Final Report: Carbon Dioxide Secretion from Carbonated Beverages

Section: 101, Group: 3

Matthew Hanley: 702-362
Erik Pronk: 325-572
Gabriel McGann: 058-444

MCEN 3047: Data Analysis and Experimental Methods
Date: December 18, 2017

Abstract

The purpose of this experiment was to use a Sparkfun CCS811 air quality sensor to measure the relative CO_2 levels secreted by various carbonated beverages. Placed in a sealed enclosure, the sensor was linked to a data acquisition device (DAQ), in this case an Arduino. The data was then imported into MATLAB, where it was analyzed.

The main purpose or research question considered during the experiment was whether container type (glass bottle vs. aluminum can) or beverage temperature (room temperature vs. refrigerated) would have a significant effect on the rate at which carbon dioxide is released from the beverage after it has been opened. It was also to be determined whether the theoretical model for CO_2 secretion matches with the data obtained experimentally.

After running tests and collecting and interpreting data, it was found that the experimental data did seem to fit the model fairly well, except for certain anomalies. Also, it was found that no clear conclusion can be drawn about the effect of container type or temperature on the rate of carbon dioxide secretion. More details on the experimental methods, analysis, and conclusions drawn from the experiment are found below.

I. Background

A. Carbon Dioxide Sensor (SparkFun CCS811)

The sensor used in this expirement is the SparkFun Air Quality Breakout (CCS811). This sensor senses many different Total Volatile Organic Compounds (TVOCs) including equivalent carbon dioxide (eCO_2). It has an eCO_2 sensing range of 400 to 8,192 ppm. eCO_2 is not equivalent to CO_2 as eCO_2 is only concerned with changes in CO_2 and not the actual levels. In this experiment, the actual levels of CO_2 are not important as the rate of change is what is concerning.

This sensor is easily integrated with an Arduino. Fig. 1 shows a typical hookup schematic for the sensor and Arduino.

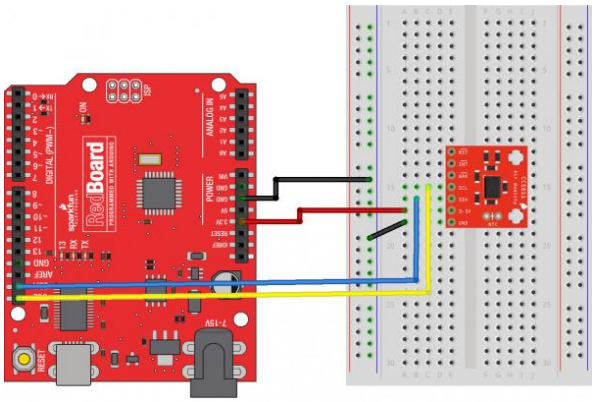

***Figure 1** – Wiring diagram showing connection to Arduino.*

In the diagram in Fig. 1, GND, 3.3V, SDA, and SCL pins on the CCS811 are respectively connected to GND, 3.3V, A4, and A5 pins on the Arduino. Code for the Arduino can be found in Appendix I. This code writes eCO_2 data to the serial stream of the Arduino.

It should also be noted that SparkFun recommends a 48-hour break-in time on the CCS811 sensor before getting reliable results. SparkFun also suggests a 20-minute warm up time when restarting the sensor.

B. Bubbles and Carbonated Beverages

Once opened, a carbonated beverage secretes carbon dioxide in via two different methods. The first is diffusion from the surface of the beverage into the surrounding air. The second is the release of carbon dioxide from bubbles.

Bubbles are formed in a carbonated beverage at what are called nucleation points. A nucleation point is simply a defect in the vessel that holds the beverage. This defect may be a manufacturing defect or simply due to the roughness of the container. As the roughness of the container increases, the number of nucleation sites increases, and

therefore the number of bubbles released increases.

A bubble prefers to form at a nucleation point because the surface tension in the liquid is decreased and thus less energy is required for the bubble to form at this site. Nucleation points are easily seen in glass vessels as a point where a stream of bubbles appears to be originating.

C. Theoretical Model

According to Sebastien Uzel in his paper "Modeling the Cycles of Growth and Detachment of Bubbles in Carbonated Beverages," the model of the number of bubbles secreted in a beverage versus time can be defined as:

$$n(t) = N\left(1 - e^{-\left(\frac{t}{\tau_n}\right)}\right)$$

Where $n(t)$ is the number of bubbles secreted at a given time, t. N is representative of the total number of bubbles secreted by the beverage, and τ_n is a time constant that defines how quickly bubbles are secreted. Using this model, it is possible to relate how quickly bubbles are secreted from beverages under different conditions. An example of this logarithmic-growth trend can be found in Fig. 2 below.

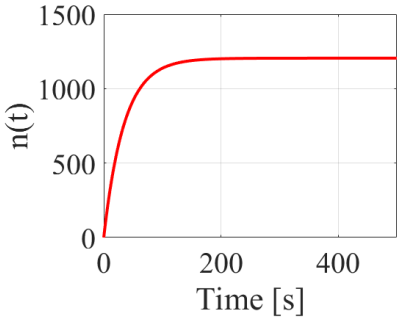

Figure 2 *– Theoretical model of number of bubbles (n), vs. time (t)*

II. Procedure

A. Break-in the sensor

If the sensor has not yet been used before, then it needs to be run for 48 hours before reliable data can be taken. Hook up the sensor according to Fig. 1 and run the Arduino code provided in Appendix I.

B. Create the test chamber

The testing apparatus needs to be made prior to the first experiment. On all other experiments, this step can instead just be a check to verify the seals are still intact.

The first step in making the test chamber is to cut holes in the side of the bucket about the size of the yellow rubber gloves. Place the yellow rubber gloves through the holes and seal them with duct tape. Cut another hole near the bottom of the bucket so that the sensor can be put into the bucket. Seal with more duct tape. Now cut a hole in the top of the cap of the bucket and tape the other bucket on top of the cap. Cut a hole in the side of the upper bucket and tape plastic wrap over is so that the beverage can be seen inside. The final product should look like the image in Fig. 3.

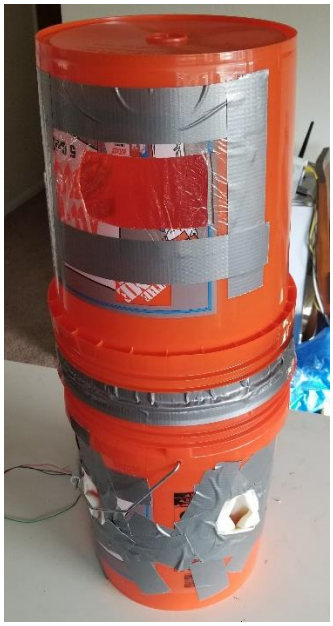

Figure 3: *The final bucket setup. The holes at the bottom are where the gloves are and the top hole is a window so that the inside can be*

seen. On the backside is where the wires for the sensor are put in.

C. Prepare beverage in test chamber

With the top bucket off of the test setup, place an unopened beverage into the bottom of the first bucket. Place the top bucket with the cap on the bottom onto the lower bucket. Apply firm pressure on the top bucket so that the cap seals.

D. Run MATLAB script

Run the MATLAB script that plots the data real time. The plot should show data being collected from the sensor at a y-value of about 400 to 500. The MATLAB code can be found in Appendix II.

E. Open beverage

Once the sensor has been confirmed to be collecting data, open the beverage in the sealed container using the yellow rubber gloves and a bottle opener if needed. Carefully remove your hands from the yellow gloves without exciting the air inside the chamber too much.

F. Data collection

Watch for the first 3-5 minutes of data collection to ensure that the sensor is working as expected. Now, let the experiment go for about half an hour.

G. Post experiment

The MATLAB script can be stopped after the 30 minutes of data collection have transpired and then the data saved. Remove the top cap from the lower bucket taking care to not spill the beverage inside of the bucket. Remove the bucket and repeat the process for however many beverages you would like to test.

III. Results and analysis

A. Data

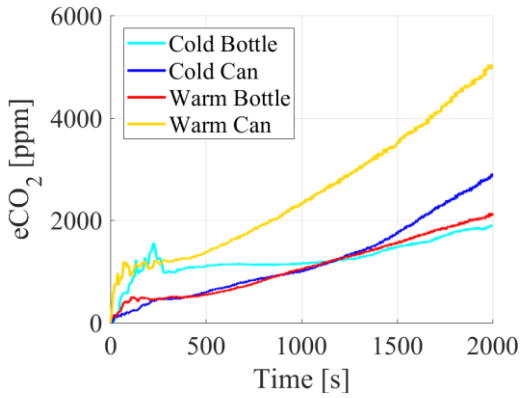

Figure 4 *– Data from all experiments overplotted for a duration of 2000 seconds.*

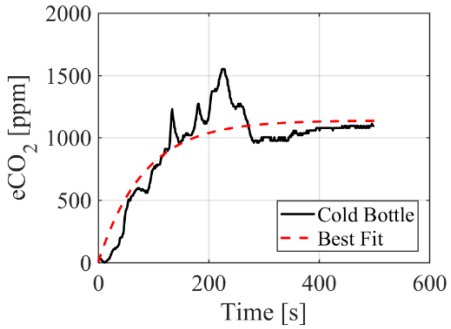

Figure 5 *– First 500 seconds of data for "Cold Bottle" experiment with best fit line from the theoretical model.*

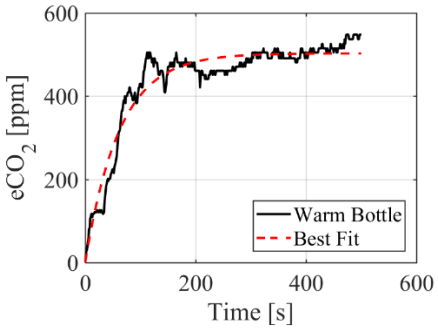

Figure 6 *– First 500 seconds of data for "Warm Bottle" experiment with best fit line from the theoretical model.*

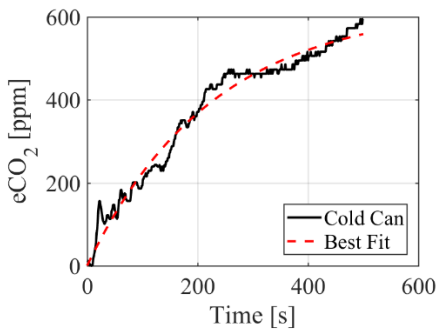

Figure 7 *– First 500 seconds of data for "Cold Can" experiment with best fit line from the theoretical model.*

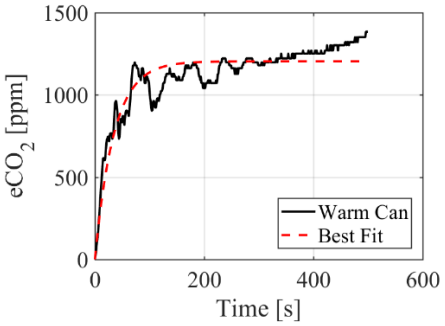

Figure 8*– First 500 seconds of data for "Warm Can" experiment with best fit line from the theoretical model.*

Table 1 *– Results from fitting model to data*

Experiment	τ_n	N
Cold Bottle	82.7	1140
Warm Bottle	62.2	503
Cold Can	227	628
Warm Can	34.7	1204

B. Analysis

The data collected shows that the secretion of carbon dioxide in beverages does roughly follow the model presented in Section I.C for a period of time of about 500 seconds. After this, as seen in Fig. 4, the model falls apart and the data becomes almost linear. One theory for why this happens is because once the beverage is opened, the most bubbles are generated in a short amount of time. Then, the diffusion of carbon dioxide from the surface of the beverage becomes the driving factor in carbon dioxide secretion as the rate of bubble production levels off.

Inspecting the data visually, specifically the first 500 seconds as shown in Fig. 5 – Fig. 8, shows that there is no apparent correlation between any of the levels or factors in this experiment. The data tends to follow a common trend, but the variance in the data does not allow for any valid conclusions to be drawn.

Although there are no strong conclusions drawn, the numbers gathered when fitting the model to the data are still presented in Table 1. The small value for τ_n and large value for N strengthens what is seen in the data. That is, the warm can secreted the most carbon dioxide in the smallest amount of time. This data also shows that there is no correlation between vessel type or temperature and bubble formation rate. The variance in this data could be motivation for further investigation.

IV. Conclusion

By using a CCS811 air quality sensor in conjunction within Arduino linked to Matlab, it was possible to measure the relative CO_2 levels secreted by various carbonated beverages. Four beverage types were tested: room temperature cans, room temperature bottles, cold cans, and cold bottles. The room temperature beverages were left out to warm up prior to the test, and the cold beverages were kept in the refrigerator. One by one, each beverage was placed in the sealed test chamber and opened, and the data was interpreted by the Arduino and recorded in

Matlab. By interpreting the data collected, it was found that the CO_2 secretion rate of these beverages did closely follow the theoretical model, but only up until about 500 seconds. It was also found that the warm can secreted the most CO_2. However, there was a great deal of variance in the data sets, which means there is no evidence of a correlation between vessel type or temperature and bubble formation rates.

It is possible that the large spikes seen in carbon dioxide at the beginning of each experiment were caused from a large initial rate of bubble formation which then leveled off, leaving secretion from the beverage surface itself as the main source of CO_2.

The largest source of error in the experiment is likely to have been the seal on the test chamber. This could have allowed CO_2 to escape. Error may also have come from running tests on different days where the CO_2 content in the atmosphere could have been different. Because our sensor only measures relative, not absolute, CO_2 levels, this would be difficult to detect. Also, if the beverage was shaken or disturbed during the experiment, this would also lead to a spike in bubble formation near the beginning of the experiment.

If this experiment were to be repeated, a more effective seal would have to be created between the buckets, gloves an viewing window, to prevent any leakage of CO_2 . This would give a higher quality of data, especially after t = 10 minutes. In addition, this experiment could be repeated more times for each container type and temperature and the results could have been averaged, which would also have given cleaner data. Ideally, a two way ANOVA test could be performed, which would give us a more definitive answer as to whether there was any correlation between secretion and either factor.

Appendix I

Arduino CCS811 Code

```
/******************************************************************************
  BasicReadings.ino

  Marshall Taylor @ SparkFun Electronics
  Nathan Seidle @ SparkFun Electronics

  April 4, 2017

  https://github.com/sparkfun/CCS811_Air_Quality_Breakout
  https://github.com/sparkfun/SparkFun_CCS811_Arduino_Library

  Read the TVOC and CO2 values from the SparkFun CSS811 breakout board

  This is the simplest example.  It throws away most error information and
  runs at the default 1 sample per second.

  A new sensor requires at 48-burn in. Once burned in a sensor requires
  20 minutes of run in before readings are considered good.

  Hardware Connections (Breakoutboard to Arduino):
  3.3V to 3.3V pin
  GND to GND pin
  SDA to A4
  SCL to A5

  Resources:
  Uses Wire.h for i2c operation

  Development environment specifics:
  Arduino IDE 1.8.1

  This code is released under the [MIT
License](http://opensource.org/licenses/MIT).

  Please review the LICENSE.md file included with this example. If you have
any questions
  or concerns with licensing, please contact techsupport@sparkfun.com.

  Distributed as-is; no warranty is given.
******************************************************************************/
#include "SparkFunCCS811.h"

#define CCS811_ADDR 0x5B //Default I2C Address
//#define CCS811_ADDR 0x5A //Alternate I2C Address

CCS811 mySensor(CCS811_ADDR);

void setup()
{
  Serial.begin(9600);
  Serial.println("CCS811 Basic Example");

  //It is recommended to check return status on .begin(), but it is not
  //required.
```

```
  CCS811Core::status returnCode = mySensor.begin();
  if (returnCode != CCS811Core::SENSOR_SUCCESS)
  {
    Serial.println(".begin() returned with an error.");
    while (1); //Hang if there was a problem.
  }
}

void loop()
{
  //Check to see if data is ready with .dataAvailable()
  if (mySensor.dataAvailable())
  {
    //If so, have the sensor read and calculate the results.
    //Get them later
    mySensor.readAlgorithmResults();

    Serial.print("CO2[");
    //Returns calculated CO2 reading
    Serial.print(mySensor.getCO2());
    Serial.print("] tVOC[");
    //Returns calculated TVOC reading
    Serial.print(mySensor.getTVOC());
    Serial.print("] millis[");
    //Simply the time since program start
    Serial.print(millis());
    Serial.print("]");
    Serial.println();
  }

  delay(10); //Don't spam the I2C bus
}
```

Appendix II

MATLAB-Arduino DAQ Interface Code

```
% --------------------------------------------------------------------
% ARDUINO_LIVE_PLOTTER
% Title: arduino_live_plotter
% Author: Matthew Hanley
% Date: 11/15/2017
%
% Purpose: Extract data from the serial port of the Arduino and plot it to
%          the screen.
% Exiting: Press "ctrl+c".
%
% On the arduino, make sure the value to be printed is sent to the serial
% port using something similar to:
%
% Serial.print(myValue); //print numeric value to serial
% Serial.println(); //print a new line
%
% The Serial.println() appends a new line to the end of the string. Without
% it, MATLAB does not know where to delimit the string.
% --------------------------------------------------------------------

% close all the plots and clear all the variables
clear all; close all;

%% Parameters you may need/want to change
serial_port = 'COM3';       % Port the arduino is connected to
num_plot_points = 1000;      % Number of points to plot on updating figure
x_label = 'Time';
y_label = 'ppm';

%% Main Procedure
disp('Press ctrl+c to terminate...')
s = serial(serial_port); % Sets up connections
fopen(s); % Opens the serial port for reading

% Function to run when user presses ctrl+c. Without it, the serial port
% will never be closed...
c = onCleanup(@()fclose(s));
figure
i=0; %init counter
fgets(s); % Get first string from serial port
plot_data = zeros(1,num_plot_points); %initialize the plotting array
time = -num_plot_points+1:0;
save_data=[];
t=[];

while(1)
    data=fgets(s); % Get first value
    data = data(1:end-2); % Chomp of the garbage characters

    % If we haven't gotten enough points yet, don't start scrolling the
    % data
    if (i < num_plot_points)
        plot_data(mod(i,num_plot_points)+1) = str2num(data); %get numeric val
```

```
    else
        % Scroll the data!
        plot_data = circshift(plot_data,-1); %shift it left
        plot_data(end) = str2num(data); %get the numeric value
    end
    plot(time,plot_data,'g*-') %plot it
    xlabel(x_label);
    ylabel(y_label);
    grid on
    i = i+1; %increment counter
    save_data(i) = str2num(data);
    t(i) = i;
    pause(0.001) %needed to update the plot
end
```

References

[1] Sébastien Uzel, Michael A. Chappell, and Stephen J. Payne*
Department of Engineering Science, University of Oxford, Parks Road, Oxford OX1 3PJ, U.K.
J. Phys. Chem. B, **2006**, *110* (14), pp 7579–7586
DOI: 10.1021/jp056531x
Publication Date (Web): March 21, 2006