**Naval Research Laboratory**

**Washington, DC 20375-5320**

**NRL/MR/6930--18-9812**

# Practical Implementation of Detection Algorithm for Reflectance-Based, Real-Time Sensing

Jeffrey S. Erickson
Anthony P. Malanoski
Brandy J. White
David A. Stenger

*Laboratory for the Study of Molecular Interfacial Interactions*
*Center for Bio/Molecular Science & Engineering*

Earl Tankard, Jr.

*Howard University*
*Washington, DC*

October 3, 2018

# REPORT DOCUMENTATION PAGE

*Form Approved*
*OMB No. 0704-0188*



| **1. REPORT DATE** *(DD-MM-YYYY)* | **2. REPORT TYPE** | **3. DATES COVERED** *(From - To)* |
|---|---|---|
| 03-10-2018 | Memorandum Report | 9/30/2013 - 08/25/2018 |

**4. TITLE AND SUBTITLE**

Practical Implementation of Detection Algorithm for Reflectance-Based, Real-Time Sensing

**5a. CONTRACT NUMBER**

**5b. GRANT NUMBER**

**5c. PROGRAM ELEMENT NUMBER**

**6. AUTHOR(S)**

Jeffrey S. Erickson, Anthony P. Malanoski, Brandy J. White, David A. Stenger and Earl Tankard, Jr.*

**5d. PROJECT NUMBER**

**5e. TASK NUMBER**

**5f. WORK UNIT NUMBER**
69-6A26-08

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

Center for Bio/Molecular Science & Engineering
Naval Research Laboratory
4555 Overlook Avenue, SW
Washington, DC 20375-5344

**8. PERFORMING ORGANIZATION REPORT NUMBER**

NRL/MR/6930--18-9812

**9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Defense Threat Reduction Agency
DTRA-Joint CBRN Center of Excellence
BLDG E-2800
APG-EA, 21010

**10. SPONSOR / MONITOR'S ACRONYM(S)**

NRL 6.2

**11. SPONSOR / MONITOR'S REPORT NUMBER(S)**

**12. DISTRIBUTION / AVAILABILITY STATEMENT**

**DISTRIBUTION STATEMENT A**: Approved for public release; distribution is unlimited.

**13. SUPPLEMENTARY NOTES**

*NRL HBCU Intern: Howard University, Washington, DC 20059

**14. ABSTRACT**

This report is focused on a component of an effort intended to develop wireless sensor networks for real-time monitoring of airborne targets across a broad area. We have previously reported on two separate algorithms intended for identification of event occurrence. The goal of this report is to provide the detailed descriptions necessary for replication of the approaches. Here, we do not present new algorithms or data analysis. Instead, we present three different practical implementations of the data analysis algorithm originally described in the published articles and in the patent application. In addition, a discussion of the underlying equations and data structures is provided.

**15. SUBJECT TERMS**

environmental sensor, chemical sensor, colorimetric, algorithm, data analysis

| **16. SECURITY CLASSIFICATION OF:** | | | **17. LIMITATION OF ABSTRACT** | **18. NUMBER OF PAGES** | **19a. NAME OF RESPONSIBLE PERSON** |
|---|---|---|---|---|---|
| **a. REPORT** | **b. ABSTRACT** | **c. THIS PAGE** | | | Jeffrey S. Erickson |
| Unclassified Unlimited | Unclassified Unlimited | Unclassified Unlimited | Unclassified Unlimited | 29 | **19b. TELEPHONE NUMBER** *(include area code)* (202) 767-0364 |



**Standard Form 298 (Rev. 8-98)**
**Prescribed by ANSI Std. Z39.18**

This page intentionally left blank.

## CONTENTS

## FIGURES

# EXECUTIVE SUMMARY

In October 2012, the Center for Bio/Molecular Science and Engineering at the Naval Research Laboratory (NRL) began an effort intended to develop wireless sensor networks for real-time monitoring of airborne targets across a broad area. The goal was to apply the spectrophotometric characteristics of porphyrins and metalloporphyrins in a colorimetric array for detection and discrimination of changes in the chemical composition of environmental air samples. The effort encompasses hardware, software, and firmware development as well as development of algorithms for identification of event occurrence and discrimination of targets. We have previously reported on two separate algorithms intended for identification of event occurrence. The first of these algorithms was based on analysis of the standard deviation of reflectance data; the second utilized analysis of the slope in those data sets. While a short technical description and a flow chart were provided in the original publications, details on implementation were limited. The goal of this report is to provide the detailed descriptions necessary for replication of the approaches. Here, we do not present new algorithms or data analysis. Instead, we present three different practical implementations of the data analysis algorithm originally described in the published articles and in the patent application. In addition, a discussion of the underlying equations and data structures is provided.

This page intentionally left blank.

# PRACTICAL IMPLEMENTATION OF DETECTION ALGORITHM FOR REFLECTANCE-BASED, REAL-TIME CHEMICAL SENSING

## INTRODUCTION

We have previously reported two separate data analysis routines. [1, 2] The first algorithm was based on analysis of the standard deviation of reflectance data of paper supported porphyrin indicators; the second utilized analysis of the slope in these data sets. While a short technical description and a flow chart were provided for these routines, details on the implementation were limited. The goal of this report is to provide the detailed descriptions necessary for replication of those approaches. Here, we do not present new algorithms or data analysis. Instead, we present three different practical implementations of the data analysis algorithm originally described in the published articles and in the patent application. [3] In addition, a discussion of the underlying equations and data structures is provided.

The three implementations chosen for use in this discussion are C, Java, and Python. Each implementation brings unique features and challenges. C is a compiled language. It is highly architecture dependent and must be compiled for each platform with which it is used. It does, however, provide two advantages: (1) fast computing time and (2) the use of pointers giving direct access to platform memory. The memory access aspect makes C particularly attractive for embedded applications. C is a structured language, resulting in a single structured program. Because this provides an easily understood program, it is the first example presented in this document.

Java is an object-oriented language designed to produce bytecode that can be used with any system offering a Java Virtual Machine (JVM). It is designed for networked operation in a server/client environment, leading to a high degree of portability. Object oriented programming (OOP) results in encapsulated combinations of data and methods, increasing the complexity of the written code. Java is appropriate here because of the significant similarities between it and the C/C++ programming languages. While Java is useful for desktop and laptop computers, most mobile devices and some web browsers (i.e., Google Chrome) have moved away from Java as a universal platform. While the Android Software Development Kit is very similar to Java 7, the iPhone does not come with pre-installed Java.

The final of the three approaches is the Python implementation. Like Java, Python is an object oriented language; unlike Java, Python is a scripted language. Python scripts tend to be shorter and to run more slowly. Python is popular, however, among the robotics and “maker” communities. As an example, Python is used in the Robot Operating System middleware that has become popular among robotics enthusiasts. It also has advantages in ease of access to peripheral devices such as COM ports, USB, and Ethernet, increasing access to devices such as unmanned vehicles and sensor systems.

One or the other of these different implementations may be preferred depending on where the data analysis will be executed. If the data analysis is to be performed onboard the embedded sensor package, C may be appropriate. On an unmanned vehicle host, Python may be more appropriate. On a laptop or desktop computer, especially in a server/client configuration, Java may be the preferable language. As mentioned above, the examples provided here are intended to provide additional detail beyond that included in the papers and patent applications published to date. The intention is to provide the interested reader with

Manuscript approved October 3, 2018.

sufficient guidance to create code in the language appropriate to the particular platforms or devices to be used in a given application.

Here, all sample code was written for use on a laptop or desktop computer. The importance of this distinction comes in the format of the input data. Data is assumed to reside in a comma-separated value (CSV) text file (described more fully below). The headers in this file are well defined, and importantly, the experiment has been completed with all data available at the time of analysis. In other applications not considered here, data may be live-streamed by the sensor, leading to time dependent input with format dependent on the device firmware.

## BACKGROUND

The data analysis routines are based on (1) standard deviation and (2) slope. Based on initial prototypes and evaluations, the standard deviation based algorithm was developed. New prototypes and additional evaluations illustrated some of the shortfalls in this approach. As a result, a slope based approach was developed.

### Standard deviation based algorithm

Pseudocode for the standard deviation based algorithm are provided in Figure 1. Initial detection criteria are based on examination of the red, green, and blue color channels for each indicator; the white channel reported by the devices is ignored. At the current time point, the standard deviation for each color channel (RGB) of a single indicator is computed using the 12 most recent time points; these values are subsequently divided by the average intensity for the same 12 points. If this value for all three color channels is greater than 0.00015 or if any two of the values are greater than 0.015, the indicator is considered to have detected a potential event. The device (six indicators) is considered to have detected an event of interest when at least the specified minimum number (one, two, or three) of indicators report a detection event at the same time. After event detection, a time window of 30 min is activated. Any simultaneous indicator events that meet the specified minimum within this window are considered part of the initial event. The window length is extended by 30 min from the last event detected. Following 30 min without event detection, the time stamps for the first and last detected events are reported along with the identification of all indictors that met event detection requirements within the time period.

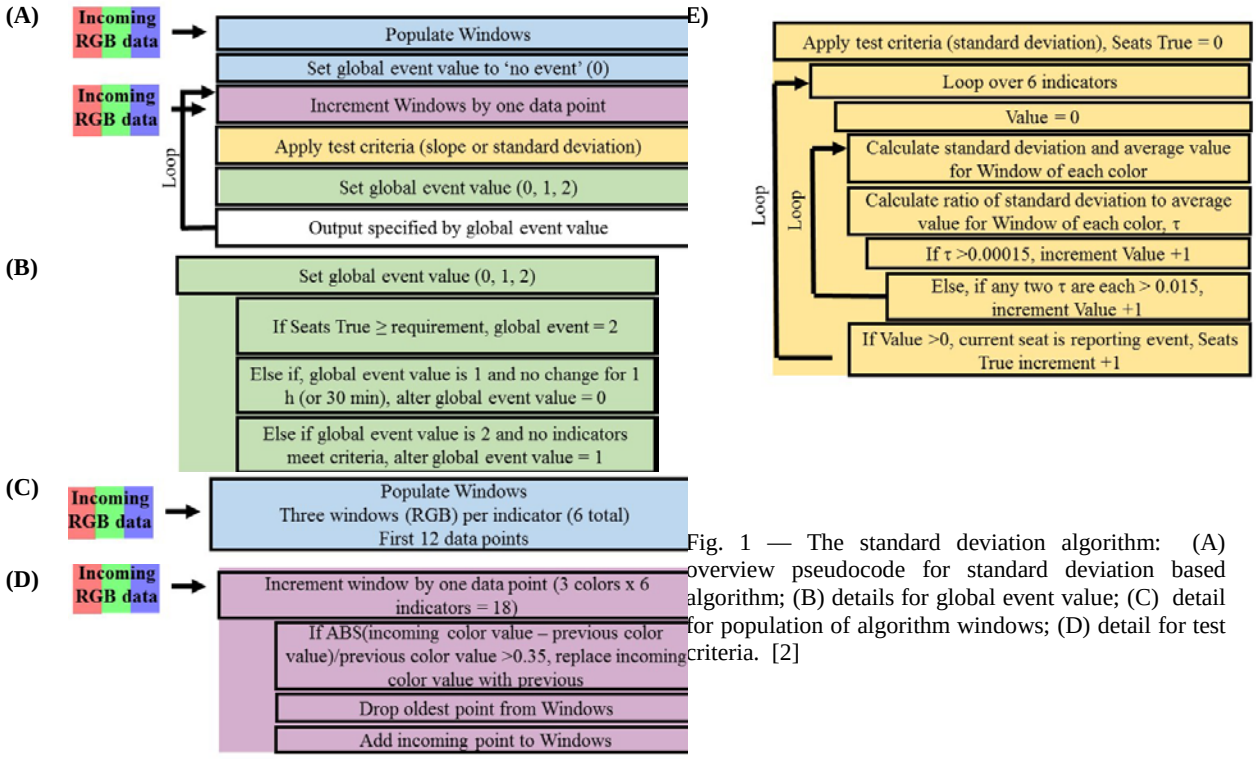


Fig. 1 — The standard deviation algorithm: (A) overview pseudocode for standard deviation based algorithm; (B) details for global event value; (C) detail for population of algorithm windows; (D) detail for test criteria. [2]

## Slope based algorithm

Pseudocode for the slope based algorithm is provided in Figure 2. Here, the threshold angle is fixed for each color value of each indicator based on the first 120 data points following initiation of the device. The initial intensity for each color is used in the following formula:

$$\emptyset = ae^{\frac{-RGB}{b}} + c \qquad \text{(Equation 1)}$$

where RGB is the initial intensity for a color channel. The parameters used depend on the minimum number of indicators or on the initial intensity values with a = 20 (for one indicator, 70 for more than one indicator), and b = 130 (for one indicator, 30 for more than one indicator), and c equal to the larger value of a user specified value (default 0.45°) or the angle of the dot product of the standard deviation calculated for the first 120 data points following initiation of the prototype. Linear regression formulas (least squares) are used to compute the slope and $R^2$ value for each color channel over two windows, current and baseline. The current window comprises the 20 (30 s sampling interval; 10 min) or 50 (5 s sampling interval; 4.2 min) most recent time points; the baseline is a window that comprises the next 120 most recent time points. The cosine of the angle between the slopes for the baseline and current windows is computed. If this value is less than the cosine of the threshold angle (above) and the $R^2$ value for the current window is greater than 0.67 (one indicator, 0.57 for more than one indicator), the color value is counted as 1; if the $R^2$ value was greater than 0.8, the color is counted as 2. Counts from all color channels for a given indicator must sum to greater than 1 for the indicator to be considered to have detected a change. High dose exposures can induce changes very rapidly; to address this situation, an additional test was added that uses only the 10 most recent data points (Snap Window in pseudocode). If the angle between the computed slope of this window and the reference region slope (Background Window in pseudocode) is greater than 12 degrees then the color contributes 1. After these tests have been applied to all indicators, cases in which the number of indicators

that have detected changes is greater than or equal to the minimum number specified are considered to have detected an event. Similarly to the standard deviation algorithm, an event window was used for determination of the end point. Here, that window is 60 min rather than 30 min. Any indicator events within this window extended the event window by 60 min until no events are detected. The time stamps of the first and last detected events are reported along with indicator identification for all indicators reporting within the window.

### Scope

The two analysis approaches are suitable for different types of data. As described in the previous publications, the standard deviation algorithm is highly effective in analysis of laboratory-based experiments conducted using petri dish exposures. [2, 3, 4] It was significantly less effective when applied to data analysis for outdoor experiments. [1, 2] For these data sets, the more complicated slope-based analysis showed significantly improved performance. In general, the code presented here is suitable for either type of analyses.

## ALGORITHM DETAILS

### Equations

There are four primary equations used in the algorithms described here:

1. The standard deviation formula used in standard deviation analysis only
2. The slope formula used in slope analysis only
3. The $R^2$ formula used in slope analysis only
4. The slope standard error formula used in slope analysis only

In order to minimize memory requirements for the calculations, formulas employ running sums. Each program also makes use of sliding windows, allowing a small number of arrays to hold all of the necessary raw data for each calculation. The standard deviation analysis uses a single sliding window with the 12 most recent data points. The slope analysis needs three sliding windows, a snap window with the 10 most recent data points, an active window with the 50 (or 20) most recent data points (for 5 s or 30 s sampling), and a background window with the 120 next most recent data points. It can also be implemented as a single sliding window of length 170 (or 140) data points with separate indices for the background, active, and snap portions of the data.

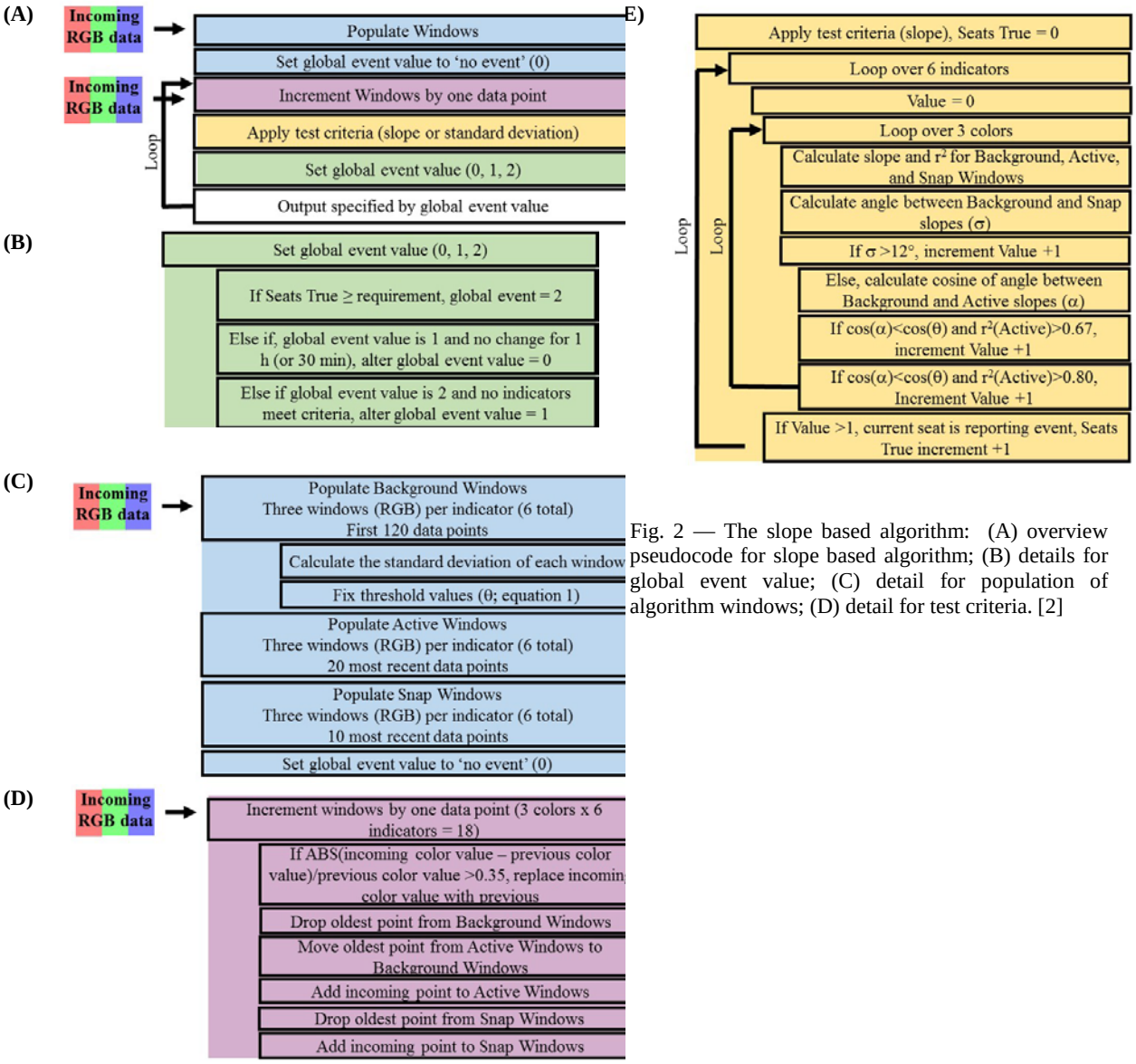


Fig. 2 — The slope based algorithm: (A) overview pseudocode for slope based algorithm; (B) details for global event value; (C) detail for population of algorithm windows; (D) detail for test criteria. [2]

Averages are calculated as:

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \qquad \text{(Equation 2)}$$

where $N$ is the number of data values and $\bar{x}$ is the average. The basis of all other calculations comes from the variance and covariance, defined here as[1]:

[1] Technically, these formulas are for the sample variance and covariance. To obtain the population variance and covariance, replace N-1 with N, as described at https://www.sciencebuddies.org/science-fair-projects/science-fair/variance-and-standard-deviation and http://www.r-tutor.com/elementary-statistics/numerical-measures/covariance.

$$VAR(x) = \sum_{i=1}^{N} \frac{x_i^2}{N-1} - \bar{x}^2 = \frac{1}{N-1}\left[\sum_{i=1}^{N} x_i^2 - \frac{\left(\sum_{i=1}^{N} x_i\right)^2}{N}\right] \quad \text{(Equation 3)}$$

$$COV(x,y) = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x}) * (y_i - \bar{y}) = \frac{1}{N-1}\left[\sum_{i=1}^{N} x_i\, y_i - \frac{\sum_{i=1}^{N} x_i * \sum_{i=1}^{N} y_i}{N}\right] \quad \text{(Equation 4)}$$

Then the standard deviation can be written as:

$$s(x) = \sqrt{VAR(x)} \quad \text{(Equation 5)}$$

The slope of a least-squares fit line through the data can be calculated as:

$$slope = \frac{COV(x,y)}{VAR(x)} \quad \text{(Equation 6)}$$

with an $R^2$ value of:

$$R^2 = \frac{COV(x,y) * COV(x,y)}{VAR(x) * VAR(y)} \quad \text{(Equation 7)}$$

Finally, the slope of the standard error[2] can be calculated with the formula:

$$Slope\ of\ SE = \sqrt{\frac{\frac{VAR(y)}{VAR(x)} - \frac{COV(x,y) * COV(x,y)}{VAR(x) * VAR(x)}}{N-2}} \quad \text{(Equation 8)}$$

Therefore, if running sums for *x*, $x^2$, y, and $y^2$ are stored, it is possible to quickly calculate variance and covariance from Equations (3) and (4). From these derived quantities, all remaining equations can be quickly evaluated. In theory, this should be sufficient to perform all of the calculations described in the patent application[3], where the *x*-values are time steps and the y-values are readings from the RGB sensors – there will be separate y-values for red, green, and blue. These RGB readings have no units, and are 16-bit unsigned integers with a value between 0 and 65,535. In some cases, however, the equations have been modified by pre-factors. The final set of equations used in the code multiplies (prefactor) by (equation):

$$s(y) = (no\ prefactor)\sqrt{VAR(y)} \quad \text{(Equation 9)}$$

$$slope = \left(\frac{1000}{y_{init}}\right)\frac{COV(x,y)}{VAR(x)} \quad \text{(Equation 10)}$$

$$R^2 = (no\ prefactor)\frac{COV(x,y) * COV(x,y)}{VAR(x) * VAR(y)} \quad \text{(Equation 11)}$$

$$Slope\ of\ SE = \left(\frac{1000}{y_{init}}\right)\sqrt{\frac{\frac{VAR(y)}{VAR(x)} - \frac{COV(x,y) * COV(x,y)}{VAR(x) * VAR(x)}}{N-2}} \quad \text{(Equation 12)}$$

[2] See, for example, http://seismo.berkeley.edu/~kirchner/eps_120/Toolkits/Toolkit_10.pdf, especially page 3, equation 17.

[3] The equations for variance and covariance presented above are mathematically correct. However, solving them in this form can introduce numerical instability. While we do not focus on this aspect of the calculations, further discussion can be found at https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance.

Here, the standard deviation calculations exclusively take place on *y*-values (intensity readings from the RGB sensor). In the *slope* and *slope of SE* prefactors, $y_{init}$ is the initial value (taken at $x = 0$) for the appropriate color channel. That is, for a slope calculation of the green voltage, use the initial green value. The standard deviation and $R^2$ calculations have no prefactors; this is explicitly shown in the equations above.

If the time steps are constant throughout the experiment, the variance in *x* does not change and needs to be calculated only once. However, the variance in *x* does depend on the size of the sliding window. Therefore, in the slope analysis separate values must be calculated for the background, active, and snap windows. In the example code provided, we calculate these quantities once, immediately after initialization is finished. The running sums for $x^2$ do not need to be stored after the variance in *x* has been calculated. The running sums for *x* are necessary, however, to calculate covariance.

In the slope analysis, it will be necessary to calculate the angle between various slopes. This is done through the use of the vector formula for the dot product:

$$\cos\theta = \frac{\vec{a}\cdot\vec{b}}{\|\vec{a}\|*\|\vec{b}\|} \qquad \text{(Equation 13)}$$

Technically, slope is a scalar quantity; however, since *y = slope * x*, vectors can be generated anytime by setting *x* = 1 and *y = slope*. An angle is computed using vectors (1, *slope a*) and (1, *slope b*). Then:

$$\|\vec{a}\| = \sqrt{1 + (slope\ a)^2} \qquad \text{(Equation 14)}$$

$$\|\vec{b}\| = \sqrt{1 + (slope\ b)^2} \qquad \text{(Equation 15)}$$

$$\vec{a}\cdot\vec{b} = 1 + (slope\ a) * (slope\ b) \qquad \text{(Equation 16)}$$

There is one special case that must be included. When calculating the parameter *c* (the slope threshold parameter, described in the next section), it is necessary to find the dot product of the slope standard error over the first 120 data points. One vector is for the average of the first 120 data points, which has no slope. The other is the slope of the standard error, given by Equation 12. The two vectors are then (1, 0) and (1, *slope of SE*). This error is multiplied by a factor of six; with the number used to obtain the parameter *c* is calculated as:

$$c(y) = larger\ of\left(\ 0.45°, \cos^{-1}\left[\frac{1}{\sqrt{1+(6*slope\ of\ SE*slope\ of\ SE)}}\right]\ \right) \qquad \text{(Equation 17)}$$

It is important to note that all calculations must take place in radians; values are presented to the user in terms of degrees here for clarity. To translate between the two:

$$quantity\ in\ degrees = \left(\frac{360°}{2\pi}\right) * quantity\ in\ radians \qquad \text{(Equation 18)}$$

$$\pi = \cos^{-1}(-1) \qquad \text{(Equation 19)}$$

**User-Specified Parameters**

The algorithm requires user supplied parameters for determination of an event. The user must specify the number of seats that must pass the event threshold value in order to indicate a positive. This can be any

value between 1 and the number of sensors in the device. In some advanced cases, the user may also wish to modify the standard deviation threshold values (default is threshold-*a* = 0.00015 and threshold-*b* = 100*threshold-*a* = 0.015), or the slope threshold parameter *c* (default is 0.45 degrees). Note that while the standard deviation threshold-*a* value can be tweaked, the threshold-b will usually remain fixed at 100 * threshold-*a*.

The programs provided in this report also give the user the ability to change the total number of sensors in the device (default is 6), and the number of text lines in the header file that the program reads from (default is 5, more on this point below).

**Initialization**

The initialization is also known as the cold start time for the sensor. Before the algorithm can begin to evaluate data for the presence of events, all sliding windows must be populated. There will be three sliding windows of each type per channel, one for each color. Initialization requires filling the first 12 data points for the standard deviation analysis, and the first 170 data points for the slope analysis (for 5 s sampling; 140 points for 30 s sampling). For the slope analysis, after the first 120 data points, the threshold angle must be determined. There will be three threshold angles per channel, one for each color. They are calculated using the equation:

$$\emptyset(y) = ae^{\frac{-RGB}{b}} + c(y) \qquad \text{(Equation 1)}$$

The parameters *a* = 20 [degrees] and *b* = 130 [no units] for a user-defined event threshold of one seat (70 and 30 for more than one seat). The parameters *c* [degrees] and *RGB* [no units] will have different values for each color and channel. To determine the parameter *c*, calculate the angle between the average of the first 120 data points and the standard error of the slope of these 120 data points, multiplied by a factor of 6, as described in the special case in the vector equation above. The parameter *c* will take the larger of this value and 0.45 degrees. The quantity *RGB* is the intensity of the first data point collected. As noted previously, although parameters are listed in degrees, all calculations must take place in radians.

**Data input tests**

In this implementation, data is read into the code by time step, ignoring data from the clear channel. For a six sensor device, this leaves 18 values that must be checked: red, green, and blue values for each of the six sensors. Each value is a 16 bit unsigned integer between 0 and 65,535. Before a new data point can be accepted, it must pass two tests. These tests are designed to eliminate missing data points and spikes. The first test is designed to eliminate zeros in the data set. If the value of the incoming data point is < 2, reject it and replace its value with that of the previous data point. If there is no previous data point, set the incoming data point to 2. The second test is designed to eliminate data spikes; it is only applied after the initialization phase is complete. Calculate the quantity:

$$\left|\frac{y_{incoming} - y_{previous}}{y_{previous}}\right| \qquad \text{(Equation 20)}$$

If this quantity is > 0.35, replace the incoming data value with the previous one.

**Conditions for a positive event**

The conditions for a positive event require a determination of whether each individual channel is positive or negative. These conditions are dependent on the type of analysis performed – standard deviation or

slope. Once all channels have been evaluated, it is necessary to sum the number of positive channels to see if the user-defined threshold has been reached. The event threshold and reporting rules are independent of analysis type.

Individual colors (RGB) → Channel → Event

To determine if an individual channel is positive by the standard deviation analysis, you will need to calculate the quantity:

$$\tau(y) = \frac{s(y)}{\bar{y}_{RED}} = \frac{s(y)}{\left(\frac{N}{\sum_{i=1}^{N} y_{RED}}\right)} \qquad \text{(Equation 21)}$$

The variable *y* can take values for each individual color, but, regardless of which of the three colors is being calculated, the standard deviation is always divided by the average intensity of the red channel over the sliding window. The three different $\tau$ values are compared to the standard deviation threshold values. If two of these $\tau$ values are greater than the threshold-*b* parameter (default is 0.015), the channel is considered positive. Alternatively, if all three $\tau$ values are greater than the threshold-*a* value (default is 0.00015), then the channel is considered positive.

In the slope analysis, positive indication is dependent on introduction of the parameter *Value*, initialized to zero. Angles between various quantities will need to be calculated using the vector formula in Equations 13-16. You will also need to know the slope threshold angle, $\phi$ (Equation 1), which is calculated after the first 120 data points are collected. The tests are as follows: for each color, first calculate the angle, $\sigma$, between the background and snap slopes. If $\sigma$ is greater than 12 degrees and $R^2$ for the snap window > 0.82, increment *Value* based on the intensity of the first data point (taken at x = 0) of that color in the experiment: increment by 2 if it was < 100; increment by 1 otherwise. Next, calculate the cosine of the angle $\alpha$ between the background and active slopes. If $\cos(\alpha) < \cos(\phi)$ and $R^2$ for the active window > 0.67 (0.57 for more than one indicator), increment *Value* by 1. If $\cos(\alpha) < \cos(\phi)$ and $R^2$ of the active window > 0.80, increment *Value* by an additional 1. After the entire set of tests has been completed for all three colors, if the parameter *Value* is greater than or equal to 2, the channel is considered positive.

Once all channels have been evaluated, sum the total number of positives. If the total number of positive channels is greater than or equal to the user-specified minimum number of required channels, a new event has started.

There are a set of basic rules that govern the classification and length of an event, as well as a buffer zone condition. The basic rules are these: (1) an event only begins when the total number of positive seats is greater than or equal to the user-set threshold value. (2) once the event has started, it continues until the total number of positive seats falls below the minimum threshold. (3) there must be a set period of time between distinct events of at least 30 minutes for the standard deviation analysis and 60 minutes for the slope analysis. If another event is registered within this window, it is not considered to be new. Instead, the original event is explicitly marked as extended. These basic rules are implemented with a parameter called the *Global Event Value* that specifies the status of the event and with a global event timer initialized to zero.

These basic rules are modified by a buffer zone condition. Recall that an event ends when the total number of positive sensors falls below the minimum threshold. In contrast, the buffer zone is a period of time that begins immediately after the total number of positives falls below the user-set threshold value to account for noise or bounce in the reported sensor positives. Every time the number of positive sensors is greater than or equal to the user defined threshold, the buffer timer is reset. Unlike the global timer, the

buffer zone is measured as a number of sensor readings (timesteps). The size of the buffer zone is set at 10 timesteps but can easily be changed in the code. The timer is stored as the variable *buffer_timer*, and the length of the buffer zone is stored in the variable *buffer_length*.

The classification of events depends on the status of both the global timer and the buffer zone timer. If both timers have expired, the event is classified as "new". If the buffer zone timer has expired but not the global timer, the event is classified as "extended". If neither timer has expired, crossing above the user-defined threshold is considered part of the original event and it receives no special marking. Because the buffer timer is significantly shorter than the global timer, there will be no cases where the global timer has expired but not the buffer zone timer.

There are six distinct conditions in the code to keep track of the flow of events. They are tracked by the parameters and timers described above.

0. A *Global Event Value* of 0 indicates that there is no event. (this is the default state)
1. If *Global Event Value* is 0 and the number of positive seats is greater than or equal to the user defined event detection threshold (1 or more), the *Global Event Value* is changed from 0 → 2. The event is classified as "new". Reset and start the buffer zone timer.
2. If *Global Event Value* is 2 and the number of positive seats is greater or equal to the user defined threshold value, reset and start the buffer zone timer.
3. If *Global Event Value* is 2 and the total number of positive sensors drops below the minimum threshold, reset and start the global event timer. The value of the global event timer is 30 min for standard deviation analysis and 60 min for slope analysis. Change the *Global Event Value* from 2 → 1 as soon as the buffer zone timer expires.
4. If the *Global Event Value* is 1, the global timer has not yet expired, and the number of positive seats becomes greater than the user defined event detection threshold, change *Global Event Value* from 1→2 and stop all timers. The event is considered to be extended.
5. If the *Global Event Value* is 1 and the number of positive seats is less than the user defined event detection threshold, decrement the global event timer by the time step.
6. Change *Global Event Value* from 1 → 0 if the global event timer expires.

Distinct events should be numbered sequentially. When reporting events to a file, use the following guidelines to determine which event any changes should be ascribed to:

1. If the *Global Event Value* was 0 and becomes 2, report the start of a new event, the timestamp at which it occurred, and the number/location of the initial positive seats.
2. If the *Global Event Value* is 2 and a new seat(s) becomes positive, add the number/location of the new positive seat(s) to the report and the timestamp at which it happened. This is part of the same event number.
3. If the number of positive seats drops below threshold, record the end time for the event. Report the end time as soon as the buffer zone timer expires.
4. If the *Global Event Value* is 1 and changes to 2 due to a sufficient number of new seats becoming positive, add the number/location of the new positive seats and the timestamp to the report. This is part of the same event number, but is reported as the event being extended.
5. When the *Global Event Value* changes from 1 to 0, report the stop time determined in step 3, as well as all seats involved in the original (and possibly extended portion) of the event.

Because of the reporting requirement in step 5, two separate seat counters are required. The sensor_on[] counter keeps track of the current condition of each seat: positive or negative. The status of each entry in this array can change at any time during an event. The cumulative_sensor_on[] counter is cumulative and only resets at the beginning of each new and distinct event.

**Code Details**

Three example codes that implement these analysis methods are included with this report. The original algorithm and the Python code were written by one contributor (APM). The C and Java codes were written by a different author (JSE) to give a second perspective on the problem. Each code uses a different programming style, with the idea that different readers will be more comfortable with different examples.

**C Program**

C is a procedural language. Resulting codes typically have a linear flow. There is a well-defined beginning and ending in main, with most of the implementation details left to function calls. Typically, professional programmers keep their functions in separate files. In order to emphasize the linear nature of the code, in this example both main and all functions are located in a single document. A flow chart is included to help the user understand the path the code takes through the functions. These are divided into five groups: a group of I/O functions, a group of data handling functions, a group of standard deviation analysis functions, a group of slope analysis functions, and a function to check for global (instrument) positives. These five groups are demarked by dashed boxes in Figure 3.

The example code is designed to be run from the command line, by typing

**> gcc Analysis.c**

to compile and create an executable file, and then

**> ./a.exe**

in order to run it. There are a number of flags that can be used to control the behavior of the code. The order of the flags is unimportant. These are:

-h prints a list of commands
-i to specify the input text datafile name (can be a prefix or include .txt)
-o to specify the output text datafile name (can be a prefix or include .txt)
-s to specify the global event threshold number of seats
-m to specify the analysis method: 0 = standard deviation , 1 = slope (default)
-a to specify the standard deviation threshold a value
-t to specify the slope threshold angle

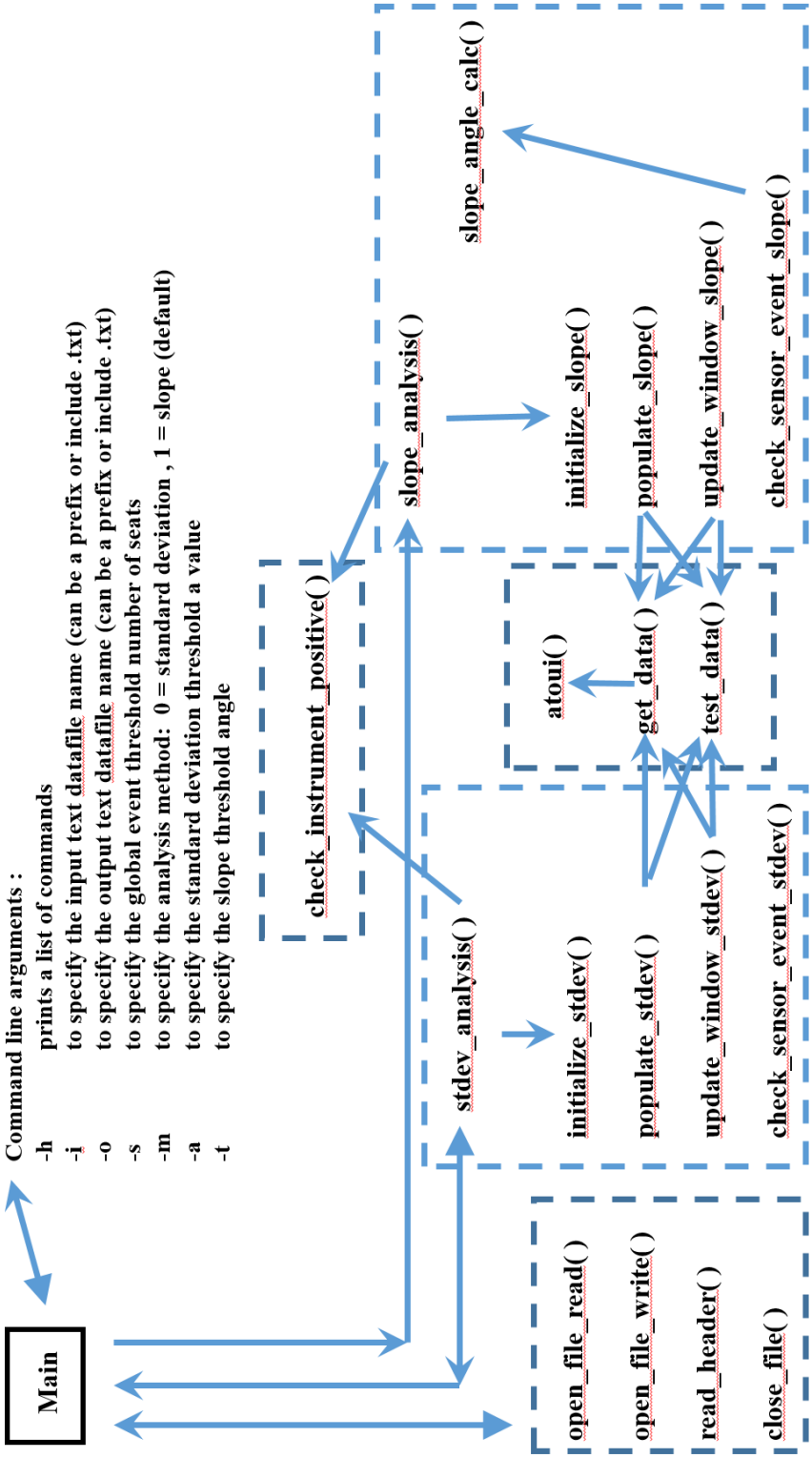


Fig. 3 — A simplified diagram of the various functions in the C code, showing how they are related to the main program and when they are called.

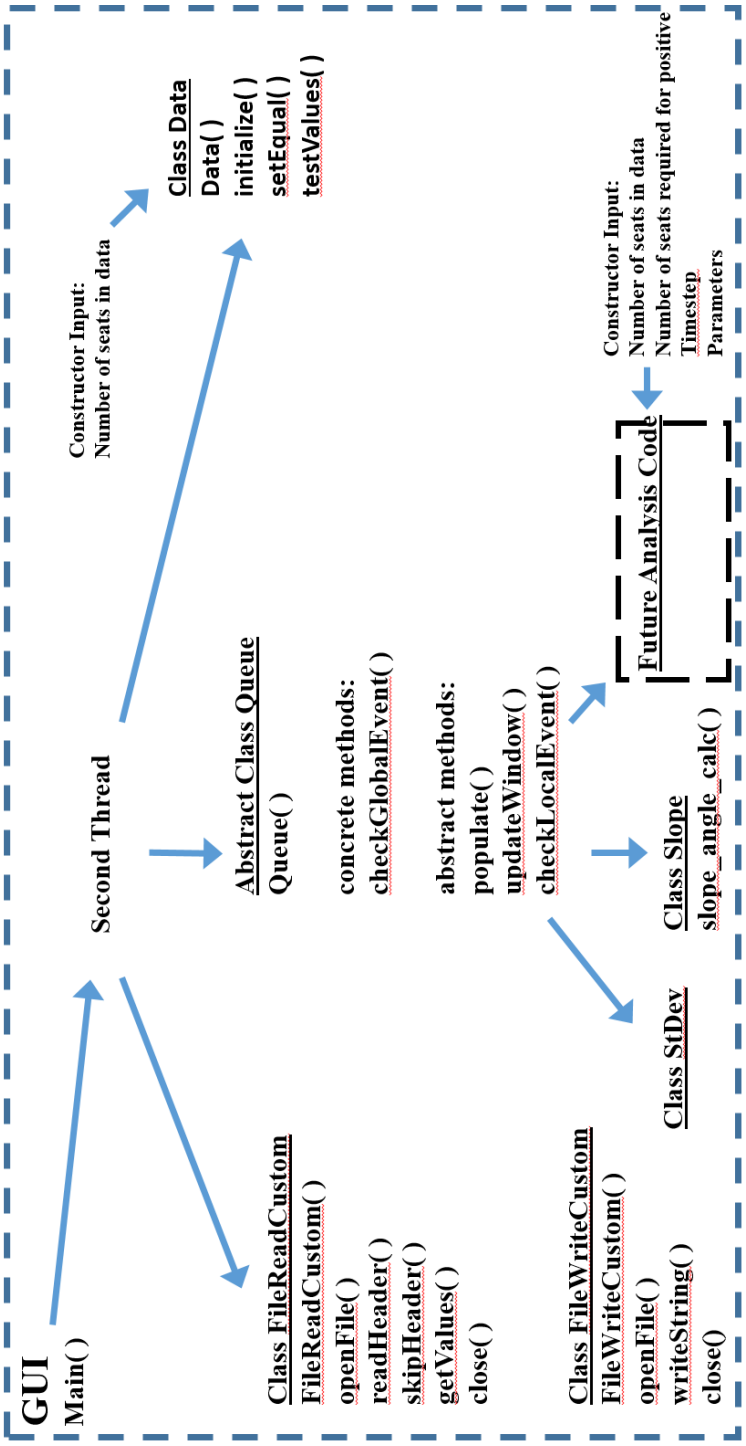


Fig. 4 — A diagram showing the different classes used in the Java code and their included methods.

Data files will be read from and generated in the folder location of the executable file. Error reporting in C is more difficult than other platforms; there are no built-in methods. Here, text messages are sent to the user interface. The program typically exits on error without completing any remaining steps in the code.

Most of the global data is stored in three different structures: a global_parameters structure, a set of nested structures for the Standard Deviation analysis, and a set of nested structures for the Slope analysis. These structures are heavily nested in order to make loop iterations compact.

**Java Code**

Java is an object oriented language. Unlike the C program, it does not have a well-defined flow. Instead, the program is based around the development of classes that hold data and methods. Figure 4 details the classes.

The first classes were developed for file input and output. Each class is instantiated once for each file to be handled. The FileReadCustom class uses the java scanner to collect input. It has methods to open files, read headers, skip headers, and get a line of values from the text document, given a data structure to pass them into. The FileWriteCustom class uses a buffered file writer.

The FileReadCustom class requires a data structure to hold lines of data while they are read. This is a new class, Data, with methods to initialize the structure, test values according to the acceptance testing in the previous functions, and to set two data structures equal to each other. The constructor requires the number of color sensors (seats) as an input.

The analysis methods themselves take advantage of polymorphism in order to make the implementation of new algorithms easier. An abstract superclass Queue was developed. It has one concrete method to check for global events on the instrument. It has abstract methods to populate sliding windows, update these windows with new data, and check local events. Queue is abstract, so it cannot be instantiated. However, it does have a constructor which is invoked when subclasses are instantiated. The constructor requires information about the total number of seats in the instrument, the user-defined global threshold value, the size of the timestep, the starting value of the global event timer, and the starting value of the buffer zone timer.

Two classes were written to implement the data analysis, StDev and Slope. Both of these methods extend Queue. They provide concrete implementations of the functions described above, which tells the program which seats are positive. In addition to data required for the constructors for their superclass Queue, they also require information about their individual threshold parameters. Various methods in both the super- and sub-classes use objects from Data, FileReadCustom, and FileWriteCustom.

The entire code is tied together with the Analysis class. It implements a GUI using JavaFX. The GUI collects information about the input and output file names and desired analysis type, as well as a number of optional parameters. When the user presses Start, rather than running the code directly, the Analysis class creates a new thread by instantiating the class SecondThread. This prevents the GUI from being slowed down while the analysis runs in the background. Error reporting takes place in the GUI itself.

As mentioned previously, the Java code was written to simplify the addition of new algorithms. In order to do this, simply add a new class named after the algorithm. This new class must extend Queue. Three methods will need to be defined: populate( ), updateWindow( ), and checkLocalEvent( ). Other methods can be added as desired. To use the new algorithm, simply take advantage of the fact that a

subclass object can be assigned to a superclass reference. In the SecondThread class, a superclass reference of type Queue is created. Subclass objects of the proper algorithm type are instantiated based on the user's GUI choices, and that particular object is assigned to the superclass reference. No other code needs to be changed. The new algorithm is forced to define concrete implementations of its abstract methods, and these are what the superclass reference calls in the SecondThread class.

Note that populate( ), updateWindow( ), and checkLocalEvent( ) require specific input types and have return values. They were left out for clarity. To be specific, they are:

int populate(Data ob1, Data ob2, FileReadCustom ob3, FileWriteCustom ob4, FileWriteCustom ob5)
int updateWindow(Data ob1, Data ob2, FileReadCustom ob3)
void checkLocalEvent(void)

In its current state, the Java code is designed to be started from the command line. In theory, it is possible to use javapackager or an IDE like NetBeans to create an executable JAR file that can be run from the desktop. However, to do this properly it is necessary to have a code-signing certificate from a Trusted Certificate Authority (ex: Verisign) to attach to it. This process is described in more detail in Appendix C.

**Python Code**

Python code is also an object oriented language. It is based on the use of objects that contain data and the methods used to act on it. Rather than what was used for the development of the Java language where data input and output generates the flow of objects, the Python code was built upon consideration of an individual sensor unit with classes below and above built to extend the usability of the code. The class pcolorsensor contains three instances of the slwindow class to track data for each window of data used for calculations, the status of the sensor, and the test criteria used to determine that status. The class has methods to initiate an instance of the class, creating the windows and passing in the test criteria, add data to the window classes, and any methods that test the current status of the sensor instance. Currently, a standard deviation test method and slope test method are defined, but the applicability of the system can be extended for any new status check algorithm by adding a new method to the pcolorsensor class. The slwindow class holds the data of a particular window including the actual intensities and times as well as sums of current values and squares of current values. The class also has methods to initiate default values, add, subtract, increment (add and subtract), and compute standard deviation and slope of the window.

Classes above pcolorsensor can be developed for specific applications. In the current instance, an object anomalytest was created that will create a specified number of pcolorsensor instances. The anomalytest class, besides holding instances of individual sensors, holds the overall state of the sensor suite. The method has an initiation method to pass in parameters and then an increment method to add data to the sensors as well as a test method to run either of the test methods for each individual sensor and execute the overall status classification routine described above. The final class above anomalytest is the main Python routine which loops through a call of the increment method of anomalytest followed by a call of the test method, if sufficient data has been generated after the data is read in. After the test method is run, the main routine handles output of results. Methods were used to fetch data so that different implementation scenarios could be easily handled. A method to read a metadata file that contains information about when and what was exposed to the sensor was created as well as the file name of the raw data. Another method was created to read this data file. The current implementation of the Python code allows for more detailed comparison than those available with the other routines. After an event is detected by anomalytest, comparisons are made to a list of exposures to verify whether a spiked exposure event was detected.

The Python code is called from the command line and uses the same flags as listed for the C code with one change in meaning. The -i flag is not the data file itself but the meta data input file which holds a list of events and the name of the data file.

## INPUT FILE FORMAT

Here, we use data files that were collected from the ABEAM v2.08, an NRL custom built reflectance sensor. The example codes provided in this report can easily be configured to use data from other platforms. The ABEAM v2.08 instrument is controlled through a GUI that collects raw output from the instrument, formats and plots it, and outputs a processed text file. This text file will be the input for the analysis code. In general, the input text file will contain a descriptive header, followed by data. GUI software version 2.0 produces a text file with header five lines long. An example is below:

```
ABEAM Data
Multiplex hardware version 2.08, Software version 2.0
Autonomous run started on 10/10/2017 at 13:43
Data in absolute counts, 100 ms integration time, Order is: red, green, blue, clear
Time (s),  ,Sensor 1,  ,  ,  ,  ,Sensor 2,  ,  ,  ,  ,Sensor 3,  ,  ,  ,  ,Sensor 4,  ,  ,  ,  ,Sensor 5,  ,  ,  ,  ,Sensor6
```

The next section of the file contains data. Data for each timestamp is on a new line. The timestamp of the first line is always 0. All data is stored as a list of comma separated values. The first value presented is the timestep, as a long unsigned integer. The sensor data is presented next, as unsigned 16-bit integers. The order of the presented data is clear, red, green, blue. Each color will have a value between 0 and 65535. Timestamp data is separated from sensor data with a series of 3 spaces and a comma. Individual sensors are also separated from each other with a series of 3 spaces and a comma. An example is below.

```
0,  ,199,27,77,89,  ,2,0,1,0,  ,5,1,2,1,  ,37,5,14,17,  ,49,6,17,20,  ,38,5,15,18
5,  ,32,4,12,14,  ,2,0,1,0,  ,2,0,1,0,  ,45,6,17,21,  ,60,7,21,25,  ,47,6,19,22
10,  ,39,5,15,17,  ,2,0,1,0,  ,2,0,1,0,  ,45,6,17,22,  ,61,7,21,25,  ,48,6,19,22
15,  ,40,5,15,18,  ,2,0,1,0,  ,2,0,1,0,  ,45,6,17,22,  ,61,7,21,25,  ,48,6,19,22
20,  ,40,5,15,18,  ,2,0,1,0,  ,2,0,1,0,  ,45,6,17,22,  ,61,7,21,25,  ,48,6,19,22
```

The header does not contain information on the sampling interval. There are two possible methods to obtain this information. In the C and Java codes presented here, the sampling interval is obtained directly from the second line of data in the file. In the Python code, the user supplies the experiment start and end times. Then, based on the total number of data points in the file, the code calculates the sampling interval. This approach is especially useful when the hardware clock on the instrument is not sufficiently accurate, although it does require the technician to manually record the additional data.

In some cases, the header files may not be five lines long[4]. In all of the codes presented here, the user has the option to input the number of text lines in their header file. Default value is 5.

## ACKNOWLEDGEMENTS

E. Tankard was at NRL during the summer of 2018 through the US Office of Naval Research (ONR) sponsored NRL HBCU internship program.

## REFERENCES

1. B.J. Johnson; A.P. Malanoski; J.S. Erickson; R. Liu; A.R. Remenapp; D.A. Stenger; M.H. Moore, "Reflectance-based detection for long term environmental monitoring," *Heliyon* **3**, e00312 (2017).

[4] For example, a much older version of the ABEAM used a two line header file. Future ABEAM GUI versions may include a sixth line in the header with the sample time information.

2. A.P. Malanoski; B.J. Johnson; J.S. Erickson; D.A. Stenger, "Development of a Detection Algorithm for Use with Reflectance-Based, Real-Time Chemical Sensing," *Sensors* **16**, (2016).
3. B.J. Johnson; J.S. Erickson; J. Kim; A.P. Malanoski; I.A. Leska; S.M. Monk; D.J. Edwards; T.N. Young; J. Verbarg; C. Bovais; R.D. Russell; D.A. Stenger, "Miniaturized reflectance devices for chemical sensing," *Meas. Sci. Technol.* **25**, (2014).
4. B.J. Johnson; R. Liu; R.C. Neblett; A.P. Malanoski; M. Xu; J.S. Erickson; L. Zang; D.A. Stenger; M.H. Moore, "Reflectance-based detection of oxidizers in ambient air," *Sens. Actuator B-Chem.* **227**, 399-402 (2016).

## Appendices

The goal of the appendices is to quickly bring a user up to speed using free coding tools available on the internet. Although free IDEs are available for all three programming languages used here (for example, Dev-C++, NetBeans for Java, and Python-xy), it takes some time to learn their use. For simplicity, in this report we use command line compilers. In order to write and execute a program with a command line compiler, a text editor is required. It is necessary to stay away from word processing programs like Microsoft Word, because, by default, they save documents in proprietary formats that can't be read by the compiler. They also add formatting that is incompatible with most compilers. We suggest using Notepad++, which is available for free at https://notepad-plus-plus.org/. At the time of writing, the most current version is v7.5.6.

You will also need access to a command prompt. We suggest using PowerShell. This is easily found using the Windows search bar tool.

Once in PowerShell, you will need to move to the directory that contains your program. You will also need to make sure that the compiler in question has access to the folder with your programs. This can be done by either navigating the command prompt to the folder containing your programs, or by specifying the path. Note that if a program you are testing locks up in PowerShell, you can break out by typing CTRL-C. Administrator access is not required to run any of these programs, but you will need it in order to install them.

## APPENDIX A: Installation and Use of C Complier in a Windows 10 Environment

There are a large number of compilers and IDEs available for the C language. A partial list of popular versions is here: https://en.wikibooks.org/wiki/C_Programming/What_you_need_before_you_can_learn. In this work, we use the GNU C Compiler, which is free and included with virtually all Linux distributions. Windows versions are also available. To check for a version of gcc on your computer, open a command prompt and type the bold text below:

**>gcc --version**
gcc (GCC) 4.4.0
Copyright (C) 2009 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
Warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

(Note: there are two dashes in front of version). If GCC is installed on your system, you should see an output similar to the response above. However, we still recommend upgrading to the latest release. At the time of writing, GCC version 7.3.0 is available.

In this Report, we use the minGW-W64 version of the GNU C compiler, available for free download here: https://mingw-w64.org/. Under "Pre-built toolchains and packages" on the downloads page, choose the "MinGW-W64 builds". You will also need a text editor; we recommend using Notepad++. Finally, you'll need to update the PATH environment variable, if it is not already set. This is necessary so that when you type gcc at the command prompt, your terminal will know where to find the compiler. In Windows 10, you can do this as follows:

1. In Search, search for and then select: System (Control Panel).
2. Click the **Advanced system setting** link. You may need to log in as an administrator to do this.
3. Click **Environmental Variables**. In the section System Variables, find the PATH environment variable and select it. Click **Edit**. If the PATH environment variable does not exist, click New.
4. In the **Edit System Variable** (or **New System Variable**) window, specify the value of the PATH environment variable. This should point to the \bin directory inside the main MinGW directory. (Example: C:\Program Files (x86)\mingw-w64\i686-7.3.0-posix-dwarf-rt_v5-rev0\mingw32\bin)
5. Click **OK**. Close all remaining windows by clicking **OK**.

On Windows machines, the C compiler can be run from PowerShell. A basic description is here: http://www.mingw.org/wiki/mingw_for_first_time_users_howto. You will need to navigate to the folder where your C programs are stored. The gcc command is used to compile. Adding a -o switch allows you to rename the output file from the default a.exe. For example, type gcc xxx.c -o yyy.exe, where xxx is the name of the program and yyy is the desired name for the executable file.

The executable file can be run directly from a command prompt window. Running executables in PowerShell is less straightforward[5]. PowerShell will automatically run executable files that exist inside search path directories. For directories that are not located inside of the PATH, if you have navigated to the directory in which your executable file resides, you can run it by preceding the command with a dot-slash (./). That is, to run foo.exe from the directory C:\Users\jerickson\programs, type

**PS C:\Users\jerickson\programs> ./foo.exe**

Otherwise, you can use the call operator (&) to notify PowerShell that the target is in fact an executable file. For example, to run the example above from the C: directory, type

**PS C:> & C:\Users\jerickson\programs\foo.exe**

[5] A more complete description of this topic can be found at https://4sysops.com/archives/use-powershell-to-execute-an-exe/

## APPENDIX B: Installation and Use of Java Complier in a Windows 10 Environment

An excellent resource for Java beginners is the JavaRanch website https://javaranch.com/. Installing and running Java on your machine is well described under the "Create your first Java program" link. Many computers come with Java pre-installed, although you will need to make sure that it is the Java Standard Edition Development Kit (JDK) and not just the Java Runtime Environment (JRE). To check for a JDK installation, open a command prompt and type the bold text below:

> **> javac –version**
> javac 1.8.0_151

(Note: there is one dash in front of version). If JDK is installed on your system, you should see an output similar to the "javac 1.8.0_151" response above. However, we still recommend upgrading to the latest release, if a newer version is available.

The most current version of JDK can be downloaded at Oracle's website, http://www.oracle.com/technetwork/java/javase/downloads/index.html. At the time of writing, the latest version available is 10.01. You will also need a text editor. As described at the beginning of the appendicies, we recommend using Notepad++. Finally, you'll need to update the PATH environment variable, if it is not already set. This is necessary so that when you type javac at the command prompt, your terminal will know where to find the Java compiler. In Windows 10, you can do this as follows: (https://www.java.com/en/download/help/path.xml)

6. In Search, search for and then select: System (Control Panel). You may need to search on "Control Panel System" to find it.
7. Click the **Advanced system setting** link. You may need to log in as an administrator to do this.
8. Click **Environmental Variables**. In the section System Variables, find the PATH environment variable and select it. Click **Edit**. If the PATH environment variable does not exist, click New.
9. In the **Edit System Variable** (or **New System Variable**) window, specify the value of the PATH environment variable. This should point to the \bin directory inside the main Java directory. (Example: C:\Program Files (x86)\Java\jdk1.8.0_151\bin)
10. Click **OK**. Close all remaining windows by clicking **OK**.

On Windows machines, Java can be run from PowerShell, as described at the beginning of the appendicies. You will need to navigate to the folder where your Java programs are stored. To compile, type javac xxx.java. To run, type java xxx. Here, "xxx" is the program name.

## APPENDIX C: Generation of Executable JAR Files

In order to generate executable JAR files, it is necessary to obtain a code signing certificate. Ideally, this would be obtained from a Trusted Certificate Authority like DigiCert or VeriSign. However, there are other options. On a managed local network, it is possible for a system administrator to generate a code-signing certificate that is valid on that particular network. It is also possible to self-sign your code. Note that self-signing is not recommended for executable files that will be distributed, although any certificate you generate can be added to the trust list of other computers.

The java command keytool is used to generate public and private keys, requests, and to join them together. Documentation for the keytool can be found here:
https://docs.oracle.com/javase/1.5.0/docs/tooldocs/solaris/keytool.html

To obtain a code signing certificate, you will need to run these commands below in the directory in which you store your java code. We have chosen the alias "server". You can change this to anything you like, as long as you are consistent throughout the list of commands.

1. First, generate a keystore file. The command to do this is:
   keytool -genkey -alias server -keyalg RSA -keysize 2048 -keystore keystore.jks
2. Next, generate a certification request.
   keytool -certreq -alias server -keystore keystore.jks -file csr.csr

This process will create two files in the directory in which you ran the commands. They are csr.csr, and keystore.jks. The keystore is the public/private encryption file, which is the basis for everythng. The csr file is the request for a certificate that you will send to the trusted certificate authority.

To get a certificate from a trusted certificate authority, you will need to send them the csr.csr file and pay the fee. Note that you will need a .p7b file, not a .crt (or .cer) file. To get a certificate on a managed local network, send the csr.csr file to your systems administrator and ask for a code-signing certificate. If the certificate is not a .p7b file, you can easily convert it as shown here:
https://knowledge.digicert.com/solution/SO7948.html. Use "method 1: Using windows".

3. Download the requested certificate. Use Base-64 encoding. Rename it as <your name>.p7b and place it in the same directory as the .csr and .jks files. Install the certificate with this command:
   keytool -import -trustcacerts -alias server -keystore keystore.jks -file <your name>.p7b

Alternatively, to generate a self-signed certificate, first generate the keystore file and certificate request as mentioned above:

1. Generate a keystore file with this command:
   keytool -genkey -alias server -keyalg RSA -keysize 2048 -keystore keystore.jks
2. Create the self-certification. Note that –validity specifies the number of days. Suggest 360 days (1 year).
   keytool -selfcert -validity 360 -alias server -keystore keystore.jks
3. Optional: if you wish to export a certificate to add to the trust list of other computers, do this:
   keytool -export -alias server -keystore keystore.jks -file <your name>.crt

Finally, to create and sign the executable JAR file, do the following.

1. Go to the directory of the program you'd like to make into a JAR.
2. Make sure the program is compiled. You need class files. Not java files.

3. In order for the JAR to be executable, you'll need a manifest file. This is simply a text file with three lines, saved with the extension .mf. The second line must list the class that contains main(). Note that the third line must be blank. Example:

   Manifest-Version: 1.0
   Main-Class: Analysis
   (leave this line blank)

4. The file is typically built with a command like this:

   jar cvfm Analysis.jar Analysis.mf *.class.

   The c option creates a new jar file. The v option specifies verbose. The f option specifies that the output goes into a file specified in jarfile (instead of standard output). The m option specifies the inclusion of a manifest.
   More information is given here:
   https://www.ntu.edu.sg/home/ehchua/programming/java/J9d_Jar.html
5. Move the jar file to the directory with your key file.
6. Run the following command, where <filename> is the name of the .jar file you are trying to sign:

   jarsigner -tsa http://timestamp.digicert.com -keystore keystore.jks <filename> server
7. You can check what certificates you have attached to your keystore as follows:

   keytool -list -v -keystore keystore.jks

## APPENDIX D: Installation and Use of Python Complier in a Windows 10 environment

There are currently two different release versions of Python: these are Python 2 and Python 3. Python 2 is mostly used for legacy work, since the last major release (version 2.7) was in 2010. Python 3 is the new, actively supported version. However, a number of currently existing computer programs will not work under Python 3, and some new machines still come with Python 2 installed as default. More about this topic can be found here: https://wiki.python.org/moin/Python2orPython3. In general, most Windows machines do not come with Python pre-installed. To check for a Python installation, open a command prompt and type the bold text below:

> **> python --version**
> Python 3.5.0

(Note: there are two dashes in front of version). If Python is installed on your system, you should see an output similar to the "Python 3.5.0" response above. However, we still recommend upgrading to the latest release, if a newer version is available.

In this work, we use Python 3. The most current release at the time of writing is Python 3.6.5, and it can be downloaded here: https://www.python.org/downloads/. When you run the installer, make sure you click the "Add Python 3.6 to PATH" option. You will also need a text editor. As described at the beginning of the appendicies, we recommend using Notepad++.

On Windows machines, Python can be run from PowerShell, as described at the beginning of the appendicies. You will need to navigate to the folder where your Python programs are stored. To run a python script, simply type python xxx.py from the command prompt, where xxx.py is the name of the script.

## APPENDIX E: Included electronic files

These additional electronic files are included with this report:

1. C Code –source code for the algorithms in C
2. Java Code – compiled and source code for the algorithm in Java
3. Python Code –source code for the algorithm in Python