open files and extract data %%%%%%%%%%%%%%%%%%
                 if selection == 1
                    %display('Control data');
```

```
                    a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09212012_processed_data\Tharunk_Sourskin_box_09';
                    b='2012_3cycles__control_rep';
                    c='_processed.txt';
                    f_name = strcat(a,num2str(i),b,num2str(j),c);
                elseif selection == 2
                    display('sample data');
                    a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09212012_processed_data\Tharunk_Sourskin_box_09 ';
                    b='2012_3cycles__sample_rep';
                    c='_processed.txt';
                    f_name = strcat(a,num2str(i),b,num2str(j),c);
                end
            end

            fid = fopen(f_name);
            tline = fgetl(fid);

            %%%%% initialize variables%%%%%%%%%%
            b_count = 0;
            k = 0;
            j = 0;
            l =0;
            tx= 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            while ischar(tline)

                if j == 1           % register the lines that contain data in the variable "A"
                    k = k + 1;        % A variables rows are counted  using variable "K"

                    tline = fgetl(fid); % code used to read line by line
                    if tline == -1
                        break;
                    end
                    j = 0;
                    if selection == 1
                        tx = find(tline == 'C');   % locate the marker in the raw data file
                    elseif selection == 2
                        tx = find(tline == 'S');   % locate the marker in the raw data file
                    end

                    if tx == 1
```

```
                else
                    if selection == 1
                        C = str2num(tline); % converts the char variable into array.
                    elseif selection == 2
                        S = str2num(tline);
                    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    % arrange repetitions in different orders
                    if l == 0
                        if selection == 1
                            C_rep1(k,:,C_count+1) = C; % variable containing read data
                        elseif selection == 2
                            S_rep1(k,:,S_count+1) = S; % variable containing read data
                        end
                    elseif l == 1
                        if selection == 1
                            C_rep1(k,:,C_count+2) = C;
                        elseif selection == 2
                            S_rep1(k,:,S_count+2) = S;
                        end

                    elseif l == 2
                        if selection == 1
                            C_rep1(k,:,C_count+3) = C;
                        elseif selection == 2
                            %display('entered 3rd cycle');
                            S_rep1(k,:,S_count+3) = S;
                        end
                    else
                    end
                end

            else                % blanks inbetween the data lines are not considered
                tline = fgetl(fid); % for skipping the blank spaces inbetween the data
files.
                j = 1;
                % below code: When marker is found, extract next line in a seperate
                % variable as it contains feature values.
                if tx == 1
                    j = 0;
                    tx = 0;
                    l = l+1;
                    k = 1;
                    %F_count = F_count + 1;
```

```
                    tline = fgetl(fid);
                    F_rep1 = str2num(tline);

                    if selection == 1
                       C_F_count = C_F_count + 1;
                       C_Feature_rep(C_F_count,:) = F_rep1;  % contains feature
values.
                    elseif selection == 2
                       S_F_count = S_F_count + 1;
                       S_Feature_rep(S_F_count,:) = F_rep1;  % contains feature
values.
                    end
                    if l == 3
                       break;
                    end
                 else
                 end
              end

           end

           fclose(fid);
           %display('close file');
           if selection == 1
              C_count=C_count+3;
           elseif selection == 2
              S_count=S_count+3;
           end
        end
     end

     display (' Completed retreiveing the raw data for both batches');
     %%% Completed retreiving the data from the raw data files
%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

     raw_data1 = zeros(4000,7,120);
     c_diff_graph1 = zeros(4000,7,120);
     s_diff_graph1 = zeros(4000,7,120);
     c_diff_graph2 = zeros(4000,7,120);
     s_diff_graph2 = zeros(4000,7,120);
     c_rel_graph1 = zeros(4000,7,120);
     s_rel_graph1 = zeros(4000,7,120);
     c_rel_graph2 = zeros(4000,7,120);
```

```
        s_rel_graph2 = zeros(4000,7,120);
        c_frac_graph1 = zeros(4000,7,120);
        s_frac_graph1 = zeros(4000,7,120);
        c_frac_graph2 = zeros(4000,7,120);
        s_frac_graph2 = zeros(4000,7,120);

        %%%%%%%%%%% extract only sensor data
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%
        for i = 1:120
          b_count = 0;
          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% time
calculations%%%%%%%%%%%%%%%%%%
          %    if C_rep1(1,2,i) == C_rep1(end,2,i)
          %        S_min = C_rep1(end,3,i)- C_rep1(1,3,i);
          %        S_Sec = C_rep1(end,4,i) - C_rep1(1,4,i);
          %        Total_time(i) = (S_min*60) + S_Sec;
          %    else
          %        S_min = (60 - C_rep1(end,3,i))+ C_rep1(1,3,i);
          %        S_Sec = C_rep1(end,4,i) - C_rep1(1,4,i);
          %        Total_time(i) = (S_min*60) + S_Sec;
          %    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
          %%%%%%%%%%% determine the different phases
%%%%%%%%%%%%%%%%%%
          %    xsize = size(C_rep1,1);
          if selection == 1
            C_xsize = size(C_rep1,1);
            raw_data1(1:C_xsize,:,i) = C_rep1(:,5:11,i);
          else
            S_xsize = size(S_rep1,1);
            raw_data1(1:S_xsize,:,i) = S_rep1(:,5:11,i);
          end
        end
        if selection == 2
          C_rep1 = S_rep1;
        else
        end
         display (' Initiate filtering the noise from the data');
        % take individual 2d matrix (sample) and do the math
        for i = 1:120
          Signal = raw_data1(:,:,i);
          if Signal(10,7,1) == 0
            continue;
```

```
        end
        Signal_nz = Signal(any(Signal,2), :);


        %%%%%%%%%%%% 5 point average filter
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        if filter == 1
           for fil_row = 6:(length(Signal_nz)-5)

              for fil_col = 1:7

                 avg_filter = Signal_nz((fil_row),fil_col)+ Signal_nz((fil_row + 1),
fil_col)+ Signal_nz((fil_row + 2), fil_col) + Signal_nz((fil_row + 3), fil_col) +
Signal_nz((fil_row + 4));

                 avg_filtered(fil_row,fil_col) = avg_filter / 5;  %% contains the filtered
data

              end
           end
           avg_filtered = avg_filtered(any(avg_filtered,2),:);

           for fil_row = 6:(length(Signal_nz)-5)

              for fil_col = 1:7
                 if selection == 1
                    C_avg_filtered(fil_row-5,fil_col,i) =(avg_filtered(fil_row-
5,fil_col));
                    x_end = length(C_avg_filtered);

                 else
                    S_avg_filtered(fil_row-5,fil_col,i) =(avg_filtered(fil_row-
5,fil_col));
                    x_end = length(S_avg_filtered);


                 end
              end
           end
        else
        end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  display('Noise has been filtered from the data using 5 point average filter');
```

```
display('**********************************************************************
****');

            %%%%%%%%%%%%% calculate initial and final values of base
%%%%%%%%%%%%%

            initial_sec =  C_rep1(1,4,i);
            F_min = C_rep1(2,3,i);  % take thefirst minute value
            F_sec = C_rep1(2,4,i);  % take the first second value
            B_sec = (base-60)+ F_sec;
            if B_sec > 60
               B_sec = B_sec - 60;
               F_min = F_min + 2;
            else

               F_min = F_min + 1;
            end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            for x = 1:x_end
               %%%%%%%%%%%%%%%% base purging time
readings%%%%%%%%%%%%
               if C_rep1(x,3,i) == F_min && C_rep1(x,4,i)== B_sec - 4
                  base_initial_t = x+1; % sensor response row number 5 seconds before
ending baseline purge
               elseif C_rep1(x,3,i) == F_min && C_rep1(x,4,i)== B_sec
                  base_final_t = x+1;  % Last row of baseline purge
               end
            end

            %%%%%%%%%%%%%%%% Sampling time
readings%%%%%%%%%%%%
            L_min = C_rep1((base_final_t),3,i);
            L_min_updt = C_rep1((base_final_t),3,i);
            L_sec = C_rep1((base_final_t),4,i);
            Sample_sec = sample+ L_sec;
            if Sample_sec > 59
               Sample_sec = Sample_sec - 60;
               L_min = L_min + 1;
            else
            end

            for x = 1:x_end
```

```
            %%%%%%%%%%%%%%%%% sample purging time
%%%%%%%%%%%%%%%%%%%%

            if C_rep1(x,3,i) == L_min && C_rep1(x,4,i)== Sample_sec
               sample_final_t = x;
            else

            end
         end

         %    %%%%%%%%%%%% baseline manipulation
%%%%%%%%%%%%%%%%%%%%%%%%

         % remove empty cells in the variables in which denoised data is stored
%%%%%%%%%%%%%%%%
         if filter == 1 && baseline ==1
            base_final_t = base_final_t - 5;
            sample_final_t = sample_final_t-5;
            %%%%%%%%%%%%%%%%%%%% base value calculations after
filtering %%%%%%%%%%%%%%%%%%%%%%%%%%%%

            if base_initial_t(1,:)  == 0 && base_final_t(1,:) == 0
               break;
            else
               if selection == 1
                  C_base_mean1 = round(avg_filtered(base_initial_t: base_final_t,1:7));
% selection of last 5 secs of baseline purge
                  C_base_mean(i,:) = round(mean(C_base_mean1));
                  %%% Few data points were not retieveing due to
                  %%% issues with real time data. Values were
                  %%% calculated and entered below
                  if i == 102 && batch == 2
                     C_base_mean(102,:) = [254 229 287 221 565 457 375  ];
                  elseif i == 42 && batch == 2
                     C_base_mean(42,:) = [336 340 438 295 701 502 424 ];
                  elseif i == 81 && batch == 2
                     C_base_mean(81,:) = [291 246 319 209 606 461 408];
                     %C_base_max(42,:) = [479 487 651 377 934 793 510 ];
                  end
                  A1 = Signal_nz((base_final_t:sample_final_t),1:7);
                  for x = 1:x_end
                     for sen = 1:7
                        switch (baseline_select)
                           case 1          % Perform differential baseline manipulation
                              C_avg_filter(x,sen,i) = (C_avg_filtered(x,sen,i) -
(C_base_mean(i,sen)));
```

```
                        case 2          % Perform relative baseline manipulation
                            C_avg_filter(x,sen,i) = (C_avg_filtered(x,sen,i) /
(C_base_mean(i,sen)));

                        case 3          % Perform fractional baseline manipulation
                            C_avg_filter(x,sen,i) = (C_avg_filtered(x,sen,i) -
(C_base_mean(i,sen)))/(C_base_mean(i,sen));
                    end
                end
            end
            A = C_avg_filter((base_final_t:sample_final_t),1:7,i);
            C_BM_base(i,:)= mean(C_avg_filter(base_initial_t:
base_final_t,1:7));
        else
            S_base_mean1 = round(avg_filtered(base_initial_t: base_final_t,1:7));
% selection of last 5 secs of baseline purge
            S_base_mean(i,:) = round(mean(S_base_mean1));
            if i == 115 && batch == 2
                S_base_mean(115,:) = [340 250 344 233 574 482 355 ];
            end
            A1 = Signal_nz((base_final_t:sample_final_t),1:7);
            for x = 1:x_end
                for sen = 1:7
                    switch (baseline_select)
                        case 1          % Perform differential baseline manipulation
                            S_avg_filter(x,sen,i) = (S_avg_filtered(x,sen,i) -
(S_base_mean(i,sen)));

                        case 2          % Perform relative baseline manipulation
                            S_avg_filter(x,sen,i) = (S_avg_filtered(x,sen,i) /
(S_base_mean(i,sen)));

                        case 3
                            S_avg_filter(x,sen,i) = (S_avg_filtered(x,sen,i) -
(S_base_mean(i,sen)))/(S_base_mean(i,sen));
                    end

                end
            end
            A = S_avg_filter((base_final_t:sample_final_t),1:7,i);
            S_BM_base(i,:)= mean(S_avg_filter(base_initial_t: base_final_t,1:7));
        end
    end

else
```

```
            %%%%%%%%%%%%%%%%%%%% base value calculations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            if base_initial_t(1,:)  == 0 && base_final_t(1,:) == 0
               break;
            else
               if selection == 1
                  C_base_mean1 = round(Signal_nz(base_initial_t: base_final_t,1:7));
% selection of last 5 secs of baseline purge
                  C_base_mean(i,:) = round(mean(C_base_mean1));

               else
                  S_base_mean1 = round(Signal_nz(base_initial_t: base_final_t,1:7));
% selection of last 5 secs of baseline purge
                  S_base_mean(i,:) = round(mean(S_base_mean1));
               end
            end

            A = Signal_nz((base_final_t:sample_final_t),1:7);
         end


            if batch == 1
               if baseline_select == 1
                  if selection == 1
                     c_diff_graph1 = C_avg_filter(:,:,i);
                  else
                     s_diff_graph1 = S_avg_filter(:,:,i);
                  end
               elseif baseline_select == 2
                  if selection == 1
                     c_rel_graph1 = C_avg_filter(:,:,i);
                  else
                     s_rel_graph1 = S_avg_filter(:,:,i);
                  end
               elseif baseline_select == 3
                  if selection == 1
                     c_frac_graph1 = C_avg_filter(:,:,i);
                  else
                     s_frac_graph1 = S_avg_filter(:,:,i);
                  end
               end
            elseif batch == 2

               if baseline_select == 1
                  if selection == 1
```

```
                c_diff_graph2 = C_avg_filter(:,:,i);
            else
                s_diff_graph2 = S_avg_filter(:,:,i);
            end
        elseif baseline_select == 2
            if selection == 1
                c_rel_graph2 = C_avg_filter(:,:,i);
            else
                s_rel_graph2 = S_avg_filter(:,:,i);
            end
        elseif baseline_select == 3
            if selection == 1
                c_frac_graph2 = C_avg_filter(:,:,i);
            else
                s_frac_graph2= S_avg_filter(:,:,i);
            end
        end
    end


    %%%%%% Calculate slope time values %%%%%%%%%%%%%%%%


    %%%%%%%%%%%%%%%%%%%%%%%%%%%% Maximum values
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    s_final_t = sample_final_t - base_final_t;

    if filter == 1 && baseline == 1


        if selection == 1
            [C_unmanip_max(i,:), C_row(i,:)] = max(A1);       % max value of
sampling for control - acutal max values
            [C_base_max(i,:), dummy_row(i,:)] = max(A);       % max value of
sampling for control - max values after filtering and basline correction
            C_max_row(i,:) = base_final_t + C_row(i,:);

        else
            [S_unmanip_max(i,:), S_row(i,:)] = max(A1);       % max value of
sampling for sampling - acutal max values
            [S_base_max(i,:), S_dummy_row(i,:)] = max(A);       % max value of
sampling for sampling - max values after filtering and basline correction
            S_max_row(i,:) = base_final_t + S_row(i,:);

        end
```

```
        else

            if selection == 1
                [C_base_max(i,:), C_row(i,:)] = max(A);       % max value of sampling
for control
                C_max_row(i,:) = base_final_t + C_row(i,:);

            else
                [S_base_max(i,:), S_row(i,:)] = max(A);       % max value of sampling
for sampling
                S_max_row(i,:) = base_final_t + S_row(i,:);
            end
        end

        %%%%%% slope timing selection %%%%%%%%%%%
        for sen = 1:7
            if selection == 1
                %%%%%% obtain time values at max point for control data
%%%%%%%%
                C_Sam_Sec(i,sen) = C_rep1(C_max_row(i,sen),4,i);
                C_Sam_min(i,sen) = C_rep1(C_max_row(i,sen),3,i);
                %%%%% calculate time values for slope - control data %%%%%

                C1_sec = C_Sam_Sec(i,sen) + L_sec;

                if C_Sam_min(i,sen) > L_min_updt
                    C_D_sec(i,sen) =  (60 - L_sec) + C_Sam_Sec(i,sen);
                else
                    C_D_sec(i,sen) = C_Sam_Sec(i,sen) - L_sec;
                end
            else
                %%%%%% obtain time values at max point for sample data
%%%%%%%%
                S_Sam_Sec(i,sen) = S_rep1(S_max_row(i,sen),4,i);
                S_Sam_min(i,sen) = S_rep1(S_max_row(i,sen),3,i);
                %%%%% calculate time values for slope - sample data %%%%%
                S1_sec = S_Sam_Sec(i,sen) + L_sec;

                if S_Sam_min(i,sen) > L_min_updt
                    S_D_sec(i,sen) =  (60 - L_sec) + S_Sam_Sec(i,sen);
                else
                    S_D_sec(i,sen) = S_Sam_Sec(i,sen) - L_sec;
                end
```

```
        end
      end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      if selection == 1
        C_Area_data = A(1: (s_final_t),:);
        C_F_Area(i,:) = round((trapz(C_Area_data)* A_freq)/10);
      else
        %%%%%%%%%%%%%%%%%Area Feature calculations
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        S_Area_data = A(1: (s_final_t),:);
        S_F_Area(i,:) = round((trapz(S_Area_data)* A_freq)/10);
      end
    end

    if selection == 2
      if baseline == 1 && Manual_baseline == 0
        S_base_mean = S_BM_base;
      end
      S_base_max = S_base_max(any(S_base_max,2),:);
      S_base_mean = S_base_mean(any(S_base_mean,2),:);
      if batch == 2 && filter == 1 && baseline ==1
        %S_base_mean(115,:) = [340 250 344 233 574 482 355 ];
        %S_base_max(115,:) = [660 733 1094 635 1213 1634 929 ];
        S_D_sec(115,:) = [46    27    40    39    33    36    26]; % couldnt retreive
this data from the text file automatically
      elseif batch == 2 && filter == 2 && baseline ==0
        S_base_mean(115,:) = [340 250 344 233 574 482 355 ];
        S_base_max(115,:) = [660 733 1094 635 1213 1634 929 ];
        S_D_sec(115,:) = [46    27    40    39    33    36    26]; % couldnt retreive
this data from the text file automatically
      end
      S_D_sec = S_D_sec(any(S_D_sec,2),:);
      S_slope_tym = S_D_sec;
      %S_BM_base = S_BM_base(any(S_BM_base,2),:);
    elseif selection == 1
      if baseline == 1 && Manual_baseline == 0
        C_base_mean = C_BM_base;
      end
      C_base_max = C_base_max(any(C_base_max,2),:);
      C_base_mean = C_base_mean(any(C_base_mean,2),:);
      if batch == 2 && filter == 1 && baseline ==1
        % C_base_mean(101,:) = [254 229 287 221 565 457 375  ];
        %C_base_max(101,:) = [397 364 471 282 783 728 453];
```

```
                C_D_sec(102,:) = [39    35    44    1    26    45    45]; % couldnt retreive this
data from the text file automatically
                C_D_sec(42,:) = [34 43 37 41 41 46 19]; % couldnt retreive this data from
the text file automatically

            elseif batch == 2 && filter == 2 && baseline ==0
                C_base_mean(101,:) = [254 229 287 221 565 457 375  ];
                C_base_max(101,:) = [397 364 471 282 783 728 453];
                C_D_sec(102,:) = [39    35    44    1    26    45    45]; % couldnt retreive this
data from the text file automatically

                C_base_mean(42,:) = [336 340 438 295 701 502 424 ];
                C_base_max(42,:) = [479 487 651 377 934 793 510 ];
                C_D_sec(42,:) = [34 43 37 41 41 46 19]; % couldnt retreive this data from
the text file automatically

                C_base_mean(80,:) = [291 246 319 209 606 461 408];
                C_base_max(80,:) = [496 553 717 299 1053 1121 531 ];
                C_D_sec(80,:) = [31 30 44 30 43 29 17]; % couldnt retreive this data from
the text file automatically
            end
            C_slope_tym = C_D_sec;
            C_slope_tym = C_slope_tym(any(C_slope_tym,2),:);

            %C_BM_base = C_BM_base(any(C_BM_base,2),:);
        end


    end
    display(' Completed noise filtration, baseline correction and paramerter
calculatoin');
    %%% manual entry of data due to incorrect slope time for one of the sensor
    C_slope_tym(80,5) = 1;
    C_slope_tym(104,5) = 1;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%% Write filtered data into text file %%%%%%%%%
    if write_text == 1
        C_avg_filtered = round(C_avg_filtered);
        i =0;
        for selection = 1:2
            for l = start_date:end_date
                for j = 1:8
                    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Write data in
text files %%%%%%%%%%%%%%%%%%
                    if batch == 1
                        i = i+1;
```

```
                    if selection == 1
                        a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09052012_filtered\Tharunk_Sourskin_box_090';
                        b='2012_3cycles__control_rep';
                        c='_filtered.txt';
                        f_name = strcat(a,num2str(l),b,num2str(j),c);

                        [nrows,ncols]= size(C_avg_filtered(:,:,i));
                        fid = fopen(f_name, 'w');
                        for row=1:nrows
                            fprintf(fid, '%d %d %d %d %d %d %d\n',
C_avg_filtered(row,:,i));
                        end
                        fclose(fid);
                    elseif selection == 2
                        a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09052012_filtered\Tharunk_Sourskin_box_090';
                        b='2012_3cycles__sample_rep';
                        c='_filtered.txt';
                        f_name = strcat(a,num2str(l),b,num2str(j),c);
                        [nrows,ncols]= size(S_avg_filtered(:,:,i));
                        fid = fopen(f_name, 'w');
                        for row=1:nrows
                            fprintf(fid, '%d %d %d %d %d %d %d\n',
S_avg_filtered(row,:,i));
                        end
                        fclose(fid);
                    end
                elseif batch ==2
                    i = i+1;
                    if selection == 1
                        a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09212012_filtered\Tharunk_Sourskin_box_09';
                        b='2012_3cycles__control_rep';
                        c='_filtered.txt';
                        f_name = strcat(a,num2str(l),b,num2str(j),c);

                        [nrows,ncols]= size(C_avg_filtered(:,:,i));
                        fid = fopen(f_name, 'w');
                        for row=1:nrows
                            fprintf(fid, '%d %d %d %d %d %d %d\n',
C_avg_filtered(row,:,i));
                        end
```

```
                    fclose(fid);
                elseif selection == 2
                    a='C:\Documents and
Settings\konduru9\Desktop\Tharun_Konduru\Enose_projectfiles\Experimental_Data_K
TK\Tharunk_sourskin_box_09212012_filtered\Tharunk_Sourskin_box_09';
                    b='2012_3cycles__sample_rep';
                    c='_filtered.txt';
                    f_name = strcat(a,num2str(l),b,num2str(j),c);
                    [nrows,ncols]= size(S_avg_filtered(:,:,i));
                    fid = fopen(f_name, 'w');
                    for row=1:nrows
                        fprintf(fid, '%d %d %d %d %d %d %d\n',
S_avg_filtered(row,:,i));
                    end
                    fclose(fid);
                end
            end
        end

    end
  end
end

if batch == 1
    final_i = 104;
elseif batch == 2
    final_i = 119;
end

for selection = 1:2
    for i = 1:final_i
        if baseline == 1 && Manual_baseline == 1
            S_base_mean(i,:) = 1;
            C_base_mean(i,:) = 1;
        else
        end
        for sen = 1:7
            %Slope_tym(i,sen) = C_row(i,sen) - base_final_t+5;
            %           if selection == 1
            %               Slope_tym(i,sen) = C_row(i,sen);
            %           else
            %               Slope_tym(i,sen) = S_row(i,sen);
            %           end
            %               Slope_tym(i,sen) = (Slope_tym(i,sen)) / S_freq;
            %               Slope_tym(i,sen) = round(Slope_tym(i,sen))+1;
```

```
            if selection == 1
               %            if batch == 2 && baseline == 1
               %
               %               if baseline_select == 1
               %                 C_base_mean(80,:) = [291 246 319 209 606 461 408];
               %                  C_base_max(80,:) = [496 553 717 299 1053 1121 531 ];
               %
               %                  C_base_mean(104,:) = [291 246 319 209 606 461 408];
               %                  C_base_max(104,:) = [496 553 717 299 1053 1121 531
];
               %               end
               %             end
               %C_F_slope(i,sen) = ((C_base_max(i,sen) - C_base_mean(i,sen))/
C_slope_tym(i,sen)); %calculate relative response for control
               C_F_slope(i,sen) = (C_base_max(i,sen)- C_base_mean(i,sen))/
C_slope_tym(i,sen);
               C_F_RR(i,sen) = (((C_base_max(i,sen) - C_base_mean(i,sen)))/
C_base_mean(i,sen)); %calculate relative response for control


            else

               S_F_slope(i,sen) = (S_base_max(i,sen) - S_base_mean(i,sen))/
S_slope_tym(i,sen); %calculate relative response for sample
               S_F_RR(i,sen) = (((S_base_max(i,sen) - S_base_mean(i,sen)))/
S_base_mean(i,sen)); %calculate relative response for sample


            end
         end

      end

   end
   Slope_tym_x = cat(1,C_slope_tym,S_slope_tym); % concatenate control and
diseasesd slope time values
   max_x = cat(1,C_base_max,S_base_max);        % Concatenate maximum values
   mean_x = cat(1,C_base_mean,S_base_mean);     % Concatenate minimum values
   C_F_Area =  C_F_Area(any(C_F_Area,2),:);
   S_F_Area =  S_F_Area(any(S_F_Area,2),:);
   Area_x = cat(1,C_F_Area,S_F_Area);
   %%%%%%%% Slope %%%%%%%%%%%
   Slope_x = cat(1,C_F_slope,S_F_slope);
   %%%%%%%% RR %%%%%%%%%%%%
   C_F_RR =  C_F_RR(any(C_F_RR,2),:);
   S_F_RR =  S_F_RR(any(S_F_RR,2),:);
```

```
    RR_x = cat(1,C_F_RR,S_F_RR);

    if batch == 1
        Area_b1 = Area_x;
        Slope_b1 = Slope_x;
        RR_b1 = RR_x;

        %Filtered_data_b1 = Filtered_data_x;
        %Control_raw_b1 = Control_raw_x;
        %Manipulated_Data_b1 = Manipulated_Data_x;
        Slope_tym_b1 = Slope_tym_x;
        max_b1 = max_x;
        mean_b1 = mean_x;
    elseif batch == 2
        Area_b2 = Area_x;
        Slope_b2 = Slope_x;
        RR_b2 = RR_x;

        Slope_tym_b2 = Slope_tym_x;
        %Filtered_data_b2 = Filtered_data_x;
        % Control_raw_b2 = Control_raw_x;
        %Manipulated_Data_b2 = Manipulated_Data_x;
        max_b2 = max_x;
        mean_b2 = mean_x;
    end
end

%%%%%%%%%%%    %%% LDA classification %%%%%%%%%%%%%%%

Area_CL = cat(1, Area_b1,Area_b2);
Slope_CL = cat(1, Slope_b1, Slope_b2);
RR_CL = cat(1, RR_b1, RR_b2);
Max_CL = cat(1, max_b1, max_b2);
Mean_CL = cat(1, mean_b1, mean_b2);

display (' Commence LDA classification');
sb1_A = size(Area_b1);
sb2_A = size(Area_b2);

sb1_S = size(Slope_b1);
sb2_S = size(Slope_b2);

sb1_R = size(RR_b1);
sb2_R = size(RR_b2);

Area_CL(1:(sb1_A/2),8) = 0;
```

```
    Area_CL((sb1_A/2)+1:(sb1_A),8) = 1;
    Area_CL((sb1_A)+1:sb1_A+(sb2_A/2),8) = 0;
    Area_CL(sb1_A+(sb2_A/2)+1:sb1_A+sb2_A,8) = 1;

    Slope_CL(1:(sb1_S/2),8) = 0;
    Slope_CL((sb1_S/2)+1:(sb1_S),8) = 1;
    Slope_CL((sb1_S)+1:sb1_S+(sb2_S/2),8) = 0;
    Slope_CL(sb1_S+(sb2_S/2)+1:sb1_S+sb2_S,8) = 1;

    RR_CL(1:(sb1_R/2),8) = 0;
    RR_CL((sb1_R/2)+1:(sb1_R),8) = 1;
    RR_CL((sb1_R)+1:sb1_R+(sb2_R/2),8) = 0;
    RR_CL(sb1_R+(sb2_R/2)+1:sb1_R+sb2_R,8) = 1;

    if baseline_select == 1
        RR_CL1 = RR_CL;
        Slope_CL1 = Slope_CL;
        Area_CL1 = Area_CL;
        max_CL1 = Max_CL;
        mean_CL1 = Mean_CL;
    elseif baseline_select == 2
        RR_CL2 = RR_CL;
        Slope_CL2 = Slope_CL;
        Area_CL2 = Area_CL;
        max_CL2 = Max_CL;
        mean_CL2 = Mean_CL;
    elseif baseline_select == 3
        RR_CL3 = RR_CL;
        Slope_CL3 = Slope_CL;
        Area_CL3 = Area_CL;
        max_CL3 = Max_CL;
        mean_CL3 = Mean_CL;
    end
    clear Area_CL Slope_CL RR_CL
end

%BM = baseline_select;
DataArr = [];
err_count = 0;
err_C = 0;
err_D = 0;
%feature = 1;
%day =5;
%BM = 3;
%%%%%% data read into Matlab %%%%%%%%%
for BM =1:3
```

```
    for feature = 1:3
        for day = 0:8
            if BM == 1
                if feature == 3
                    data = RR_CL1;
                    %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_RR_differential_bothbatches.xls',1);
                elseif feature == 2
                    data = Slope_CL1;
                    %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_Slope_differential_bothbatches.xls',1);
                elseif feature == 1
                    data = Area_CL1;
                    %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_Area_differential_bothbatches.xls',1);
                end

            elseif BM == 2
                if feature == 3
                    data = RR_CL2;
                    %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_RR_relative_bothbatches.xls',1);
                elseif feature == 2
                    data = Slope_CL2;
                    %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_Slope_relative_bothbatches.xls',1);
                elseif feature == 1
                    data = Area_CL2;
                    %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_Area_relative_bothbatches.xls',1);
                end
            elseif BM == 3
                if feature == 3
                    data = RR_CL3;
                    %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_RR_fractional_bothbatches.xls',1);
                elseif feature == 2
                    data = Slope_CL3;
```

```
            %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_Slope_fractional_bothbatches.xls',1);
        elseif feature == 1
            data = Area_CL3;
            %data =
xlsread('Z:\TharunKonduru\Enose_projectfiles\Experimental_Data_KTK\Tharunk_sour
skin_box_09052012\Tharunk_Area_fractional_bothbatches.xls',1);
        end
    end

    %%%%%% LDA classification %%%%%%%%%%
    Enose_data = data;
    col_used = 1:7; % 1:7; % [4,7]Enose_data = data;
    col_flag = 8;
    Q = size(Enose_data,1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    switch day
        case 1
            ind_Tr = [1:16, 105:120]; % Last day
            ind_Test = [209:232,328:351];
        case 2
            ind_Tr = [17:32, 121:136]; % Last day
            ind_Test = [233:256, 352:375];
        case 3
            ind_Tr = [33:55, 137:159]; % Last day
            ind_Test = [257:279, 376:398];
        case 4
            ind_Tr = [57:80, 161:184]; % Last day
            ind_Test = [280:303, 399:422];
        case 5
            ind_Tr = [81:104, 185:208]; % Last day
            ind_Test = [304:327, 423:446];
        case 0
            ind_Tr = [1:208];
            ind_Test = [209:Q];
        case 6
            ind_Tr = [57:104, 161:208]; % Last two days
            ind_Test = [280:327, 399:446];
        case 7
            ind_Tr = [33:104, 137:208]; % Last two days
            ind_Test = [257:327, 376:446];
        case 8
            ind_Tr = [17:104, 121:208]; % Last two days
```

```
        ind_Test = [233:327, 352:446];
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%

    data_train = Enose_data(ind_Tr, col_used);
    data_test = Enose_data(ind_Test, col_used);
    Q1 = (size(data_train ,1)+size(data_test,1));
    class_train = Enose_data(ind_Tr,col_flag);
    class_test = Enose_data(ind_Test,col_flag);

    [final_class,error,POSTERIOR,logP,
coeff]=classify(data_test,data_train,class_train,'linear','empirical');

    Result_comparison = [final_class,class_test];
    %%% Calculate error %%%%%%%%%%%%%%%
    for i =  1:size(data_test,1)
      if final_class(i,1) == class_test(i,1)
      else
        err_count = err_count + 1;
      end
    end

    for i = 1:size(class_test,1)
      if class_test(i,1) == 0;
        if final_class(i,1) == class_test(i,1)
        else
          err_C = err_C + 1;
        end
      elseif class_test(i,1) == 1;
        if final_class(i,1) == class_test(i,1)
        else
          err_D = err_D + 1;
        end
      end
    end
    count_C = err_C/size(class_test,1)/2;
    error_C = 100 - (count_C * 100);


    count_D = err_D/size(class_test,1)/2;
    error_D = 100 - (count_D * 100);
    error_rate = err_count/ size(data_test,1);
    error_rate = 100 - (error_rate * 100)
```

```
        if BM == 1
            DataArr = vertcat(DataArr, [Q1 BM feature day error_rate err_count err_C
err_D error_C error_D]);
        elseif BM == 2
            DataArr = vertcat(DataArr, [Q1 BM feature day error_rate err_count err_C
err_D error_C error_D]);
        elseif BM == 3
            DataArr = vertcat(DataArr, [Q1 BM feature day error_rate err_count err_C
err_D error_C error_D]);
        end
        err_count = 0;
        err_C  = 0;
        err_D = 0;
        error_C = 0;
        error_D = 0;
      end
   end
end

%%%%%%%%%%%%%%%%%END%%%%%%%%%%%%%%%%%%%%
```