Personal Research Notebook 2025

Genevieve Reeves

Augusta University
Department of Physics and Biophysics

Contents

1 11/11/2025 Programming Common Arduino Temperature Sensors 2
- 1.1 Notes Overview . . . 2
- 1.2 Introduction . . . 2
- 1.3 Materials and Methods . . . 2
  - 1.3.1 The Hardware . . . 2
  - 1.3.2 The Circuit(s) . . . 2
  - 1.3.3 The Code . . . 5
- 1.4 Results . . . 6
  - 1.4.1 Serial Output from Code . . . 6
  - 1.4.2 Consistency of Temperature Readings Over Time . . . 7
- 1.5 Discussion and Conclusions . . . 7

2 11/24/2025 Introduction to The Garmin LiDAR Sensor 8
- 2.1 Introduction . . . 8
- 2.2 Materials and Methods . . . 8
  - 2.2.1 The Hardware . . . 8
  - 2.2.2 The Circuit . . . 8
  - 2.2.3 The Code . . . 9
- 2.3 Results . . . 14
- 2.4 Discussion and Conclusion . . . 14

3 11/27/2025 Exploration of the Metal Oxide Substrate (MOS) Sensor 16
- 3.1 Introduction . . . 16
- 3.2 Materials and Methods . . . 17
  - 3.2.1 The MOS Sensor . . . 17
  - 3.2.2 MOS Sensor End-User Calibration . . . 17
  - 3.2.3 The Circuit . . . 19
  - 3.2.4 The Code . . . 19
  - 3.2.5 Initial experimental design . . . 20
- 3.3 Results . . . 20
- 3.4 Discussion and Conclusion . . . 21

Bibliography 23

Chapter 1

11/11/2025 Programming Common Arduino Temperature Sensors

1.1 Notes Overview

A detailed description of building and working with two different temperature sensors - the TC74 using I2C and the LM35 using analog measurements - on an Arduino microcontroller are detailed.

Keywords: TC74 temperature sensor , LM35 temperature sensor , sensors , chemical sensing

1.2 Introduction

The vents behind the office are constantly spewing steam from them. The chemical composition of these clouds is of interest to me as I am concerned about what things we may be exposed to on a daily - hourly - moment-to-moment basis. One of my first steps in building a sensor that can remotely sample components of these clouds is to get hands-on with sensors and build up a repertoire of knowledge that will be used to build a sensor array. To start, I wanted to look at temperature sensing. While temperature sensing isn't incredibly helpful in the analysis of steam, all great endeavors must start somewhere.

1.3 Materials and Methods

1.3.1 The Hardware

This project involves the use of an Arduino UNO, two 4.7kΩ resistors which act as pull-ups for the HC74, the I2C-based TC74 temperature sensor, and the analog-based LM35 temperature sensor. These two ICs have pinouts as shown in Figures 1.1a and 1.1b below.

1.3.2 The Circuit(s)

Proposed circuits derived from online resources and the datasheets for the two ICs are shown in Figures 1.2 and 1.3 below. A picture of the working circuit is also shown in Figure 1.4 below.

(a) The pinout of the TC74 temperature sensor

(b) The pinout of the LM35 temperature sensor

Figure 1.1: The different different temperature sensors and their pinout diagrams

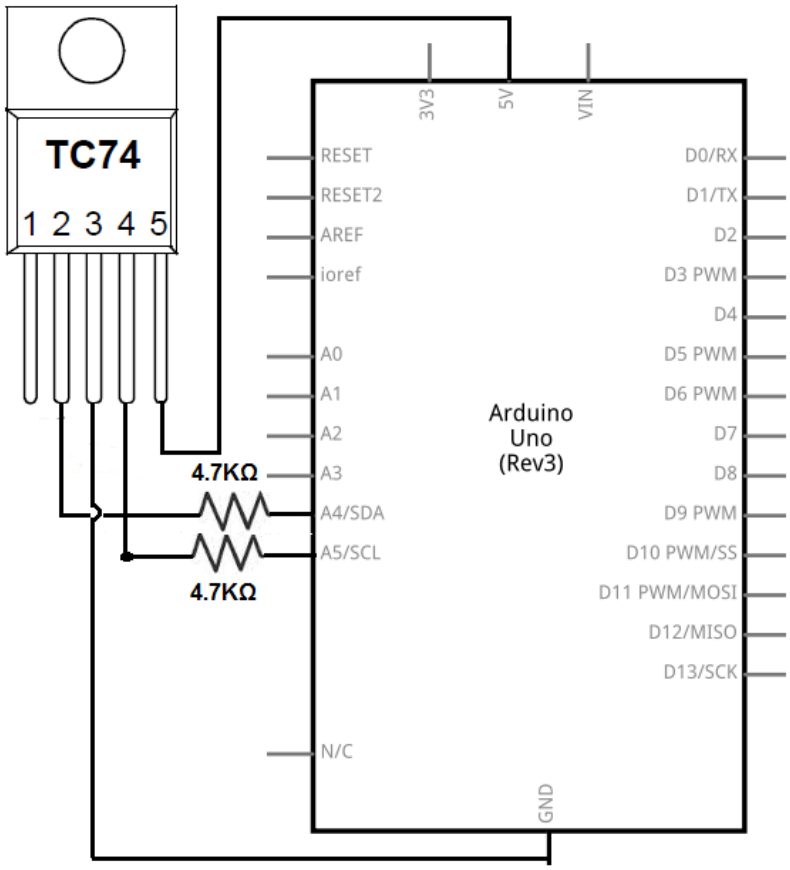


Figure 1.2: The basic circuit of the TC74 temperature sensor hooked up to the Arduino Uno is shown. Illustrated in the figure is the exact pins to be used as well as the pull up resistors required for I2C communications. **NOTE:** This circuit appears different from the circuit I actually built. Namely, the two pull up resistors in the circuit that I built are running between 5V and the SDA and SCLK pins not between the pins and Arduino.

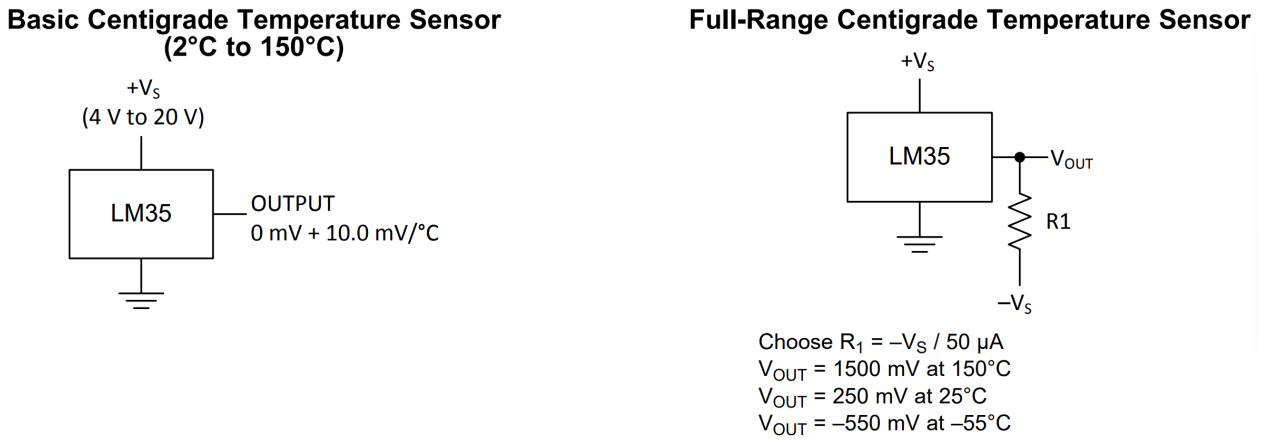


Figure 1.3: Two possible circuits for the LM35 are shown with a basic temperature sensor shown in the left diagram and a wider range temperature sensor involving a voltage divider being shown at right.

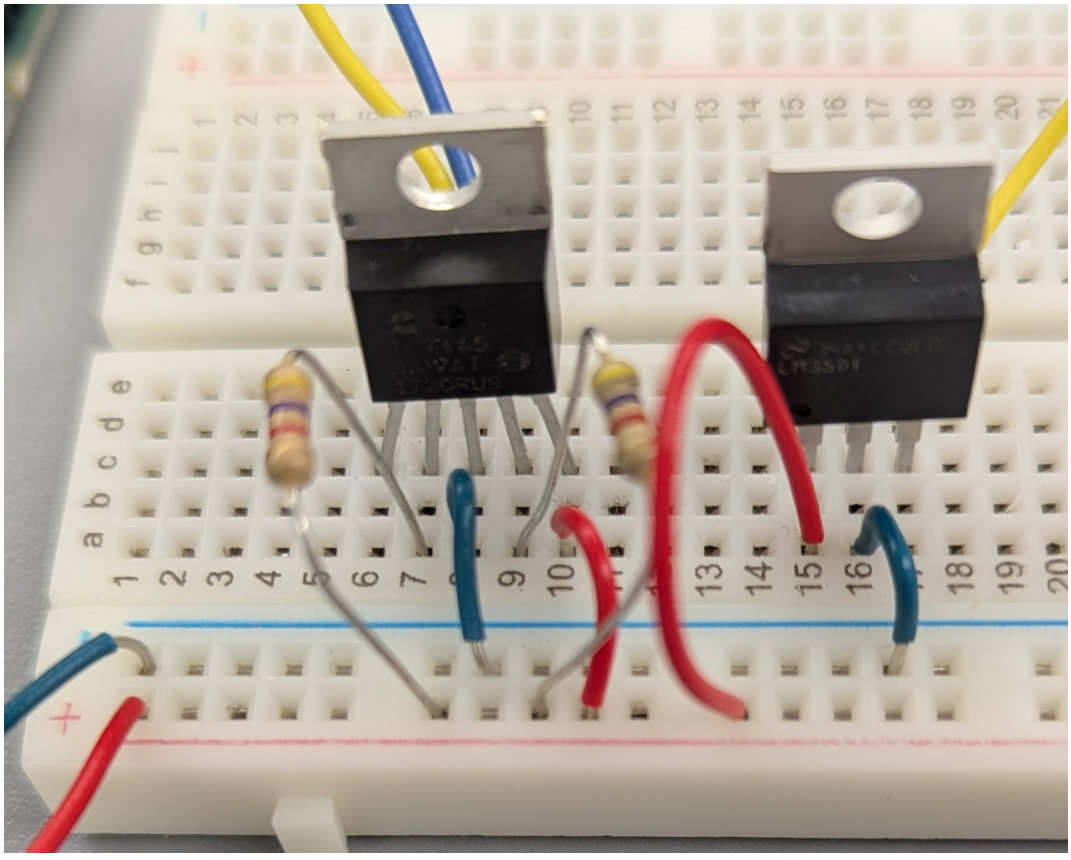


Figure 1.4: A picture of the working circuit. The picture shows the set up used to interface with the TC74 (left IC) and the LM35 (right IC) sensors.

1.3.3 The Code

Below is the code used in programming the Arduino Uno to communicate with the temperature sensors.

```
#include <Wire.h>
int ii;
// Define an index that will be used in for loops
int reg;
// Define the reg(ister) variable that will store the I2C address
long tempIIC  = 0;
// This long variable will store the sum of all I2C temp readings
long tempAnalog = 0;
// This long variable will store the sum of all analog temp readings
float avgIicTemp;
// This float will contain the average temp of the I2C sensor
float avgAnaTemp;
// This float will contain the average temp of the analog sensor

void setup() {
  Wire.begin();
  // Set up I2C communications
  Serial.begin(9600);
  // Start the serial monitor
  for(ii =0; ii<128; ii++){
  // Loop over all possible addresses
  Wire.beginTransmission(ii);
  // Ping the IC's register at a specific address
    if(Wire.endTransmission()==0){
 // If there is an ACK from the IC
      Serial.print("The temperature register is ");
   // Print out message
      Serial.println(ii, HEX);
   // Print out the hexadecimal representation of the register address
      reg = ii;
   // Save the register address (0x4D)
      break;
   // Break out of loop when address is found
    }
    delay(10);
  }
  Wire.write(0x01);
  // Tell replica to use specific address
  Wire.write(0x00);
  // Configure the replica
  Wire.endTransmission();
  // Stop transmission
}
```

```
void loop() {
  for(ii=0; ii<500; ii++){
  // Perform ii (500) iterations of measurements
    Wire.beginTransmission(reg);
 // Set up transmission to IC
    Wire.write(0x00);
 // Tell replica it's about to be read
    Wire.endTransmission();
 // Stop transmission
    Wire.requestFrom(reg, 1);
 // Request 1 byte of information (temperature) from the IC
    tempIIC += Wire.read();
 // Read in the information from the I2C sensor and compound add
    tempAnalog += analogRead(A0);
 // Read the analog voltage reading from the analog sensor and compound add
  }

  avgIicTemp = tempIIC/float(ii);
  // Average the compound sum over ii iterations
  avgAnaTemp = (tempAnalog/float(ii))*(5000.0/1023.0)/10.0;
  // Average the readings, multiply by the ADC step size,
  //divide by conversion factor (10mV/C)

  // Printing the average temperature readings
  Serial.print("My average I2C temperature is:     ");
  Serial.println(avgIicTemp, 2);
  Serial.print("My average analog temperature is:  ");
  Serial.println(avgAnaTemp, 2);

  // Reset the temperature readings so a new cumulative sum can be performed
  tempIIC = 0;
  tempAnalog = 0;
}
```

1.4 Results

1.4.1 Serial Output from Code

Using the code shown above, I was able to obtain outputs to the serial monitor as seen below. Over 1010 measurements of 500 averaged temperature readings in a cool, temperature-controlled room, the TC74 showed a value of 23.27 ± 0.29°C and the LM35 showed a temperature reading of 22.51 ± 0.06°C

```
The temperature register is 4D
My average I2C temperature is:     22.00
My average analog temperature is:  21.66
My average I2C temperature is:     22.49
My average analog temperature is:  21.65
```

1.4.2 Consistency of Temperature Readings Over Time

Seen in Figure 1.5 below is a plot of temperature readings over time. As can be seen in the figure, the analog LM35 does not run into bitwise errors associated with the temperature readings. This is because of the precision of the TC74 whose resolution is to 1 degree celcius.

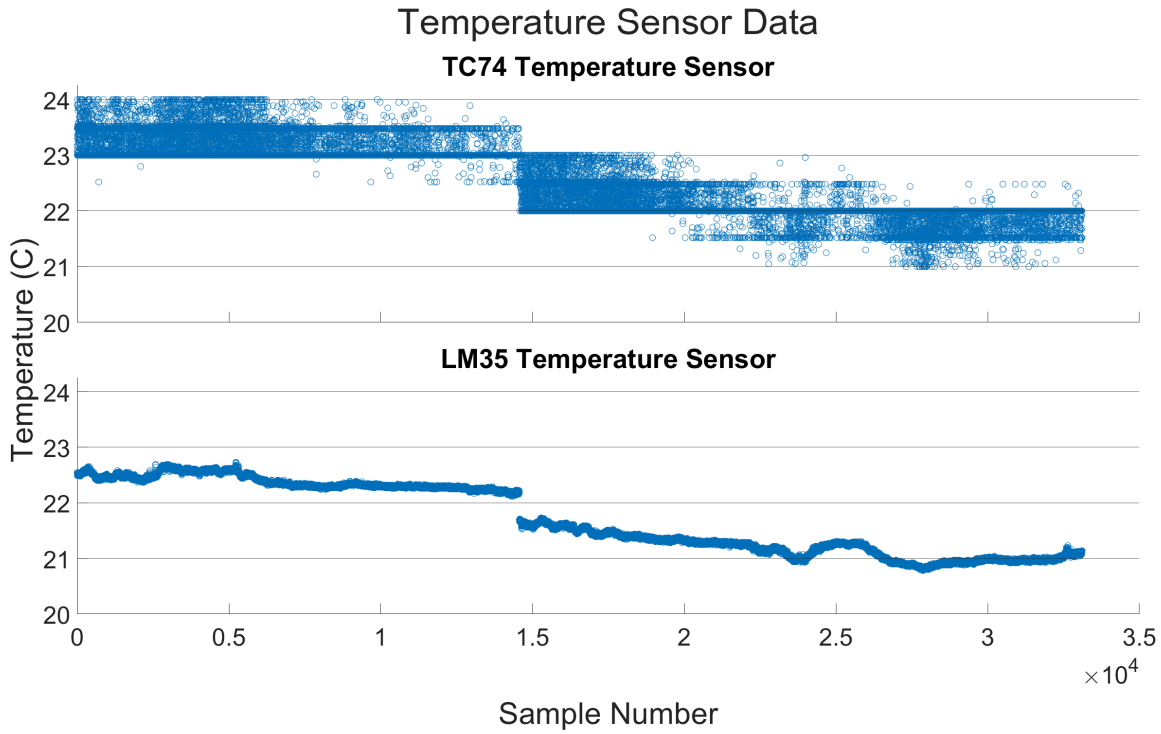


Figure 1.5: A scatter plot of over 34,000 averaged readings of the TC74 I2C temperature sensor and the LM35 analog temperature sensor.

1.5 Discussion and Conclusions

This brief overview of how to interface with a couple of well known temperature sensors provides a basis for further work in building a sensor array. I would like to note a couple of things, however. First, whenn I hooked up the LM35 to analog pin `A5`, I couldn't get any reading that made sense. This could entirely be the fault of the Arduino I was using, but it is worth noting that when I moved to pin `A0`, I was able to get the readings without any further issue.

The second thing to note is that the TC74 appears to fluctuate more in its measurements. I believe this to be caused by the bitwise resolution of 1°C whereas the LM35 uses the analog-to-digital converter on the arduino as its resolution. This means, with a headspace of 5V, and the arduino's 10-bit ADC, the resolution of the analog sensor is $5000mV/1023 = 4.89mV$. Given the conversion factor of $10mV/C$ that means we can detect changes in temperature of $4.89mV/(10mV/C) = 0.489°C$. This means that the LM35 has over twice the resolution of the TC74! I suppose that's what you get when you are 38% ($0.84) more expesnive.

Chapter 2

11/24/2025 Introduction to The Garmin LiDAR Sensor

A discussion of how to build and program the Arduino UNO to use the Garmin LiDAR sensor and how to use the arduino statistic library to perform averaging and population standard deviation.

Keywords: LiDAR, Garmin LIDAR-Lite Optical Distance Sensor - V3, sensors, Statistic library

2.1 Introduction

We are currently working on a project involving sensing levels of grass (the "smart (gr)ass" project), but the ultimate goal of this project is to build and familiarize ourselves with the tools available to sense the distances from a medium. The ultimate goal of the project is to develop a smart sewer monitor that can remotely monitor sewer water levels. I will be examining the use and programability of a Garmin LiDAR sensor with the ultimate goal being to design a circuit that could put one of these on the underside of a manhole cover.

2.2 Materials and Methods

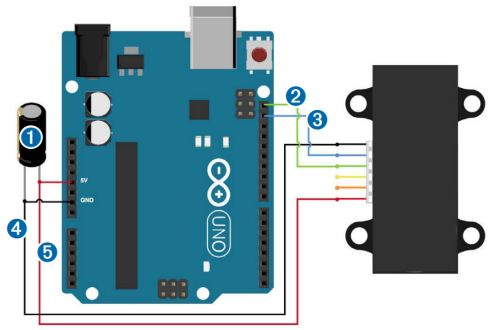
2.2.1 The Hardware

This project involves the use of an Arduino Uno, two 4.7kΩ resistors for the I2C communication protocol, a 1000μF smoothing capacitor and the Garmin LIDAR-Lite Optical Distance Sensor - V3 (the datasheet for which can be found here)

2.2.2 The Circuit

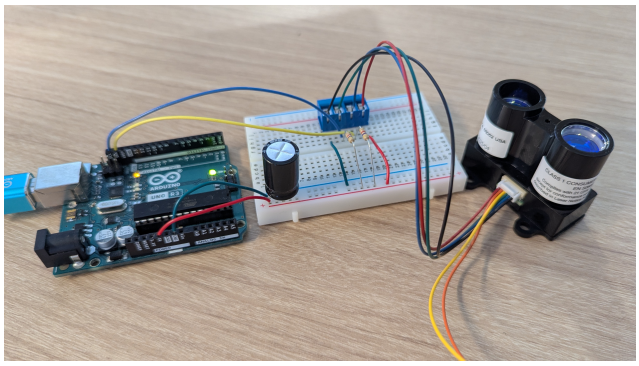
Figure 2.1a shows the circuit that can be found in the datasheet, but the working circuit I built (seen in Figure 2.1b) looked a bit different.

Standard Arduino I2C Wiring



Item	Description	Notes
1	680µF electrolytic capacitor	You must observe the correct polarity when installing the capacitor.
2	I2C SCA connection	Green wire
3	I2C SDA connection	Blue wire
4	Power ground (-) connection	Black wire
5	5 Vdc power (+) connection	Red wire The sensor operates at 4.75 through 5.5 Vdc, with a max. of 6 Vdc.

(a) The circuit illustrated in the Garmin LiDAR's datasheet. Note the 680μF capacitor and the *lack* of pull up resistors for the SDA and SCL pins.



(b) The working circuit I built to test the sensor.

Figure 2.1: Theoretical vs experimental circuit design.

2.2.3 The Code

I worked on two versions of the code that interfaces with the sensor. The version 1 (V1) is seen in the first code snippet below. The V1 code simply interacts with the sensor, gets a distance reading, and prints it to the serial monitor. I also wanted to expand my knowledge of Arduino libraries, so I also made a V2 of the code that calculates the average and population standard deviation of 100 distance readings.

```
/* ===== Version 1 of the code ===== */
#include <Wire.h>
const int deviceAddress = 0x62;
//Default I2C address
bool conversionComplete = false;
//Boolean trigger
byte conversionStatus;
//A trigger for waiting for measurements
byte hi;
//The first byte in distance measurement
byte lo;
//The second byte in the distance measurement
int distance;
//The distance value after combining hi and lo

int measureDistance(){
//Define distance measurement function
```

```
  Wire.beginTransmission(deviceAddress);
  //Open communications with sensor
  Wire.write(0x00);
  //Access register 0x00
  Wire.write(0x04);
  //Write 0x04 to 0x00 register
  Wire.endTransmission();
  //Close transmission to register 0x00

  while(!conversionComplete){
  //While the sensor conversion isn't complete
    Wire.beginTransmission(deviceAddress);
 //Open communications with sensor
    Wire.write(0x01);
 //Access register 0x01
    Wire.endTransmission();
 //Close transmission to register 0x01
    Wire.requestFrom(deviceAddress, 1);
 //Request 1 byte from the sensor
    conversionStatus = (Wire.read() & 0x01);
 //Boolean test to see if the sensor is readable
    if(!conversionStatus){
 //If the sensor is read
      conversionComplete = !conversionComplete;
   //Invert the logic of the bool trigger
    }
  }

  Wire.beginTransmission(deviceAddress);
  //Open communications with sensor
  Wire.write(0x8F);
  //Access the 0x8F register
  Wire.endTransmission();
  //Close transmission to register 0x8F
  Wire.requestFrom(deviceAddress, 2);
  //Request 2 bytes from the sensor
  hi = Wire.read();
  //Read the first byte
  lo = Wire.read();
  //Read the second byte
  Wire.endTransmission();
  //Close transmission to sensor
  return hi*256 + lo;
  //Convert the two bytes to an integer of distance in cm
}

void setup() {
```

```
  Serial.begin(9600);
  //Start serial monitor, 9600 baud
  Serial.println("Starting measurements...");
  //Print starting statement
  Wire.begin();
  //Initialize the I2C communications
}

void loop() {
  distance = measureDistance();
  //Measure the distance, save integer value (cm)
  Serial.print(distance);
  //Print out the distance
  Serial.println(" cm");
  delay(10);
  //Wait a tenth of a second
}
```

```
/* ===== Version 2 of the code ===== */
#include <Wire.h>
#include <Statistic.h>
const int deviceAddress = 0x62;
//Default I2C address
bool conversionComplete = false;
//Boolean trigger
byte conversionStatus;
//A trigger for waiting for measurements
byte hi;
//The first byte in distance measurement
byte lo;
//The second byte in the distance measurement
statistic::Statistic<float, uint32_t, true> distance;
//Define distance object

int measureDistance(){
//Define distance measurement function
  distance.clear();
  for(int ii = 0; ii < 100; ii++){
    Wire.beginTransmission(deviceAddress);
 //Open communications with sensor
    Wire.write(0x00);
 //Access register 0x00
    Wire.write(0x04);
 //Write 0x04 to 0x00 register
    Wire.endTransmission();
 //Close transmission to register 0x00

    while(!conversionComplete){
 //While the sensor conversion isn't complete - WAIT
      Wire.beginTransmission(deviceAddress);
   //Open communications with sensor
      Wire.write(0x01);
   //Access register 0x01
      Wire.endTransmission();
   //Close transmission to register 0x01
      Wire.requestFrom(deviceAddress, 1);
   //Request 1 byte from the sensor
      conversionStatus = (Wire.read() & 0x01);
   //Boolean test to see if the sensor is readable
      if(!conversionStatus){
   //If the sensor is read
        conversionComplete = !conversionComplete;
  //Invert the logic of the bool trigger
      }
    }
```

```
    Wire.beginTransmission(deviceAddress);
 //Open communications with sensor
    Wire.write(0x8F);
 //Access the 0x8F register
    Wire.endTransmission();
 //Close transmission to register 0x8F
    Wire.requestFrom(deviceAddress, 2);
 //Request 2 bytes from the sensor
    hi = Wire.read();
 //Read the first byte
    lo = Wire.read();
 //Read the second byte
    Wire.endTransmission();
 //Close transmission to sensor
    distance.add(hi*256 + lo);
 //Convert the two bytes to an integer of distance in cm
    delay(5);
 //Small delay to add numbers together (errors w/o)
  }
}

void setup() {
  Serial.begin(9600);
  //Start serial monitor, 9600 baud
  Serial.println("Starting measurements...");
  //Print starting statement
  Wire.begin();
  //Initialize the I2C communications
}

void loop() {
  measureDistance();
  //Measure the distance, save integer value (cm)
  Serial.print(distance.average(), 2);
  //Print out the average of 100 distances
  Serial.print(" +/- ");
  //Print out plus minus for visualization
  Serial.print(distance.pop_stdev(), 2);
  //Print out the standard deviations of 100 distances
  Serial.println(F(" cm"));
  //Include units
```

2.3 Results

Seen in Figure 2.2 below is a visualization of data collected by the different versions of code interfacing with the Garmin LiDAR sensor. Noteably, the V1 code was taken the day following the collection of the data seen in the V2 output graphs and shows a mean distance reading of approximately 3.5 cm lower than those of the V2. The Garmin was not moved or jostled between when the V2 and V1 measurements was performed.

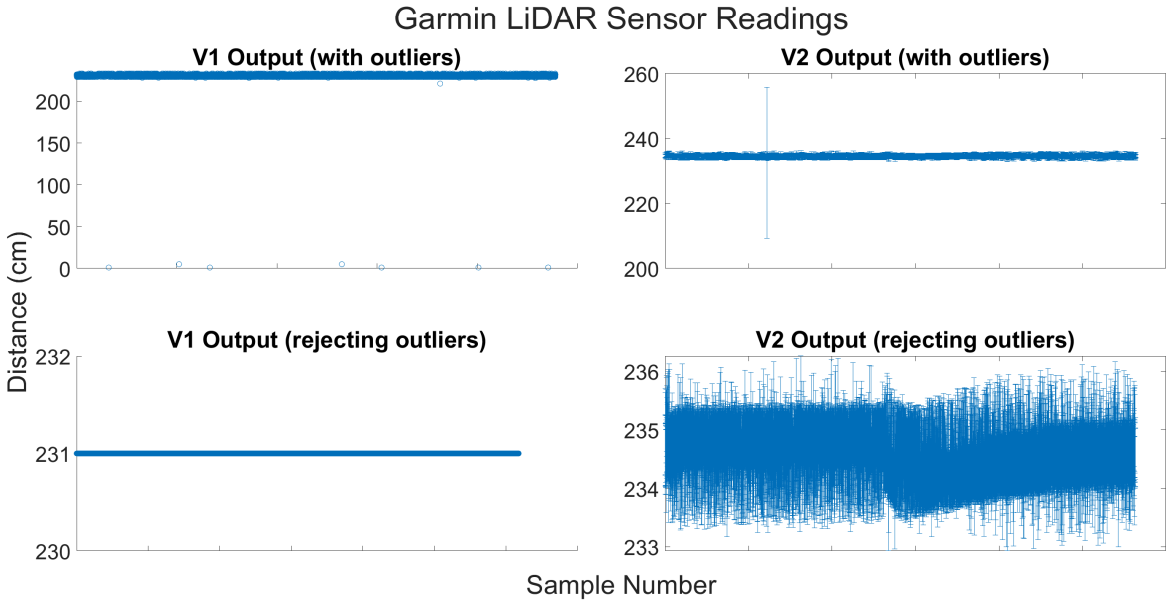


Figure 2.2: Graphs obtained from the V1 and V2 versions of the code I used to interface with the LiDAR sensor. The top left plot shows the V1 output as raw distance readings. The bottom left plot shows the findings after rejection of outliers. The top right plot shows the output of the V2 code with all data points shown. The bottom right plot shows the output of the V2 code but after post-hoc outlier detection and rejection was performed. Each plot's y-axis is given as a measure of distance in centimeters and each plot's x-axis is the sample number which can be interpreted as a time axis.

2.4 Discussion and Conclusion

Overall the Garmin LiDAR works well. The code also interfaces with it well and is able to do a passable job of data analysis. The object oriented nature of the statistic library called in the V2 code currently is beyond my comprehension, but I'll work on that. The only concerns I have with the sensor is the occasional outlier. This is of biggest concern in Figure 2.2 where the V1 of the code shows a couple of very low measurements which need to be compensated for in some way.

The statistic library doesn't seem to be too heavy on the arduino and the readings appear reasonable. I can't quite read the code and how it works, but I believe it's working as I intend it to. The question of how best to process this data and how best to implement its structure needs to come from end-goal design parameters. If I have a storm surge event that causes water levels to rise in the manhole, how often do I want to sample the level for it to be useful? Should the sample rate by minute-by-minute? Should I have a window of time (i.e. 10 minutes) where I wake, take measurements, and sleep? What is the sampling

frequency and how many measurements should make up a given averaged measurement? These are all questions to discuss with a wider group - specifically with those who know more about transient water events and long-term goals for the project.

Chapter 3

11/27/2025 Exploration of the Metal Oxide Substrate (MOS) Sensor

Discussion regarding the metal oxide substrate (MOS) sensors is had with its possible uses and calibrations as well as preliminary code and data collected from that code are shown.

Keywords: metal oxide substrate (MOS) sensor, MQ-135, MQ-9, LM393 comparator, volatile organic compounds (VOCs)

3.1 Introduction

To determine the chemical makeup of the air that we breathe every day, I want to measure the amount of volatile organic compounds (VOCs) found in the steam being released from pipes all around the College of Science and Mathematics building. To this end, I want to employ something known as a metal oxide semiconductor (MOS) sensor that can measure the amount of various chemicals in the air. The MOS sensor operates on a straightforward principle: A substrate is heated to form an outer oxide layer - in this case, the substrate is tin (Sn) which forms SnO_2 when heated - and when reducing gasses - i.e. carbon monoxide (CO), carbon dioxide (CO_2), alcohol (C_2H_5OH), nitrous oxide (NO_2), ammonia (NH_3) - come nearby the substrate, the oxygen in the substrate-oxide is removed. This removal of oxygen molecules causes free electrons to move through the substrate and change the resistance of the sensor. This change in resistance is proportional to the amount of reducing gas present in the sample[1]. These sensors are quite sensitive to temperature and humidity[2] and have other limiting factors such as the lack of selectivity in the gas sampled and baseline drift over time. That being said, the low cost, sensitivity, low power consumption, and relatively long lifespan of 5-10 years (for reference see footnote 1) make it an appealing option for gas sampling especially when compared to mobile mass spectroscopy systems that can cost upwards of $80,000.

[1] For a brief overview of MOS sensors and the related chemistry check here.
[2] This makes sampling steam particularly challenging...

3.2 Materials and Methods

3.2.1 The MOS Sensor

The MOS sensor itself is a small roughly 2cm diameter × 2cm tall IC that contains a heating element, the Sn substrate, a grating that prevents debris accumulation, and six pins. The MOS sensors I'm working with come on a daughter board that, while containing several resistors and capacitors that presumably are required for operation along with a power LED indicator as seen in Figure 3.1a, which can output signal using a couple of different operations depending on what pin you elect to read from (see Figure 3.1b). The first operation is that it can serve as an alarm through its digital output. This mode uses a LM393 comparator to detect when a reference voltage exceeds a certain level. As described here the comparator returns LOW when the input voltage is below a certain threshold and returns HIGH when the input voltage exceeds that amount. In the case of these sensors, the reference voltage is set by a potentiometer.

(a) Overview of the MQ135 MOS sensor.

(b) Pinout diagram of the MQ 135 MOS sensor.

Figure 3.1: Pictures of the MQ 135 MOS sensor attached to its daughter board.

The second method of operation is the analog output. The analog output returns the raw analog signal coming from the MOS sensor itself. The meaning of this signal is significantly more challenging to resolve.

Regardless of the method of data acquisition, the MOS sensor must be treated with special care. Specifically the heating of the MOS sensor must be allowed to take place before measurements are allowed to take place. This typically takes a minute or two, but the datasheets say that for long-term storage of 6+ months the sensor must be "aged" for "no less than 168 hours" before accurate measurements can be made.

3.2.2 MOS Sensor End-User Calibration

The datasheet for the MQ135 shows a calibration curve as seen in Figure 3.2 below which appears to show how the resistance of the sensor (R_s) changes as a function of different *known* gas concentrations. If we link these calibration curves with the fact that nowhere in the datasheet does it provide a simple conversion factor, the sensor must be end-user calibrated for precise operation. One could simply read the graph shown if Figure 3.2 and derive the ratio of resistance to ppm measurements, but that would, at best, be a "hand grenade" kind of measurement as opposed to a "scalpel".

To my knowledge, there doesn't exist a single method for calibration of the MOS sensor. Before designing a template for calibration, we must keep a couple of factors in mind. The first factor in calibration

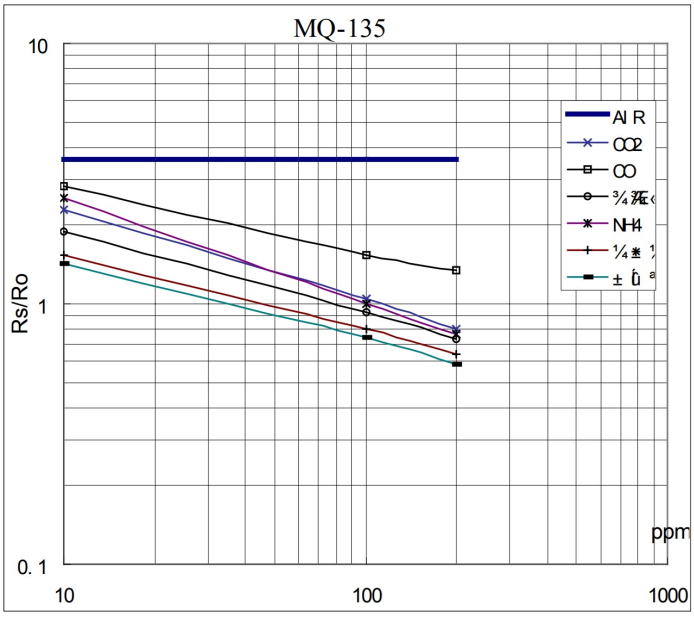


Fig.3 is shows the typical sensitivity characteristics of the MQ-135 for several gases.
in their: Temp: 20
Humidity: 65%
O_2 concentration 21%
RL=20kΩ
Ro: sensor resistance at 100ppm of NH_3 in the clean air.
Rs:sensor resistance at various concentrations of gases.

Figure 3.2: A calibration curve of the MOS sensor is shown along with its original caption to the right of the figure.

design is the temperature and humidity. There must be a consideration of temperature and humidity regulation in the test chamber. I think a temperature and humidity sensor such as the DHT11 (datasheet available here) could be employed to track the temperature and humidity during such calibrations. We must also devise a strategy for calculating the resistance of the sensor at the given temperature and humidity. According to this (seemingly) well thought out post on the Arduino project hub, we can calculate the resistance of the sensor by the following equation.

$$Rdg = \frac{1024 \cdot R_{\text{ref}}}{R_{\text{ref}} + R_{\text{s}}} \quad \longrightarrow R_{\text{s}} = \left(\frac{1024}{Rdg} - 1\right) R_{\text{ref}} \tag{3.1}$$

In the previous equation, Rdg is the reading coming from the sensor, R_{ref} is a reference resistance (assume STP, a given PPM, and "clean air" for this baseline reference resistance), and the 1024 - I believe - has to do with the Arduino's 10-bit ADC. It is noteworthy that there is no conversion from "ADC step" to voltage. Calibration strategy can be outlined by the following:

1. Build a MOS, temperature, & humidity sensor all-in-one such that the arduino can easily talk to all of the components at once.
2. While controlling for the temperature and humidity, introduce known gas concentrations into an airtight chamber measuring the PPM of the gas by known, calibrated tools.
3. Measure the sensor resistance as a function of the PPM, temperature, and humidity.
4. Repeat measurements for a range of gas concentrations.
5. Repeat measurements for a range of temperature and humidity conditions.

Maybe there are other ways to accomplish this that don't involve thousands of dollars of high-speed lab equipment, but that's the best idea I have for now. Regardless of precision calibration the sensors will still produce a voltage and can be used to track relative increases in gas concentrations.

3.2.3 The Circuit

The current circuit I am using is seen in Figure 3.3. I am borrowing the circuit from another professor, so it has multiple MOS sensors of different series on it. The actual connections required for the MOS sensor to function are seen in Figure 3.1b with 5V and GND supplied by the Arduino and the analog out being connected to one of the analog pins on the Arduino as well.

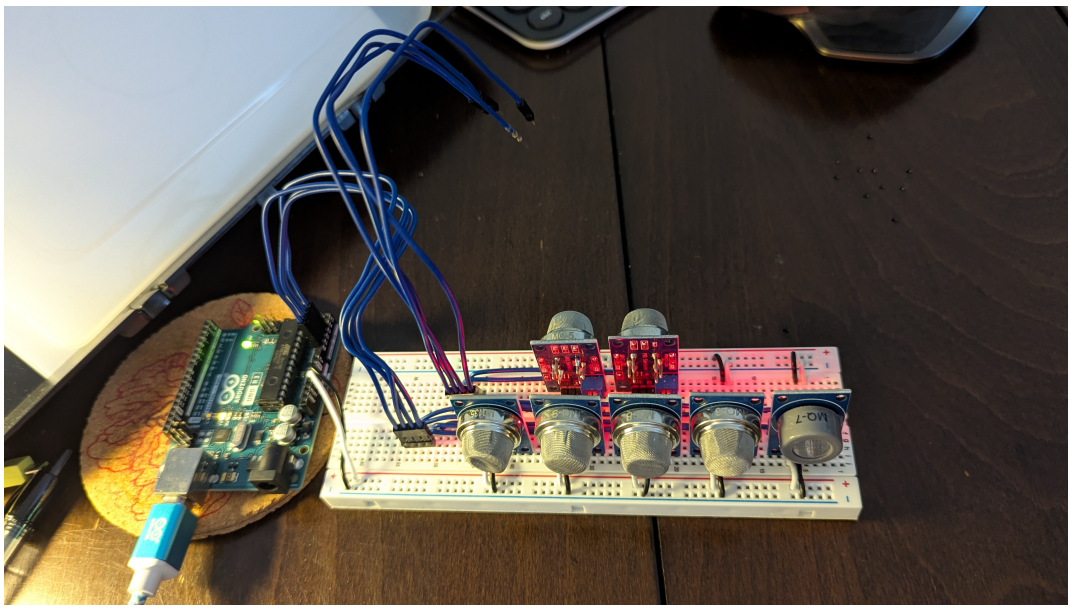


Figure 3.3: The actual experimental setup of my initial MOS sensor array. There are 7 different series of MOS sensors in total on this board wired for power and ground delivery with each being brought over to the Arduino via blue wires.

3.2.4 The Code

Coding the Arduino to interface with the MOS sensor is as difficult as interfacing with any analog sensor. Set up the analog pin, read it, and conversion can be done to measure the voltage coming from the pin. The code for this is seen below.

```
int rdg;                            //Define the reading variable
float voltage;                      //Define a float voltage reading
const int analogPin = A1;           //Define sensor pin

void setup() {
  pinMode(analogPin, INPUT);        //Set analogPin to INPUT
  Serial.begin(9600);               //Start serial monitor
}

void loop() {
  rdg = analogRead(analogPin);      //Read from the sensor
  voltage = rdg * (5000.0/1023.0);  //Convert the raw reading to mV
```

```
  Serial.print(voltage);            //Print the voltage
  Serial.println(" mV");            //Print out units
  delay(250);                       //Delay 250 ms
}
```

3.2.5 Initial experimental design

Using the previously described code, the Arduino interfaced with the MQ-9 MOS sensor which is primarily designed for the detection of carbon monoxide (CO) and other flammable gasses. Once the sensor was allowed enough time to heat, a baseline voltage reading was obtained. Once this reading reached a consistent minimum value, I got close by the sensor and breathed out onto the sensor. This small-scale test would allow the study of the response curve of the sensor. I breathed on the sensor for approximately 4.5 seconds and allowed the sensor to reobtain its consistent minimum value before concluding data collection.

3.3 Results

Results of the preliminary experiment are seen in Figure 3.4 below. Figure 3.4 clearly shows the peak reading of 391.01 mV and the time it took to fully recover to the consistent minimum value of 268.82 mV (87.75 s).

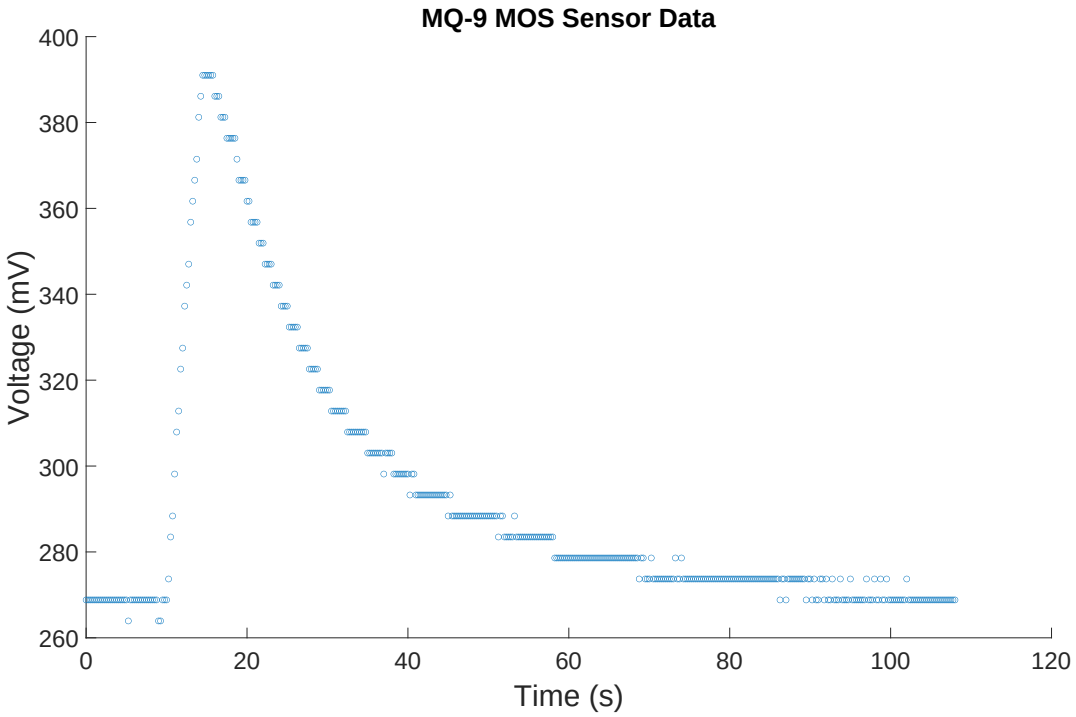


Figure 3.4: The response curve of the MQ-9 MOS sensor is shown. In the figure, the 4.5 seconds of breath are seen as a large peak upwards and the recovery time is seen as the rightward tail of the peak.

3.4 Discussion and Conclusion

The MOS sensor is a bit more delicate than I initially thought it would be. Given the challenges in calibration and its sensitivity to temperature and humidity, I don't know if this is a great sensor for analyzing the chemical composition of the steam. Typical applications of these kinds of sensors also focus on dynamic measurements from moving sources instead of what was done here - a static measurement in a room without much air circulation. Voss et al., 2022 showed the ability of the MOS sensors to be used in the detection of liver cirrosis, but they also mention that the possible drift and "Time-consuming recalibration" process make these sensors more difficult to work with. As a note, the aforementioned paper also reports sensor resistances of $\sim 400\text{k}\Omega$. If the goal of using the MOS sensor is to get a readout of x PPM of y substance, that will require a lot of work once you receive the sensor. That goal might be unachievable given the lack of selectibility of the sensor.

When finishing my research into the MOS sensor, I found there are alternative sensors that exist that *may* well be the answer to the problem I'm looking to solve. The Nondispersive infrared (NDIR) sensor provides gas sensing with broadband infrared light. Some examples of these sensors are the T6793 (datasheet) which is an UART/I2C capable device that can be used for CO_2 detection and the APC1001U (datasheet) - another UART/I2C device with 3.3V communication interface - which can be used to detect, among other things, air quality index, temperature, humidity, and "equivalent CO_2 values". These sensors may be a better alternative to MOS sensors if they prove to delicate for gas analysis.

While there are major shortcomings of the MOS sensor, its main point of strength is its sensitivity. The results seen in Figure 3.4 were quite surprising in that a "quick and dirty" test of my breath was able to illicit a strong signal response in the sensor. Detecting gas concentration *changes* appears to be a very strong point of the sensors and might still make them useful.

Index

Arduino Library - Statistic, 8

chemical sensing, 2

Garmin LIDAR-Lite Optical Distance Sensor - V3, 8

light detection and ranging (LiDAR), 8
LM35 temperature sensor, 2
LM393 comparator, 16

metal oxide substrate (MOS) sensor, 16
MQ-135, 16
MQ-9, 16

sensors, 2, 8

TC74 temperature sensor, 2

volatile organic compounds (VOCs), 16

Bibliography

Voss, A., Schroeder, R., Schulz, S., Haueisen, J., Vogler, S., Horn, P., Stallmach, A., & Reuken, P. (2022). Detection of liver dysfunction using a wearable electronic nose system based on semiconductor metal oxide sensors. *Biosensors*, *12*(2). https://doi.org/10.3390/bios12020070