Design and Development of an Arduino based Real Time Monitoring System for Measuring NOx Emission from Vehicular Exhaust.

Abstract

The goal of my project is to find a way to remove the NOx emission from air particularly that emitted from automobile exhaust. NOx is one of the most dangerous pollutants in existence, not only being highly corrosive and toxic itself, but also catalyzing the formation of ground level ozone from oxygen. Since my project focuses on NO_2 removal, I needed a device to measure the emission of NO_2. However, sensors that have capabilities that I needed were extremely expensive. So, I decided to build a device. The measuring device uses a gas sensor - MICS 2714 to measure NO_2 concentration and an arduino nano as the microcontroller. The system also records the data every 30 seconds onto a storage device (SD card) and displays the data on a LCD screen. It is controlled via two buttons, one to reboot the device in the event of a failure, and the other to take user input. I had tried to use another gas sensor MICS 6814 but the result from that was unreliable. The final device however, was able to measure the gas emission. The voltage measurement which is the raw input from the sensor is converted to ppm of NO_2 and is recorded on a microSD card every 30 second. My device is able to measure the difference in gas concentration between the initial input (exhaust) and final output which passes through the carbon adsorption layer. The device uses the conversion formula similar to the one used in Libelium Gases V3.0. In the future, I would like to calibrate the sensor using the known concentration of NO_2, however, the necessary calibration cylinders were unavailable to me at this time.

Design

The device to measure the emission of NO_2 required 5 main capabilities :

- Measure the concentration of NO_2 in the initial exhaust input and after it has passed through the adsorption layer.
- Record the above data to a SD card every 30 seconds.
- Display the data on a screen while the test is being performed.
- Give accurate results outdoors and in a snowblower exhaust.
- Run on its own without a computer and only a power outlet.

The final device contained five components :

- a MICS 2714 NO2 sensor to take the readings
- an I2C LCD display to show the readings and other messages
- an arduino nano to control the device
- a microSD card and adapter to store the data
- a control board containing two buttons and an LED.

I decided to use an arduino nano because of its small size and because it is easy to solder. However, it has no way of storing large amounts of data after reboot. Because of this, I had to add an external storage device. I used a microSD card to store the data. The full circuit connections are shown in Figure 1a and the actual device is shown in Figure 1b.

Arduino based NOx measurement system.

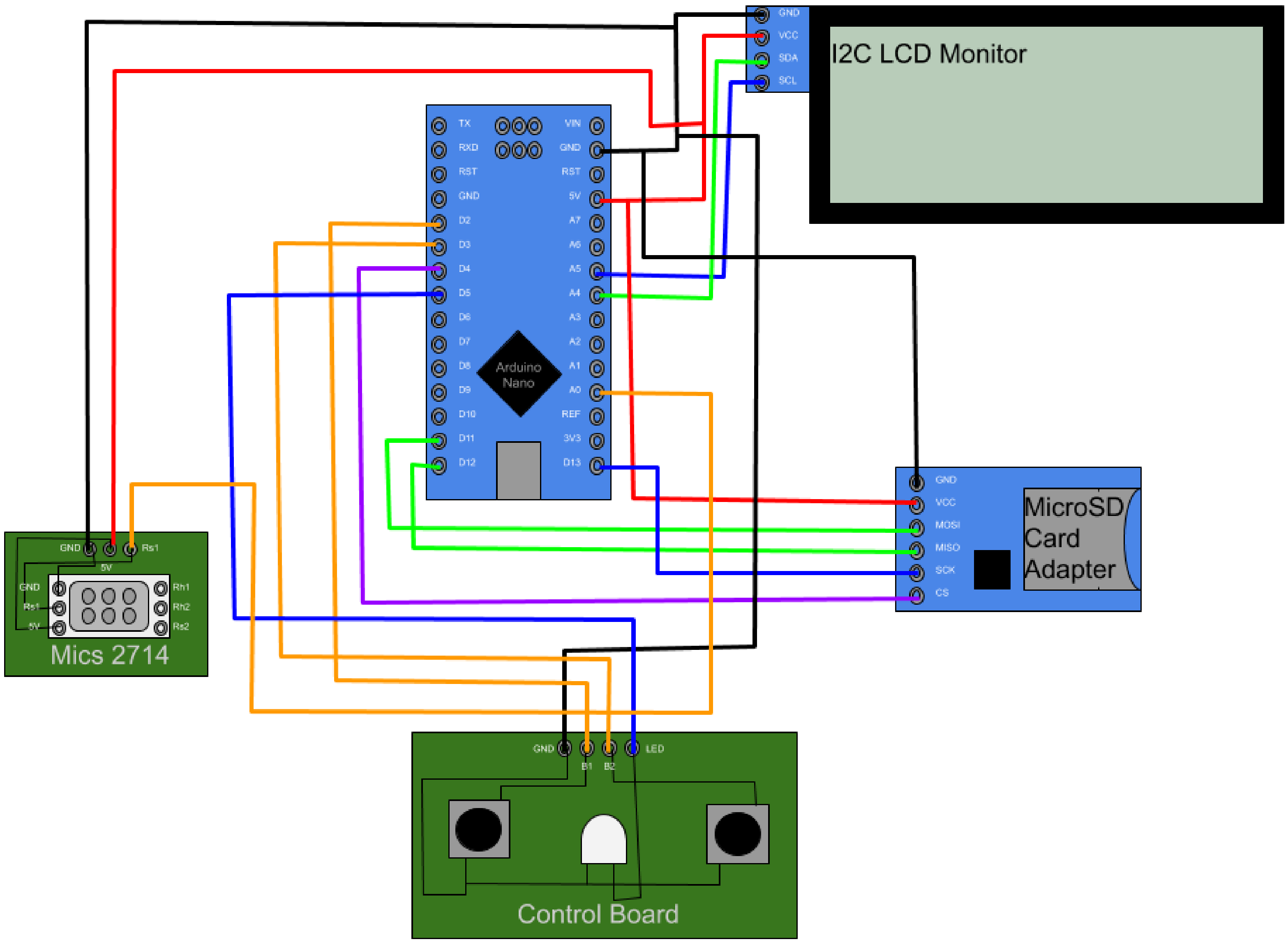


Figure 1a

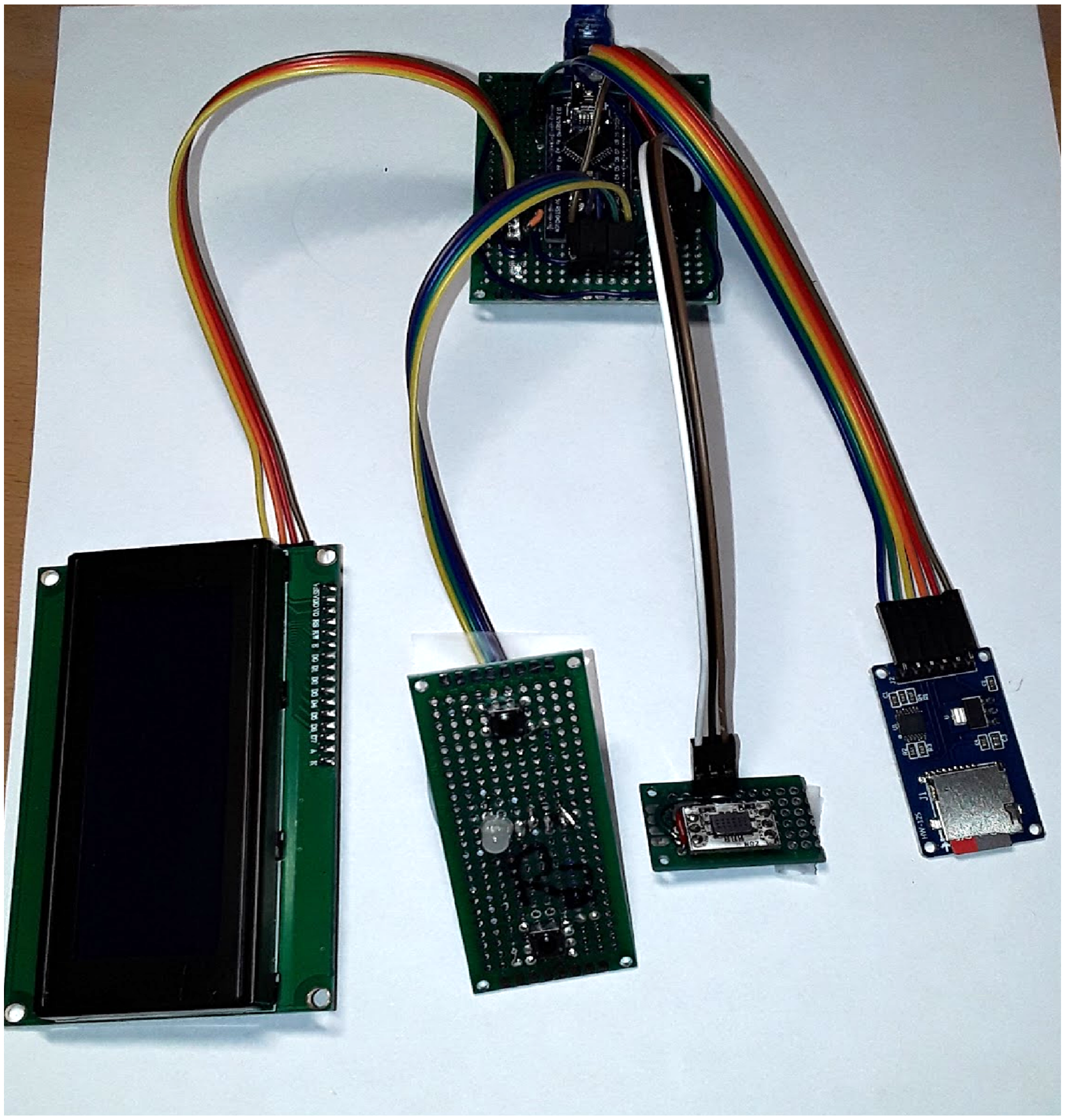


Figure 1b

For the first prototype I had two sensors and no control panel. It was difficult to use and often gave erroneous results. I found multiple flaws in this device:

- It could not take user input, all it did from the moment it was plugged in was start writing to the SD card.
- The arduino could not supply enough power to power both sensors, only one sensor was accurate, making the device even more difficult to use.

- The gas sensor I used, the Mics 6814, could only operate accurately at a particular temperature range, requiring that the sensor be given 10-20 minute warm-up times and that the temperature be very close to 25˚ C.

For my second prototype, I fixed all of these issues. I changed the prototype to include only one sensor and have a control panel. The control panel allowed the user to communicate with the machine and made the device much easier to use. To resolve the sensor issue, I used one Mics 2714. The Mics 2714 operates at a wider range of temperatures. However, by using only one sensor, I can only get the real time data from the treated gas and I have to assume that the NO2 output from the snowblower is constant. This means that the file that stores the untreated gas ppm only has one data point but the treated gas values are being taken in real time. Now we will get into a more in-depth look at each of the parts.

Device Parts

SD Card Reader

The microSD card is used to store the data from the sensor. Figure 3 shows both the microSD card and the reader. Since it has an on-board voltage regulator, it can be powered on both 5V and 3.3V It communicates with the board using SPI, Serial Peripheral Interface, communication protocol. SPI uses four pins, SCK, MOSI, MISO, and SS. SS is Slave Select, it allows the master to select a certain slave to communicate with. I only had one device, so this pin wasn’t very important. SCK is the clock pin, it pulses to allow both the master and the slave to communicate synchronously. MOSI stands for Master Out Slave In and is the connection through which the master writes to the slave. MISO stands for Master In Slave Out and is the connection through which the master takes data from the slave. To avoid having to write the SPI communication code, I used the SD.h library. This allowed me to use pre-made functions and made the code much easier to write.

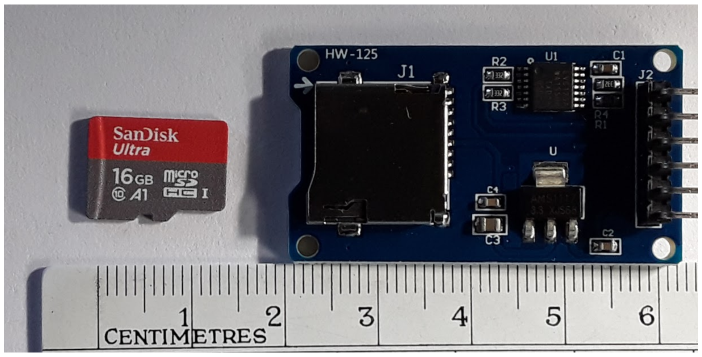


Figure 3

Gas Sensors

The Mics 2714 and the Mics 6814 are NO2 sensitive gas sensors. Figure 4a shows the Mics 2714 and figure 4b shows the Mics 6814. Below is a comparison of the sensors.

Mics 2714		Mics 6814	
Pros	**Cons**	**Pros**	**Cons**
Wide temperature range.	Can only sense NO2.	Can sense CO, NH3, and NO2.	Require specific temperature range.
Not cross sensitive to any gases.	Requires Calibration.	Calibration Free	Cross sensitive to hydrocarbons.
Industrial Grade sensor.	Hard to find in usable form, is mostly sold as a SMD device.	Very easy to find in THT breakout board form.	Very un-precise, useful only for high, medium, and low indicators.
Heats up in 30 seconds			Takes 10 mins to heat up.

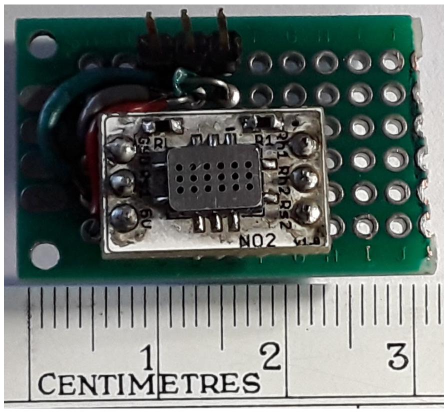


Figure 4a

The Mics 2714 is a much better sensor than the Mics 6814. However, it is a SMD sensor, meaning that it needs to be soldered on a printed circuit board. The small silver rectangle in figure 4a is the sensing device. On top of this, since the sensor is quite sensitive, it needs to be soldered in a neutral air reflow soldering oven. I had neither the circuit board nor the oven. Because of this, I tried to use the Mics 6814 on a breakout board (A breakout board is a board onto which a component is placed that makes a component easier to use. In this case, it converted the SMD sensor to a THT device).

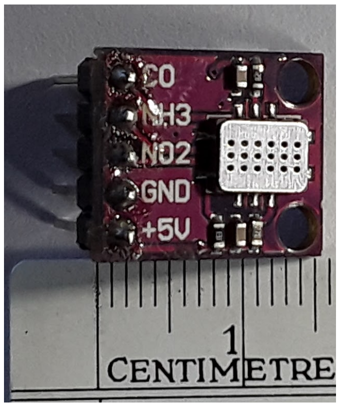


Figure 4b

However, the Mics 6814 didn’t fit my needs, its accuracy was further reduced in the temperature range outdoors, it was cross sensitive to hydrocarbons like ethanol, often gave erroneous results, and it required long heat up periods of ten to twenty minutes. When designing my second prototype, I designed it to include only one sensor. I also purchased a Mics 2714 breakout board.

The Mics 2714 gave much better results, however, it required calibration that I did not have the tools to do. So, I used a formula from waspmote gas sensor website which also uses mics 2714 in the device to get an approximation of the ppm. The formula is

$$10^{\left(\frac{\log_{10}\left(\frac{1.8\cdot 2}{mV}-2\right)-2.176}{-1.737}\right)}$$

where mV is the milliVolt reading taken from the sensors. This final system worked quite well with the NO2 value increasing when the sensor was exposed to the snowblower exhaust. However, I do not know whether these values are 100% accurate as I couldn't test the exhaust against a properly calibrated NOx measuring device. However my project required the concentration difference between the initial input exhaust and after it has passed through a carbon adsorption layer and it was able to measure the required data.

Serial I2C LCD

The I2C LCD is used to display the data from the sensor as well as print messages to instruct the user what to do next. Figure 5 shows the LCD screen. It uses I2C (Inter-IC) communication protocol to communicate with the arduino. This shortens down the sixteen pins in a normal LCD down to only four, 5V power, ground, SCL and SCL. SCL is the clock, just like in SPI. SDA is the data transfer pin. Because I2C does not have a slave select pin, it is necessary to specify the hexadecimal address of the device to be used. Since I was only using one LCD, this was not important to me. Unlike the arduino uno, the arduino nano doesn't have a separate SDA and SCL pin. Instead, analog input pin four is SDA and analog input pin five is SCL. To avoid having to write the I2C communication code, I used the libraries Wire.h and LiquidCrystal_I2C.h.

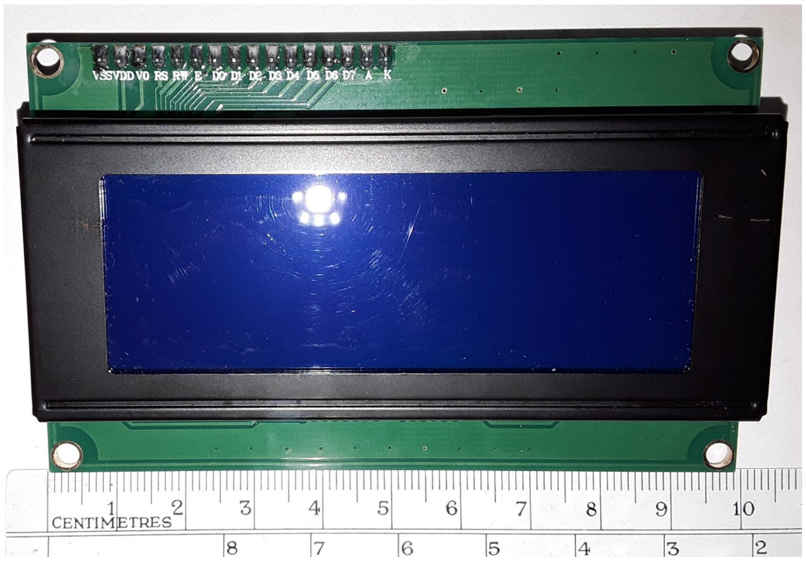


Figure 5

Arduino Nano

The arduino nano is a programmable microcontroller with an 14 pin GPIO interface, 8 analog inputs and an ATmega328 as the CPU. Each GPIO pin can be set to either INPUT or INPUT_PULLUP. INPUT means that the pin when the pin is neither LOW (connected to ground) or HIGH (connected to 5V), it will randomly fluctuate between the two. When INPUT_PULLUP is enabled, the inbuilt pull-up resistor will be activated and the pin will default to HIGH unless otherwise specified. The arduino nano also has 30 KB of open flash memory, memory that the code is stored in. This cannot be used by the program itself. The arduino has 1 KB of EEPROM, usable memory that can stay after reboot and 2 KB of SRAM, program memory that stores variables and is deleted after reboot. The arduino Nano was powered using a standard power outlet to USB converter and a mini USB to standard USB connector cable. Figure 6 shows the Arduino Nano on it's board.

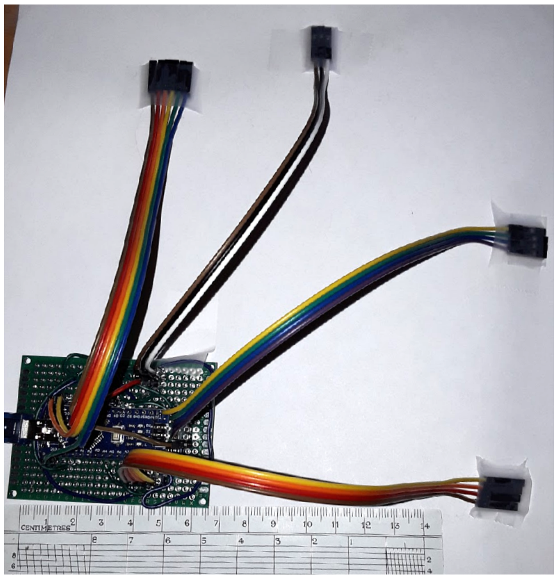


Figure 6

Control Board

The control board consisted of an LED and two pushbuttons. The device has four pins, LED input, button one output, button two output, and ground. When the button is pressed, the button's pin goes to LOW. To make sure the value doesn't fluctuate between HIGH and LOW when the button is not pressed, the code activates the pullup resistor for both the button pins. Figure 7 shows the control panel.

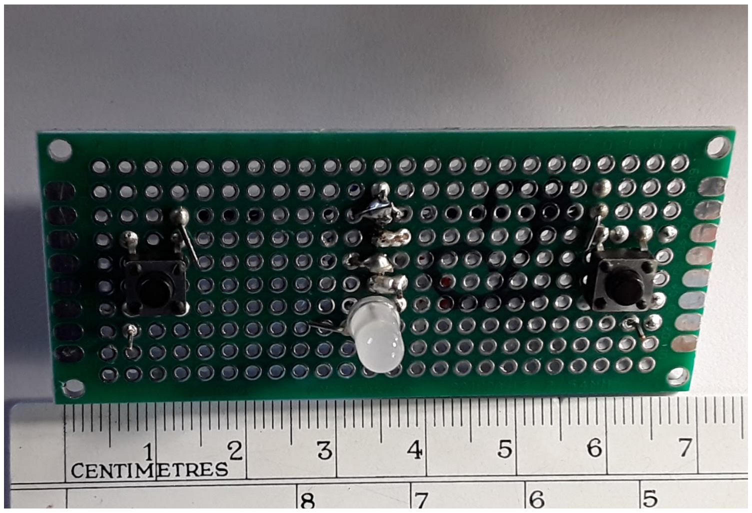


Figure 7

Code

The code for the device has four stages, heat up, untreated input measurement, treated data measurement, and shutdown. Figure 8 is a detailed flow chart.

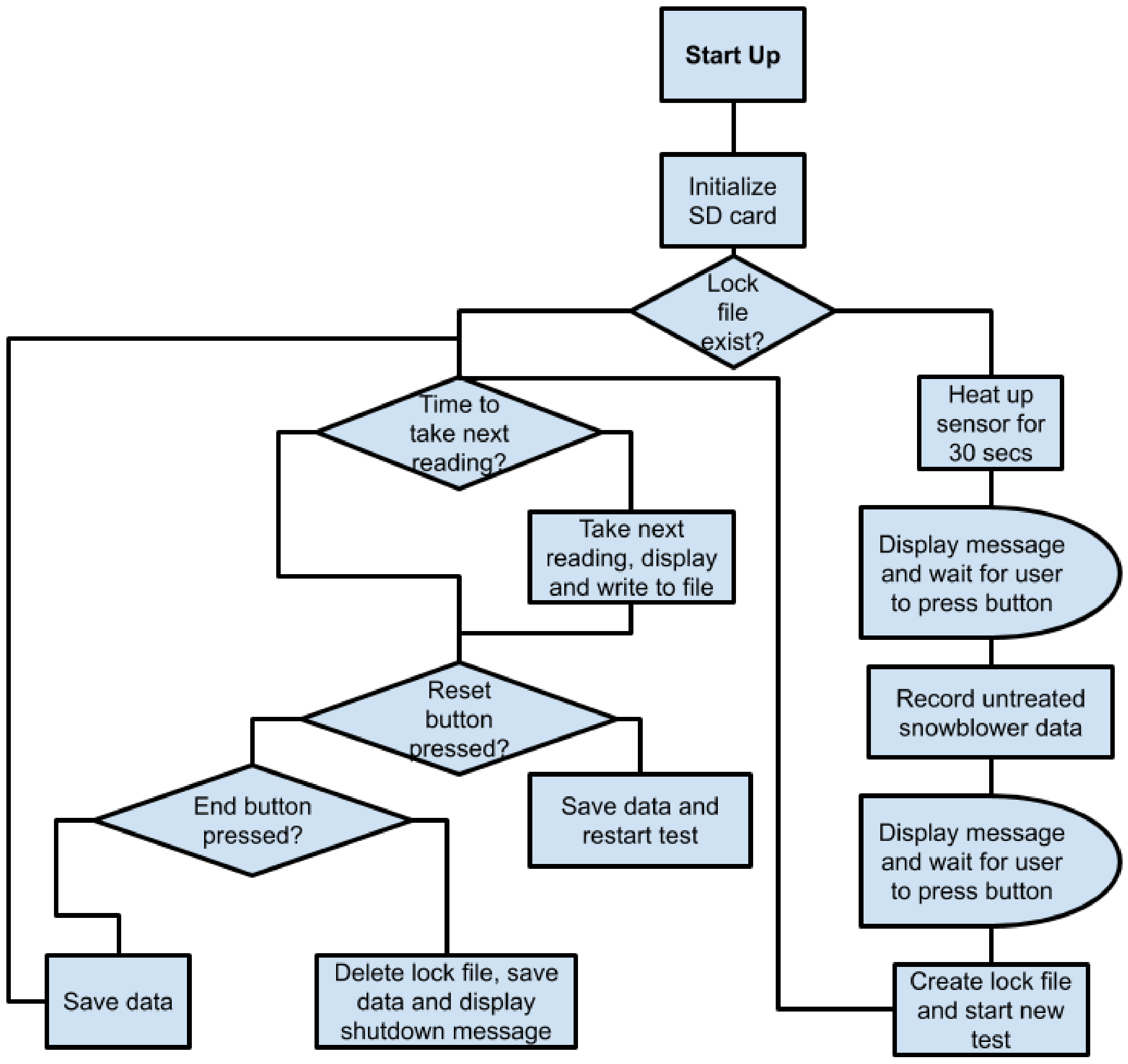
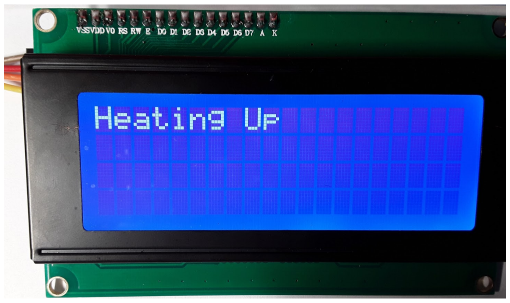

Start Up
Initialize SD card
Lock file exist?
Heat up sensor for 30 secs
Display message and wait for user to press button
Record untreated snowblower data
Display message and wait for user to press button
Create lock file and start new test
Time to take next reading?
Take next reading, display and write to file
Reset button pressed?
Save data and restart test
End button pressed?
Save data
Delete lock file, save data and display shutdown message

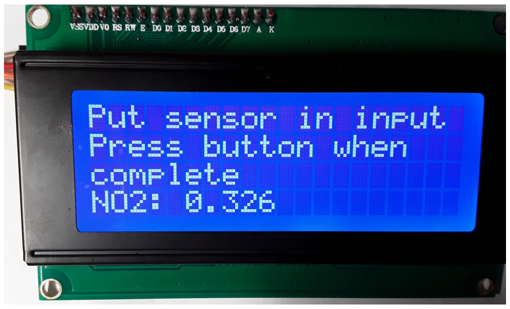
Figure 8

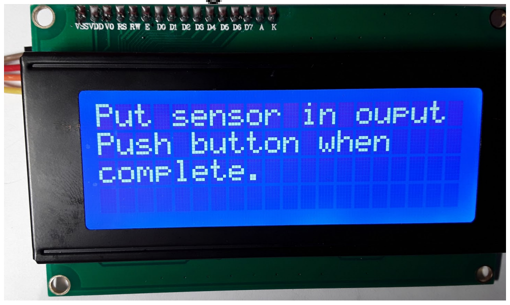
Code Steps

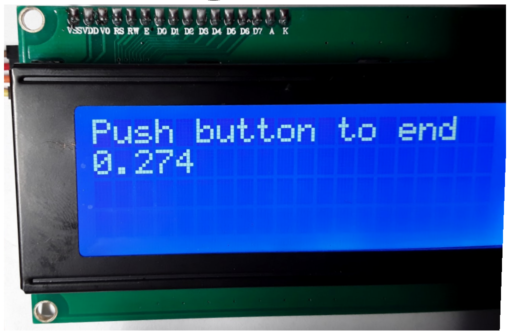
To start, the code checks if the lock file exists. The lock file is created if the device is in the middle of a test. That way, if the device loses power, the program gets right back to taking data, instead of re-entering the heat up stage. If the lock file exists, the code skips the heat up and untreated gas measurement and gets back to the data collection. If the lock file doesn't exist, then it starts by heating up the sensor. The Mics 2714 only needs 30 seconds, so this doesn't take long. Figure 9 shows the LCD in the heat up phase. Then, once the device has warmed up, it takes the untreated gas reading, which is labeled on the LCD as *input*. Figure 10 shows the LCD at this stage. When the user presses the button, it records the NO2 reading for the untreated gas into the file and moves onto the next step. Then it tells the user to put the sensor behind the treatment device and press the button, as shown in Figure 11. Then, as shown in Figure 12, the device will start taking the NO2 measurement every 30 seconds. It continues this until the button is pressed. Then, it deletes the lock file and displays the message "Please Shutdown".

The code is shown in Appendix A.


```
Heating Up
```
Figure 9


```
Put sensor in input
Press button when
complete
NO2: 0.326
```
Figure 10


```
Put sensor in ouput
Push button when
complete.
```
Figure 11


```
Push button to end
0.274
```
Figure 12

Output Data

The code has five separate files:

1. The untreated gas ppm called *Input.txt*
2. The treated gas ppm called *Data.txt*
3. The raw voltage data from the untreated gas called *RawInput.txt*
4. The raw voltage data from the treated gas called *RawData.txt*
5. The lock file called *Lock.txt*

When a new test begins, the code writes ****New Test Begins Here****. When the sensor is introduced into a new environment, the value fluctuates for some time before stabilizing.

This is a some sample data from Input.txt:

```
***New Test Begins Here***
0.41
***New Test Begins Here***
0.45
```

This is a some sample data from Data.txt:

```
***New Test Begins Here***
0.25
0.25
[More data points which have been omitted]
0.23
```

This is a some sample data from RawInput.txt:

```
***New Test Begins Here***
565
***New Test Begins Here***
414
```

This is a some sample data from RawData.txt:

****New Test Begins Here****

427

432

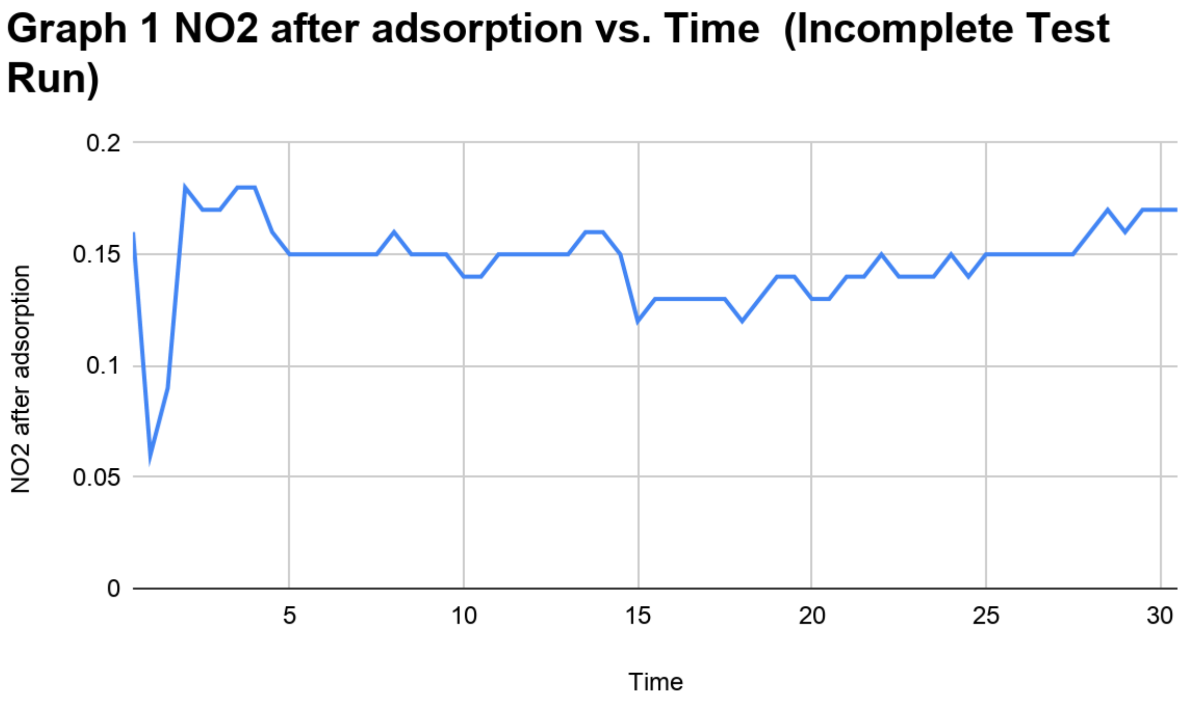
[More data points which have been omitted]

369

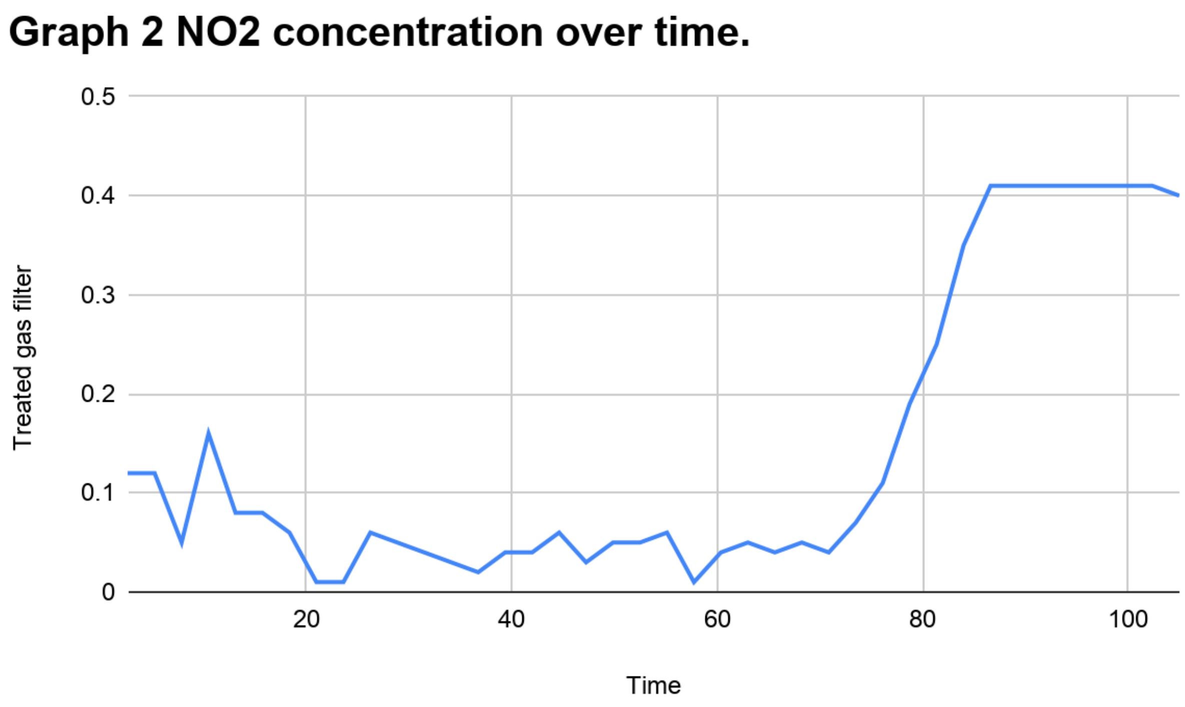
The data from the SD card is then transferred to a computer, where I can graph the data and see how the test went.

Results

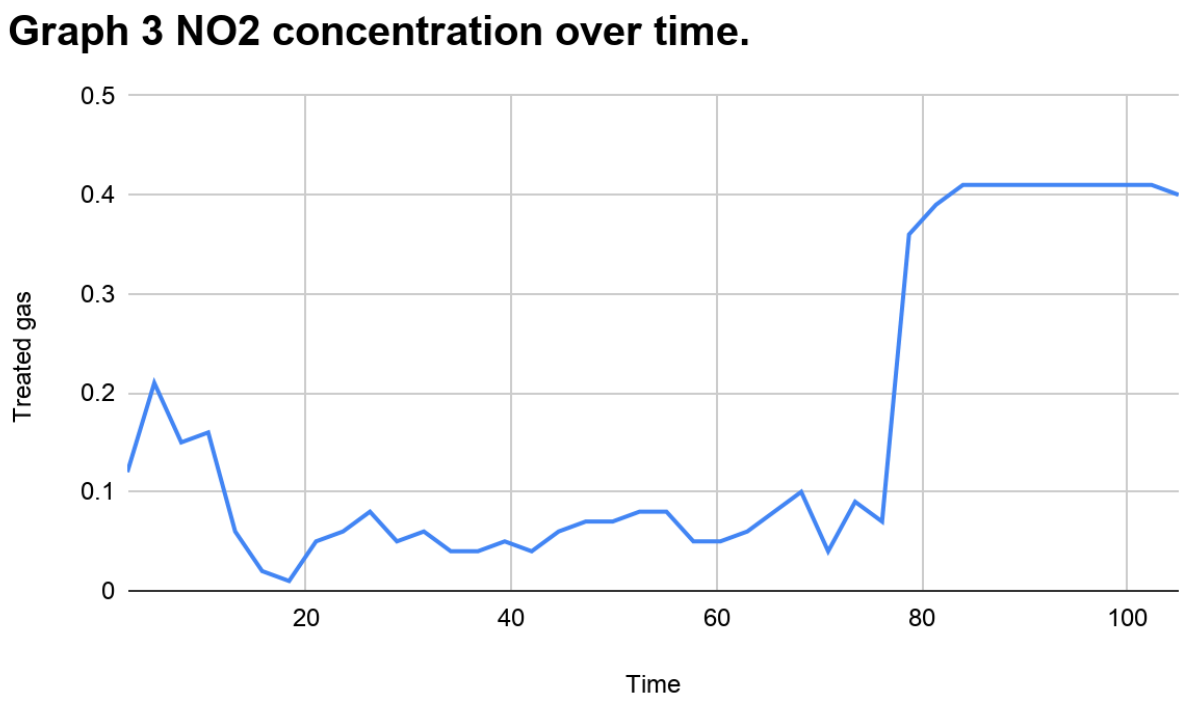
Graph 1 - This test didn't run to completion as the snowblower ran out of gasoline after 30 mins.



Graph 2 - This test ran till the activated carbon was saturated. Outliers were the sensor gave 0 ppm were remove



Graph 3 - This test ran until the charcoal was saturated.The same outliers were removed.



Discussion

At the beginning of test 2, for around 5 mins, the sensor fluctuates, then it evens out. The sensor's accuracy is 0.05, so while the values seem to fluctuate, it is just random fluctuations. With error taken into account, there is very little fluctuation in the ppm. This is because of the geometry of the adsorption device. The charcoal is in a long thin tube, so only a small fraction of the charcoal is exposed to the NO_2 at a time. Little to no NO_2 is allowed into another section before the carbon in front of it is completely adsorbed. Then, the process repeats with the next section of carbon. This keeps happening until the last section of carbon is used. Then, with nothing to stop it, the NO_2 comes out with little to no adsorption. At this point, the test is over. The same pattern is shown in test 3. Test 1 shows only the beginning of this process, so the NO_2 value never climbs to the final 0.5 ppm. However, the charcoal wasn't well packed in this test so more of the NO_2 escaped.

Conclusion

This device worked according to my original standards, it could run on its own, it could take the readings from the test setup. The gas concentration in ppm may not be absolutely accurate though. In the future, I would like to calibrate the sensors using known calibration cylinders. This device greatly improved the quality of my experiment, giving me an inexpensive way to measure NO2 concentration.

References

- Micro SD Card Micro SDHC Mini TF Card Adapter Reader Module for Arduino. Retrieved from http://datalogger.pbworks.com/w/file/fetch/89507207/Datalogger%20-%20SD%20Memory%20Reader%20Datasheet.pdf

- Mics 2714 datasheet. Retrieved from https://www.sgxsensortech.com/content/uploads/2014/08/1107_Datasheet-MiCS-2714.pdf
- Mics 6814 datasheet. Retrieved from https://sgx.cdistore.com/datasheets/sgx/1143_datasheet%20mics-6814%20rev%208.pdf
- Libelium Smart Gases 3.0. Retrieved from http://www.libelium.com/downloads/documentation/gases_sensor_board_3.0.pdf

}

Appendix A: NO_2 Monitoring Code

```
//include libraries
#include <SPI.h>
#include <SD.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 20, 4); //define the I2C lcd address to be at 0x27 hex.
//Mics 2714 Pin
int SensorPin = A0;
//Button to stop and start test
int Button = 2;
//Button to reset test
int ResetButton = 3;
//LED indicator - not used yet.
int LED = 5;

//PPM value of NO2
float NO2ppm;
//Raw sensor value
int AnalogVal;
unsigned long CurrentTime = 0;
unsigned long AlarmTime = 0;
int gapTime = 30000;

//Define Files
File RawData; //Stores raw voltage for data
File RawInput;//Stores raw voltage for input
File Data; // Stores ppm for data
File Input; // Stores ppm for input
File lock; // The lock file exists if the code was running before restart

void setup() {
  pinMode(Button, INPUT_PULLUP);
  pinMode(ResetButton, INPUT_PULLUP);
  //Initialize SD card
  if (!SD.begin(4)) {
    Serial.println("initialization failed!");
    return;
  }
  //Open the files
  lcd.init();  //initialize the lcd
  lcd.backlight();//activate the backlight
  if (!SD.exists("lock.txt")) { // the lock file indicates that the code was running before reset, and
that it should resume void loop without going through setup
    //Open files
    RawData = SD.open("RawData.txt", FILE_WRITE);
    RawInput = SD.open("RawInput.txt", FILE_WRITE);
    Data = SD.open("Data.txt", FILE_WRITE);
    Input = SD.open("Input.txt", FILE_WRITE);
    //Write new data line
    RawData.println("***New Test Begins Here***");
    RawInput.println("***New Test Begins Here***");
    Data.println("***New Test Begins Here***");
    Input.println("***New Test Begins Here***");
    //Close files
    RawData.close();
    RawInput.close();
    Data.close();
    Input.close();
    //Display on LCD
    lcd.setCursor(0, 0);
    lcd.print("Activated");
    delay(2000);
    lcd.clear();
    //Heat up sensor for 30 secs
    heatUp();
    // Read initial data
    readInitialData();
    lcd.clear();
    delay(1000);
    //Ready for testing
    lcd.setCursor(0, 0);
    lcd.print("Put sensor in output");
    lcd.setCursor(0, 1);
    lcd.print("Push button when");
    lcd.setCursor(0, 2);
    lcd.print("complete.");
    while (digitalRead(Button) != LOW) {
    }
    //Create lock file to indicate test started.
    lock = SD.open("lock.txt", FILE_WRITE);
    lock.close();
    lcd.clear();
    delay(1000);
  }
}

//Reset the arduino board
void(* resetFunc) (void) = 0;

//Processing code
void loop() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Push button to end");
  //Open file for writing data
  Data = SD.open("Data.txt", FILE_WRITE);
  RawData = SD.open("RawData.txt", FILE_WRITE);
  while (digitalRead(Button) != LOW) {
    if (alarm()) {
      //Get average of 5 reads
      AnalogVal = averageRead();
      //Calculate PPM value
      NO2ppm = calculatePPM(AnalogVal);
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Push button to end");
      lcd.setCursor(0, 1);
      lcd.print(NO2ppm, 3);
      Data.println(NO2ppm);
      RawData.println(AnalogVal);
    }
    if (digitalRead(ResetButton) == LOW) {
      Data.close();
      RawData.close();
      resetFunc();
    }
  }
  //Close files
  Data.close();
  RawData.close();
  lcd.clear();
  SD.remove("lock.txt");
  lcd.setCursor(0, 0);
  // System ok to shutdown
  lcd.print("Please shutdown");
  while (true) {
  }
}

bool alarm() {
  bool Output = false;
  CurrentTime = millis(); //set current time to system time
  if (CurrentTime >= AlarmTime) {
    CurrentTime = millis();
    AlarmTime = CurrentTime + gapTime;
    Output = true;
  }
  return (Output);
}
//Read initial input data
void readInitialData() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Put sensor in input");
  lcd.setCursor(0, 1);
  lcd.print("Press button when ");
  lcd.setCursor(0, 2);
  lcd.print("complete");
  while (digitalRead(Button) != LOW) {
    lcd.setCursor(0, 3);
    lcd.print("NO2: ");
    lcd.setCursor(5, 3);
    NO2ppm = calculatePPM(averageRead());
    lcd.print(NO2ppm, 3);
    delay(1000);
  }
  Input = SD.open("Input.txt", FILE_WRITE);
  RawInput = SD.open("RawInput.txt", FILE_WRITE);
  int Val = averageRead();
  RawInput.println(Val);
  Input.println(calculatePPM(Val));
  RawInput.close();
  Input.close();
}

//Give 30 seconds heat up time
void heatUp() {
  lcd.setCursor(0, 0);
  lcd.print("Heating Up");
  delay(30000);
  lcd.clear();
  lcd.print("Done");
  delay(1000);
}

//Calculate PPM value
float calculatePPM(int analogVal) {
  float mvoltage = (analogVal * (5.0 / 1023.0)) / 1000;
  float rawSensorVal = ((1.8 * 2) / mvoltage) - 2;
  float ppm = pow(10, (log10(rawSensorVal) - 2.176) / (-1.737));
  return (ppm);
}

//Calculate average sensor value based on 5 reads
int averageRead() {
  int read1 = analogRead(SensorPin);
  int read2 = analogRead(SensorPin);
  int read3 = analogRead(SensorPin);
  int read4 = analogRead(SensorPin);
  int read5 = analogRead(SensorPin);
  int readAvg = (read1 + read2 + read3 + read4 + read5) / 5;
  return (readAvg);
}
```