

MRE 320: SENSORS AND ACTUATORS

GROUP PROJECT MILESTONE 1: SENSOR(s) MEASUREMENT(s) PROPOSAL

**BY: LUCAS SCHEIBAL, LIN SHEN,
YIYANG CHEN, CHENG CHEN AND
IRETOMIWA IDOWU (Group 3)**

PROFESSOR: DR. MINGSHAO ZHANG

September 24, 2023

INTRODUCTION

This project marked the beginning of our very first introduction to all things sensors and groupwork. As we have learnt in class, sensors make up a very crucial part of the electrical world which is why as a mechatronics engineer, it is important we get familiar with them before anything else.

In order to make sure we covered more ground in our sensor study, we were divided into groups and our group was tasked with measuring four sensors which are: The MPU6050 Module, The DHT-11 Module, The Ultrasonic Ranging Module and The Water Level Detection Sensor Module.

In the rest of the proposal is the summary of our findings from our sensor study, wiring diagrams of the sensors, the Arduino code used to operate the sensors of course, a picture of the sensor readings and finally the proposal of our testing plans of the sensors' measurements.

1. The MPU6050 Module

The first sensor we were assigned was the MPU6050 module/sensor. The MPU6050 is a multi-axis accelerometer and gyroscope sensor. It is commonly known as the accelerometer. It features a combines 3-axis gyroscope, 3-axis accelerometer, a Digital Motion Processor (DMP) and a temperature sensor. In our study, we found that it is the world's first and only 6-axis motion sensor that also doubles down as a temperature sensor.

Its primary purpose is to measure the two types of acceleration although it can also measure temperature. The accelerometer (whose working principle we would see below) measure linear acceleration while the gyroscope measures angular acceleration. This sensor is embedded in our everyday lives as, although we might not interact with it physically, is what makes gadgets (smartphones, tablets, wearable devices etc.) have a lot of extra features that makes living easier!

Finding the specs of the sensor above is very paramount to the very purpose of this project! Below is the datasheet containing the specification of the sensor.

- **Features, Characteristics and Specifications of the MPU6050 Sensor**

| Specification | Measurement |
|----------------------------------|---|
| Voltage Supply | 3, 3-5 V (DC) |
| Logic Level | 3.3V |
| Power consumption (Current draw) | 3.9mA |
| Dimensions (excluding pins) | 21.2mm (0.84") length x 16.4mm (0.65") width x 3.3mm (0.13") height |
| Weight | 2.1g (0.07oz) |
| Accelerometer ODR range | 8kHz to 1.25Hz |
| Accelerometer Range | ±2g, ±4g, ±8g, ±16g |
| Accelerometer sensitivity | at ±2g: 16384 LSB/g at ±4g: 8192 LSB/g at ±8g: 4096 LSB/g |

| | |
|--|---|
| | at $\pm 16g$: 2048LSB/g |
| Gyroscope ODR range | 8kHz to 1.25Hz |
| Gyroscope range | $\pm 250, \pm 500, \pm 1000, \pm 2000$ dps |
| Gyroscope sensitivity | at ± 250 dps: 131 LSB/dps at ± 500 dps: 65.5 LSB/dps at ± 1000 dps: 32.8 LSB/dps at ± 2000 dps: 16.4 LSB/dps |
| Interface | I ² C |
| Degrees of freedom | 6x |
| Temperature sensor measuring/operating range | -40 to 85 °C |
| Temperature sensor sensitivity | 340 LSB/° |
| Temperature sensor accuracy | $\pm 3^\circ\text{C}$ |
| Built-in chip | 16-bit AD converter |
| Shock Tolerance | 10,000g |

Table containing specifications of MPU6050 Module

• Working Principle of the MPU6050 Module

- How does the MPU6050 do what it does?

As mentioned before, the MPU6050 has a built-in accelerometer that measures linear acceleration. This is done by utilizing electrodes and a proof mass that are interlaced in a comb-like structure. The proof mass is free to move in one linear direction and will change distance between its “sensing fingers” and the electrodes. This then creates a differential capacitor between the two electrodes on either side of the proof masses sensing fingers. The core of this is that the capacitance changes as plates are closer or farther away. Finally, the accelerometer uses tiny systems to see how close to plates are to each other, and then converts this capacitance reading to an analog signal which is in turn displayed to the user. The gyroscope works in a similar way.

The gyroscope measures angular acceleration and it does this with a continuously oscillating mass and the known physics of the Coriolis effect. Capacitive plates surrounding two masses are able to move the masses to create oscillation. They are also the key when determining rotation. This is because the capacitance variation will increase with movement of the masses from their expected route. This is how the Coriolis effect is measured. These variations are converted to an analog voltage which allows it to be read electronically!

• Connection and Test Set-Up/ Wiring Diagram

In order to test the sensor, we connected the sensor to an Arduino Mega 2560 Board, then connected the whole set up to our laptop which contained the Arduino IDE we used to code instructions to test the sensor.

As for the setup, following the wiring diagram, we made note of the eight pins and their names and made sure we matched them on Arduino Mega Board (This is what helps us connect the sensor to the IDE and thus the code used to make it run). From previous knowledge, we know the first pin called Vcc is the

power supply of the sensor and it is connected to the 5v power source on the board and the second pin called GND which closes the circuit is connected to GND on the board which we did accordingly! (We could see this across all the sensors we were assigned regardless of the number of pins). The only other two pins needed for the test were SDA and SCL which are used to receive and send data respectively were then connected to their SDA and SCL port counterpart which is located on the communication section of the Arduino board! We know our connection was successful due to two lights on the Arduino coming on!

Below is an image our connected hardware.

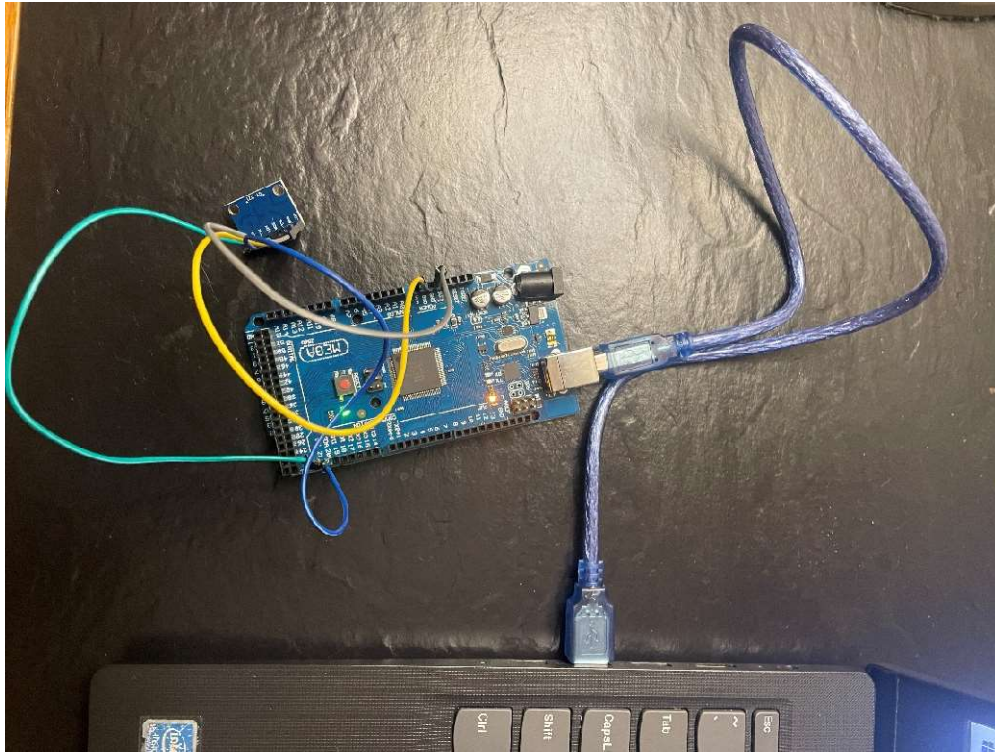


Figure 1: The MPU6050 Hardware set-up

• Arduino Code

Just like any other device/program, our connection needs instructions to work! The code is divided into two main parts; the first part declares the variables we want to measure while setting up the address of the sensor and the second part reads those variables and displays them on the screen! The code below is what was used to test the sensor and attached below is the output displayed.

```
#define ACCELE_RANGE 4
#define GYROSC_RANGE 500

#include<Wire.h>
const int MPU_addr = 0x68; // I2C address of the MPU-6050
float AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
```

```

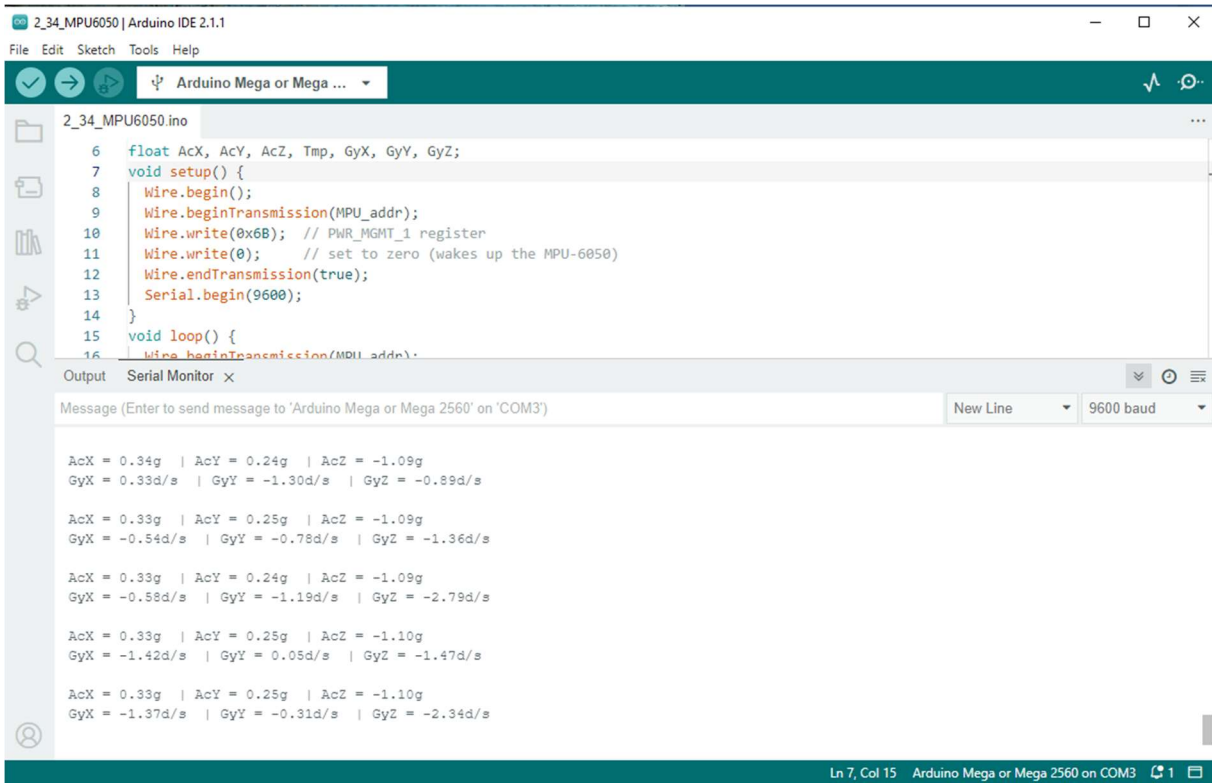
void setup() {
    Wire.begin();
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x6B); // PWR_MGMT_1 register
    Wire.write(0);    // set to zero (wakes up the MPU-6050)
    Wire.endTransmission(true);
    Serial.begin(9600);
}

void loop() {
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_addr, 14, true); // request a total of 14 registers
    AcX = Wire.read() << 8 | Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C
    (ACCEL_XOUT_L)
    AcY = Wire.read() << 8 | Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E
    (ACCEL_YOUT_L)
    AcZ = Wire.read() << 8 | Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40
    (ACCEL_ZOUT_L)
    Tmp = Wire.read() << 8 | Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
    GyX = Wire.read() << 8 | Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44
    (GYRO_XOUT_L)
    GyY = Wire.read() << 8 | Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46
    (GYRO_YOUT_L)
    GyZ = Wire.read() << 8 | Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48
    (GYRO_ZOUT_L)
    Serial.print(" AcX = "); Serial.print(AcX / 65536 * ACCELE_RANGE-0.01);
    Serial.print("g ");
    Serial.print(" | AcY = "); Serial.print(AcY / 65536 * ACCELE_RANGE);
    Serial.print("g ");
    Serial.print(" | AcZ = "); Serial.print(AcZ / 65536 * ACCELE_RANGE+0.02);
    Serial.println("g ");
    // Serial.print(" | Tmp = "); Serial.println(Tmp/340.00+36.53); //equation for
    temperature in degrees C from datasheet
    Serial.print(" GyX = "); Serial.print(GyX / 65536 * GYROSC_RANGE+1.7);
    Serial.print("d/s ");
    Serial.print(" | GyY = "); Serial.print(GyY / 65536 * GYROSC_RANGE-1.7);
    Serial.print("d/s ");
    Serial.print(" | GyZ = "); Serial.print(GyZ / 65536 * GYROSC_RANGE+0.25);
    Serial.println("d/s \n");
    delay(500);
}

```

• Sensor Readings – Output

Below is an image containing the output of the sensor code which is the readings of the sensor.



The screenshot shows the Arduino IDE interface. The top window displays the code for '2_34_MPU6050.ino'. The code includes variable declarations for acceleration (AcX, AcY, AcZ), angular velocity (GyX, GyY, GyZ), and temperature (Tmp). The setup function initializes the I2C bus and configures the MPU6050 sensor. The loop function reads sensor data and prints it to the serial monitor.

```
6 float AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
7 void setup() {
8   Wire.begin();
9   Wire.beginTransmission(MPU_addr);
10  Wire.write(0x6B); // PWR_MGMT_1 register
11  Wire.write(0);    // set to zero (wakes up the MPU-6050)
12  Wire.endTransmission(true);
13  Serial.begin(9600);
14 }
15 void loop() {
16  Wire.beginTransmission(MPU_addr);
```

The bottom window shows the serial monitor output, displaying the sensor readings at 9600 baud. The output shows acceleration and angular velocity values in g and d/s respectively.

```
AcX = 0.34g | AcY = 0.24g | AcZ = -1.09g
GyX = 0.33d/s | GyY = -1.30d/s | GyZ = -0.89d/s

AcX = 0.33g | AcY = 0.25g | AcZ = -1.09g
GyX = -0.54d/s | GyY = -0.78d/s | GyZ = -1.36d/s

AcX = 0.33g | AcY = 0.24g | AcZ = -1.09g
GyX = -0.58d/s | GyY = -1.19d/s | GyZ = -2.79d/s

AcX = 0.33g | AcY = 0.25g | AcZ = -1.10g
GyX = -1.42d/s | GyY = 0.05d/s | GyZ = -1.47d/s

AcX = 0.33g | AcY = 0.25g | AcZ = -1.10g
GyX = -1.37d/s | GyY = -0.31d/s | GyZ = -2.34d/s
```

Figure 2: Output of the MPU6050 Arduino code

With these values and the testing as a whole, we were able to make informed decisions on our testing plans.

• Test Plans/Measurement Proposal

We plan to measure both linear and angular acceleration and then extract the characteristics of the sensor from our readings

- * To measure linear acceleration, we would like to use a treadmill, start it at one speed, then quickly ramp it up and measure the acceleration recorded with the expected acceleration associated with the treadmill's motor

- * We are hoping to take a similar route with angular acceleration, except with a drill. Initially we had the idea of using a wheel which we planned to access hopefully with Professor Zhang's involvement in the art school but we realized that would still be a linear motion and thus switched to using a drill!

Below are the characteristics we hope/wish to try measure alongside how we plan to extract them from our approach.

SENSITIVITY:

To measure sensitivity, we would use a set of different springs could be used to change the acceleration rate. A few different springs acceleration could be calculated using $a=kx/m$. Then, over a few different springs, the change in output could be compared to the change in input to determine sensitivity. On the other hand, a drill could be used to measure sensitivity of rotational motion. By pressing the trigger of the drill to different depths each time, different accelerations would occur but it would be necessary to reverse engineer the acceleration of the drill if we are to get any readings

RESOLUTION:

To determine resolution, we would use the same spring testing, but this time using smaller and smaller springs until the device can no longer sense the acceleration. A similar method to measuring sensitivity would occur for the rotation measurements but it would require very small rotational movements from a drill at minimum power.

REPEATABILITY:

Repeatability could be tested using a treadmill for a linear test. This would allow a change of velocity to subject the sensor to. This would allow the measurement of acceleration which we would then compare to the acceleration expected from the treadmill motor. We could use a similar testing method to compare the repeatability of the gyroscope measurements using a drill to allow for rotational motion.

RANGE:

For this, we would be measuring the range provided in the datasheet. Using the values given, we could calculate how much force we need to put in to get the acceleration (using $F = ma$) and since the sensor is light, it could be flicked to reach the min and max acceleration which we would then compare to the values we have at hand.

2. The DHT-11 Module

The second sensor we were assigned was the DHT-11 module. The DHT-11 is a temperature and humidity sensor. It features a resistive-type humidity measurement component, a NTC temperature measurement component and an 8-bit microcontroller. Its primary purpose is to measure temperature and humidity. This is another sensor that is embedded in our everyday lives as it helps us regulate and maintain temperature! We can see it in appliances like refrigerators, freezers, thermostats, ovens and so on.

As mentioned earlier, finding the specs of the sensor above is very paramount to the very purpose of this project! Below is the datasheet containing the specification of the sensor.

- **Features, Characteristics and Specifications of the DHT-11 Module**

| Parameters | Conditions | Minimum | Typical | Maximum |
|-------------------------|-------------------------|---------|------------|---------|
| Humidity | | | | |
| Resolution | | 1%RH | 1%RH | 1%RH |
| | | | 8 Bit | |
| Repeatability | | | ±1%RH | |
| Accuracy | 25°C | | ±4%RH | |
| | 0-50°C | | | ±5%RH |
| Interchangeability | Fully Interchangeable | | | |
| Measurement Range | 0°C | 30%RH | | 90%RH |
| | 25°C | 20%RH | | 90%RH |
| | 50°C | 20%RH | | 80%RH |
| Response Time (Seconds) | 1/e(63%)25°C , 1m/s Air | 6 S | 10 S | 15 S |
| Hysteresis | | | ±1%RH | |
| Long-Term Stability | Typical | | ±1%RH/year | |
| Temperature | | | | |
| Resolution | | 1°C | 1°C | 1°C |
| | | 8 Bit | 8 Bit | 8 Bit |
| Repeatability | | | ±1°C | |
| Accuracy | | ±1°C | | ±2°C |
| Measurement Range | | 0°C | | 50°C |
| Response Time (Seconds) | 1/e(63%) | 6 S | | 30 S |

| Item | Measurement Range | Humidity Accuracy | Temperature Accuracy | Resolution | Package |
|-------|---------------------|-------------------|----------------------|------------|------------------|
| DHT11 | 20-90%RH 0-50 °C | ±5%RH | ±2°C | 1 | 4 Pin Single Row |

Table containing specifications of the DHT-11 Module

• Working Principle of the DHT-11 Module

- How does the DHT-11 Module do what it does?

The DHT11 sensor consists of a capacitive humidity sensing element and a thermistor for sensing temperature. The humidity sensing capacitor has two electrodes with a moisture holding substrate as a dielectric between them. So as the humidity changes, the conductivity of the substrate changes or the resistance between these electrodes' changes. The IC then measures, processes this changed capacitance values and change them into digital form which is finally read and displayed by the microcontroller

On the other hand, for measuring temperature, the DHT-11 uses a Negative Temperature coefficient thermistor, which causes a decrease in its resistance value with increase in temperature. This resistance change is then measured and converted to temperature which is in turn outputted to the user.

• Connection and Test Set-Up/Wiring Diagram

In order to run the sensor, we connected the sensor to an Arduino Mega 2560 Board, then connected the whole set up to our laptop which contained the Arduino IDE we used to code instructions to test the sensor.

As for the setup, following the wiring diagram, we made note of the 3 pins and their names and made sure we matched them on Arduino Mega Board (This is what helps us connect the sensor to the IDE and thus the code used to make it run). From previous knowledge, we know the first pin called + is the power supply of the sensor and is always connected to the 5v power source on the board and the last pin called - which closes the circuit is connected to GND on the board which we did accordingly. Finally, the middle pin was then connected to any numbered port (we chose port 4) of our choice on the digital PWM!

Below is an image our connected hardware.

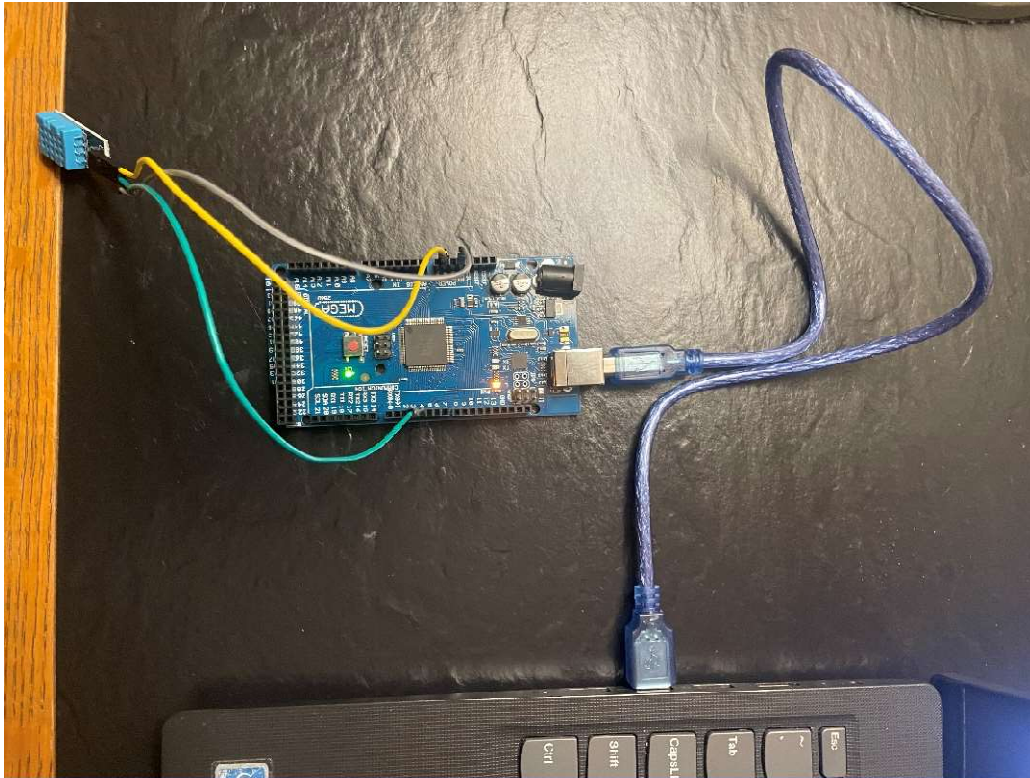


Figure 3: The DHT-11 Module Hardware set-up

• Arduino Code

Once again to run our sensor, we need instructions for the sensor to IDE to follow! The code we used for testing our sensor is divided into three distinct parts as seen below. The first part for setting up the sensor, the second for declaring our variables, reading the values and for error checking and the third is for displaying the values which we can see as the output. Below is the code and attached beyond that is the output displayed containing our values!

```
#include "DHT.h"

#define DHTPIN 4 // Set the pin connected to the DHT11 data pin
#define DHTTYPE DHT11 // DHT 11
//Setting up sensor
DHT dht(DHTPIN, DHTTYPE);
```

```

void setup() {
  Serial.begin(9600);
  Serial.println("DHT11 test!");
  dht.begin();
}

//error checking and declaring variables
void loop() {
  // Wait a few seconds between measurements.
  delay(2000);

  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (it's a very slow sensor)
  float humidity = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float temperatureCelcs = dht.readTemperature();
  float temperatureFah = ( temperatureCelcs * (9/5)) + 32;

  // Check if any reads failed and exit early (to try again).
  if (isnan(humidity) || isnan(temperatureCelcs)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print the humidity and temperature
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");
  Serial.print("Temperature(°C): ");
  Serial.print(temperatureCelcs);
  Serial.println(" *C");
  Serial.print(" %\t");
  Serial.print("Temperature(°C): ");
  Serial.print(temperatureFah);
  Serial.println(" *F");
}

```

• Sensor Readings – Output

Below is an image containing the output of the sensor code which is the readings of the sensor.

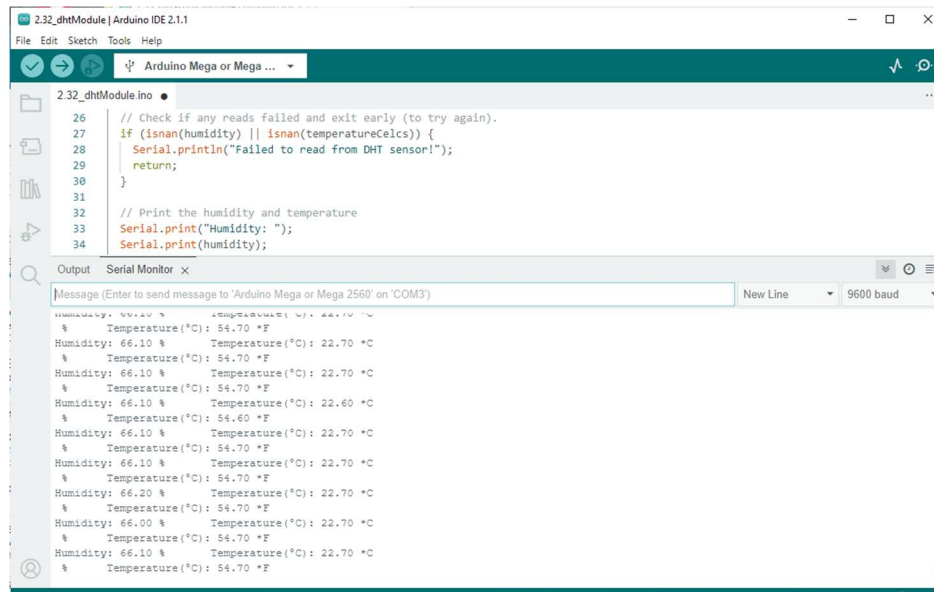
The image shows the Arduino IDE interface. The top window displays the code for '232_dhtModule.ino'. The code includes comments and C++ syntax for reading humidity and temperature from a DHT-11 sensor. The Serial Monitor window at the bottom shows the output of the code, displaying humidity and temperature readings in both Celsius and Fahrenheit. The output is as follows:
Humidity: 66.10 % Temperature(°C): 22.70 °C
% Temperature(°F): 54.70 °F
Humidity: 66.10 % Temperature(°C): 22.70 °C
% Temperature(°F): 54.70 °F
Humidity: 66.10 % Temperature(°C): 22.70 °C
% Temperature(°F): 54.70 °F
Humidity: 66.10 % Temperature(°C): 22.60 °C
% Temperature(°F): 54.60 °F
Humidity: 66.10 % Temperature(°C): 22.70 °C
% Temperature(°F): 54.70 °F
Humidity: 66.20 % Temperature(°C): 22.70 °C
% Temperature(°F): 54.70 °F
Humidity: 66.00 % Temperature(°C): 22.70 °C
% Temperature(°F): 54.70 °F
Humidity: 66.10 % Temperature(°C): 22.70 °C
% Temperature(°F): 54.70 °F

Figure 4: Output of the DHT-11 Module Arduino code

With these values and the testing as a whole, we were able to make informed decisions on our testing plans.

• Test Plans/M Measurement Proposal

In other to test our sensor, we came up with ways to obtain temperature data which we can then use to measure some of its characteristics. We plan to obtain an ice and heat box from our professor or since it is easier to get hot temperatures, go to the 3d printing lab in the school of engineering and set it to the temperature we want and if that falls through use a freezer to obtain cold temperatures and a stove (or a heat pack/pad) for hot temperatures keeping in mind the safety level for the sensor.

To get constant varying temperature, we plan to put the sensor in the freezer and close it up and for hot temperatures put the sensor close at a distance (whose temperature we measured) from the stove.

For varying temperature, we plan to move the sensor at intervals close to the stove and do the same for the freezer/fridge! Below are the characteristics we hope/wish to try measure alongside how we plan to extract them from our approach.

DRIFT:

We can adjust the temperature (both hot and cold) to a constant value and use a sensor to measure the temperature. This process will be maintained for a period of time, which can be three minutes or five minutes, and then observe the changes in the value (drift) measured by the sensor during this time.

STATIC ERROR:

We will premeasure a specified temperature using a better temperature measuring device starting from room temperature and then measure it using the sensor. The recorded temperature by the sensor will then be compared to the premeasured temperature. This will be repeated for a range of values, the graph of true temperature vs measured temperature will be plotted and a relationship established which will be then used to find the error in the sensor.

LINEARITY:

We can continuously change the input temperature (either hot or cold or both) and then get the output from the sensor. We can do this to obtain multiple readings. We can then plot the resulting data and see how best the readings/plot fit a linear line. We can also alongside test the sensitivity by obtaining a transfer function with the aid of a computing software from the plot, in which the slope of the function would be the sensitivity

SATURATION:

We can increase the temperature at regular intervals until the sensor stops responding, and of course if after that it stops responding we can decrease the temperature at smaller intervals so we get a more accurate value of when it stops responding

REPEATABILITY:

This will be tested by repeating the same measurement several times at a specified temperature. This will be done for different specified temperature; we can then compare the difference between values recorded. To ensure we get accurate results, we can test this in both temperature conditions: hot and cold

3. The Ultrasonic Ranging Module

The third sensor we were assigned was the Ultrasonic Ranging module (HS-SR04). The HS-SR04 is a proximity sensor. It features two ultrasonic transducers. One is transmitter which outputs ultrasonic sound pulses and the other is receiver which listens for reflected waves. Its primary purpose is to measure distance. This is another sensor that although we do not interact with physically is embedded in our everyday lives! We can see it used in sensing cars at drive-thru restaurants and at car-washes, used in medicine for sonography and surgery and so much more.

As mentioned earlier, finding the specs of the sensor above is very paramount to the very purpose of this project! Below is the datasheet containing the specification of the sensor.

• Features, Characteristics and Specifications of the Ultrasonic Ranging Module

| Specification | Measurement |
|---------------------------|--------------------------------|
| Input Voltage | 5v |
| Current Draw | 20mA (Max) |
| Operating Temperature | -15°C to 70°C |
| Sensing Angle | 30° Cone |
| Effective Angle | 15° Cone |
| Operating Frequency | 40kHz |
| Range | 2cm – 400cm (Practically 80cm) |
| Accuracy | ± 3mm |
| Resolution | 0.3cm |
| Dimensions | 45mm x 20mm x 15mm |
| Trigger input pulse width | 10µs |
| Sensitivity | -65dB min |

Table containing specifications of the ultrasonic Ranging Module

• Working Principle of the Ultrasonic Ranging Module

- How does the Ultrasonic Ranging Module do what it does?

The measurement principle is very simple, generally using the echo time method, that is, detecting the time of ultrasonic round trip to the measured distance, when the transmitter emits a short pulse, the timer starts; the timer stops immediately when the receiver receives a return pulse. The recorded time value at this time is $D = CT / 2$, where D is the distance between the ultrasonic sensor and the measurement object, C is the propagation speed of the sound wave in the medium ($C = 331.4 + t273 / 1m / s$, t is the temperature in Celsius), and T is the time interval between ultrasonic emission echoes.

• Connection and Test Set-Up/Wiring Diagram

Once again, in order to test the sensor, we connected the sensor to an Arduino Mega 2560 Borad, then connected the whole set up to our laptop which contained the Arduino IDE we used to code instructions to test the sensor.

As for the setup, following the wiring diagram, we made note of the 4 pins and their names and made sure we matched them on Arduino Mega Board (This is what helps us connect the sensor to the IDE and thus the code used to make it run). From previous knowledge, we know the first pin called Vcc is the power supply of the sensor is always connected to the 5v power source on the board and the last pin called GND which closes the circuit is always connected to GND port on the board which we did accordingly! (We could see this across all the sensors we were assigned regardless of the number of pins). The other two pins called the Trig and Echo pin which are the transmitter and receiver pin respectively also known as the output and input were then connected once again to any two numbered ports of our choice on the digital PWM (we picked port 7 for output and 6 for input). We know our connection was successful due to two lights on the Arduino coming on!

Below is an image our connected hardware.

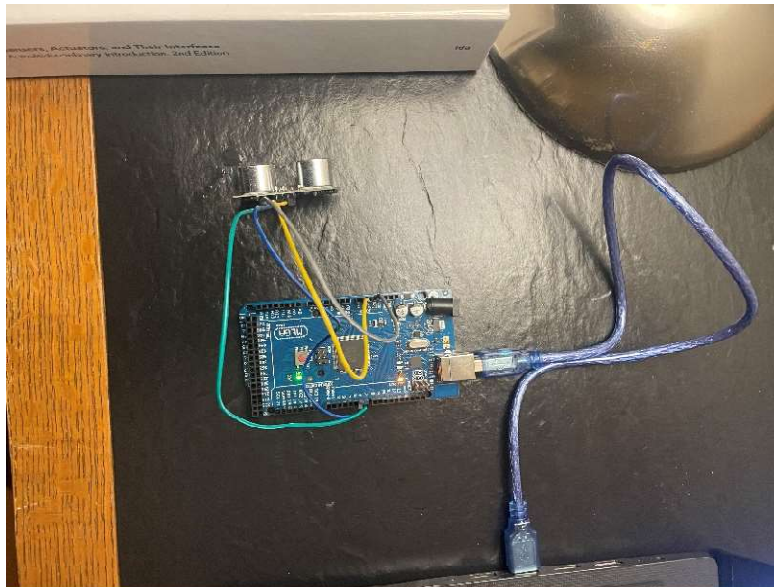


Figure 5: The Ultrasonic Ranging Module Hardware set-up

• Arduino Code

Once again to run our sensor, we need instructions for the sensor to IDE to follow! The code we used for testing our sensor does three main things as seen below. It first defines the pins and the subsequent ports used on the board and also declares the variables to be measured, it then declares the input and output pins, and finally loops through the stored value to calculate the distance.

Initially it sets the trigger to low to turn off any deviations or residue and then saves the time taken for the ultrasonic sound to go and return to the receiver when set to high. This time as we can see in the final part of the code is then multiplied with the speed of sound (since $\text{velocity} = \text{distance} / \text{time}$ therefore $\text{distance} = \text{velocity} * \text{time}$) and divided by 2 since the distance covered is both to and from the destination object. This distance value is then shown as an output on the screen.

Below is the code and attached beyond that is the output displayed containing our values!

```
const int echoPin = 6;
const int trigPin = 7;

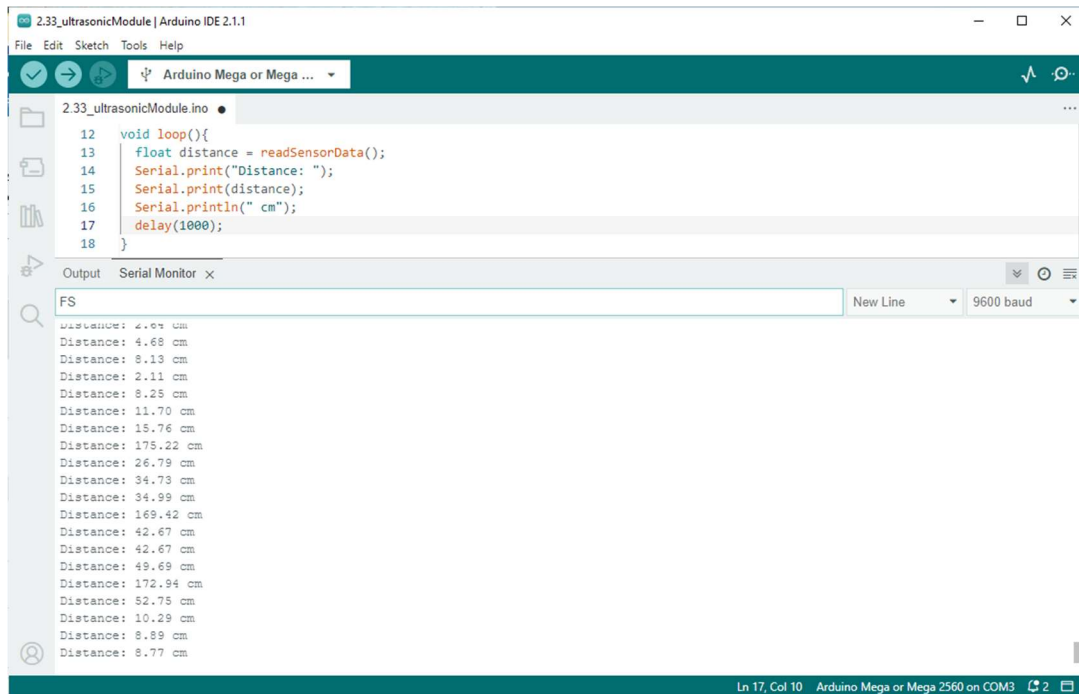
void setup(){
  Serial.begin(9600);
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);
  Serial.println("Ultrasonic sensor:");
}

void loop(){
  float distance = readSensorData();
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(1000);
}

float readSensorData(){
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  float distance = (pulseIn(echoPin, HIGH) * 0.034)/2; //Equivalent to
  (340m/s*1us)/2
  return distance;
}
```


• Sensor Readings – Output

Below is an image containing the output of the sensor code which is the readings of the sensor.



```
2.33_ultrasonicModule.ino
12 void loop(){
13   float distance = readSensorData();
14   Serial.print("Distance: ");
15   Serial.print(distance);
16   Serial.println(" cm");
17   delay(1000);
18 }
```

Output: FS

Distance: 4.61 cm
Distance: 4.68 cm
Distance: 8.13 cm
Distance: 2.11 cm
Distance: 8.25 cm
Distance: 11.70 cm
Distance: 15.76 cm
Distance: 175.22 cm
Distance: 26.79 cm
Distance: 34.73 cm
Distance: 34.99 cm
Distance: 169.42 cm
Distance: 42.67 cm
Distance: 42.67 cm
Distance: 49.69 cm
Distance: 172.94 cm
Distance: 52.75 cm
Distance: 10.29 cm
Distance: 8.89 cm
Distance: 8.77 cm

Figure 6: Output of the Ultrasonic Ranging Module Arduino code

With these values and the testing as a whole, we were able to make informed decisions on our testing plans.

• Test Plans/Measurement Proposal

In order to test this sensor, we narrowed down most of the idea to the best possible one which was to create an environment in which we could totally control the distance. This plan was feasible for this sensor compared to the other sensors as distance is much more easily controlled and can also be premeasured using an easily more accessible and accurate tool. The environment was just a simple setup of objects of different sizes and materials, a ruler or a tape rule and of course the sensor.

To get varying measurements or distance, we plan to set the object at a premeasured distance and then vary the proximity of the sensor to the object (and or vice versa) and to get constant measurements, we plan to set it also at a premeasured distance and then measure it under different conditions!

Below are the characteristics we hope/wish to try measure alongside how we plan to extract them from our approach.

RANGE:

We would be confirming the range provided in the datasheet above (Assuming that is the true value) and we would be doing this setting the sensor at a premeasured distance and with the ruler in between the sensor and the object, we would be adjusting the distance in increments of 10cm till we stop getting an output from the sensor confirming the max range. We would also do the same for the min range but this time in decrements of 5cm till it stops responding. Giving us the range of the sensor!

SENSITIVITY

This will be done in several ways, first the sensor will be tested in different temperature conditions to see if the same distance to an object can be detected in different conditions. Also, the surface of this object will be altered, and the sensor used to confirm if there is any change or if the measurements are the same. Or we can also alter the surface on which the object is placed and see if it responds to this change. Although at the end of this approach, we would not have any concrete value to define the sensitivity of the sensor, we would at least have an idea of how sensitive it is to changes! In order to get values that actually describes this property, we just need to obtain multiple readings at various distances, plot it with the help of a software and extract the linear transfer function in which the slope of the function would be the sensitivity

RESOLUTION

Just the range we would be setting, we would be confirming the resolution provided in the datasheet above (Assuming that is the true value) and we would be doing this by setting the sensor at a premeasured distance and with the ruler in between the sensor and the object, we would be adjusting the distance in increments of 0.1cm till we start getting a discernable output from the sensor! We would be trying this at different premeasured distances to get an accurate value!

HYSTERISIS

To measure this, we would measure and set a distance and then use the object to approach that distance from different directions, we would then record the subsequent outputs and then compare them to the premeasured/ set distance to observe the difference in the readings!

4. The Water Level Detection Sensor Module

This fourth sensor was one we were not assigned but instead tasked with picking on our own and the sensor we chose was The Water Level Detection Sensor Module (HW-038). The HW-038 is very self-descriptive as it does what it says it does: detect water level. Although its purpose goes beyond just detecting water levels as it is also used to monitor a sump pit, detect leaks and even rainfall! It features ten copper wires in which 5 are sense traces and the remaining five are power traces. These traces are disconnected in the absence of water, but when they are submerged in water, bridges are formed.

This is a sensor that we actually do not interact with in our daily lives except there is a need for it! But it is one we can easily incorporate if we ever do need it. As mentioned earlier, finding the specs of the sensor above is very paramount to the very purpose of this project! Below is the datasheet containing the specification of the sensor.

- **Features, Characteristics and Specifications of the Water Level Detection Sensor Module**

| Specification | Measurement |
|-----------------------|---|
| Operating Voltage | 3-5V |
| Operating Current | < 20mA |
| Sensor Type | Analog |
| Detection Area | 40mm * 16mm |
| Operating temperature | 10°C-30°C Humidity: 10% -90% non-condensing |
| Dimensions | 60mmx20mmx8mm |
| Weight | 3g |

Table containing specifications of the Water Level Detection Sensor Module

• Working Principle of the Water Level Detection Sensor Module

The working principle of the sensor is quite simple. The power and sensing traces form a variable resistor (similar to a potentiometer) whose resistance value changes based on their exposure to water. This resistance is inversely proportional to the depth of immersion of the sensor in the water: the deeper the sensor is immersed in the water, the better the conductivity and the lower the resistance. The shallower the sensor is immersed in water, the less conductive it is and the higher its resistance. The sensor produces an output voltage proportional to the resistance; by measuring this voltage, the water level can thus be determined, measured and thus displayed!

• Connection and Test Set-Up/Wiring Diagram

Again, in order to run the sensor, we connected the sensor to an Arduino Mega 2560 Board, then connected the whole set up to our laptop which contained the Arduino IDE we used to code instructions to test the sensor.

As for the setup, following the wiring diagram, we made note of the 3 pins and their names and made sure we matched them on Arduino Mega Board (This is what helps us connect the sensor to the IDE and thus the code used to make it run). From previous knowledge, we know the middle pin called + is the power supply of the sensor and is always connected to the 5v power source on the board but in this case due to the sensitivity of the sensor to power, we connected it to a numbered pin (7) so we could control power to go off when not in use and come on when in use! The last pin called - which closes the circuit was as usual connected to GND port on the board! The other pin called S was then connected any port of our choice on the analog section of the board (which in this case was A0). The S is an analog pin which is why we connected it to an analog port! We know our connection was successful due to Power LED on the sensor coming on!

Below is an image our connected hardware.

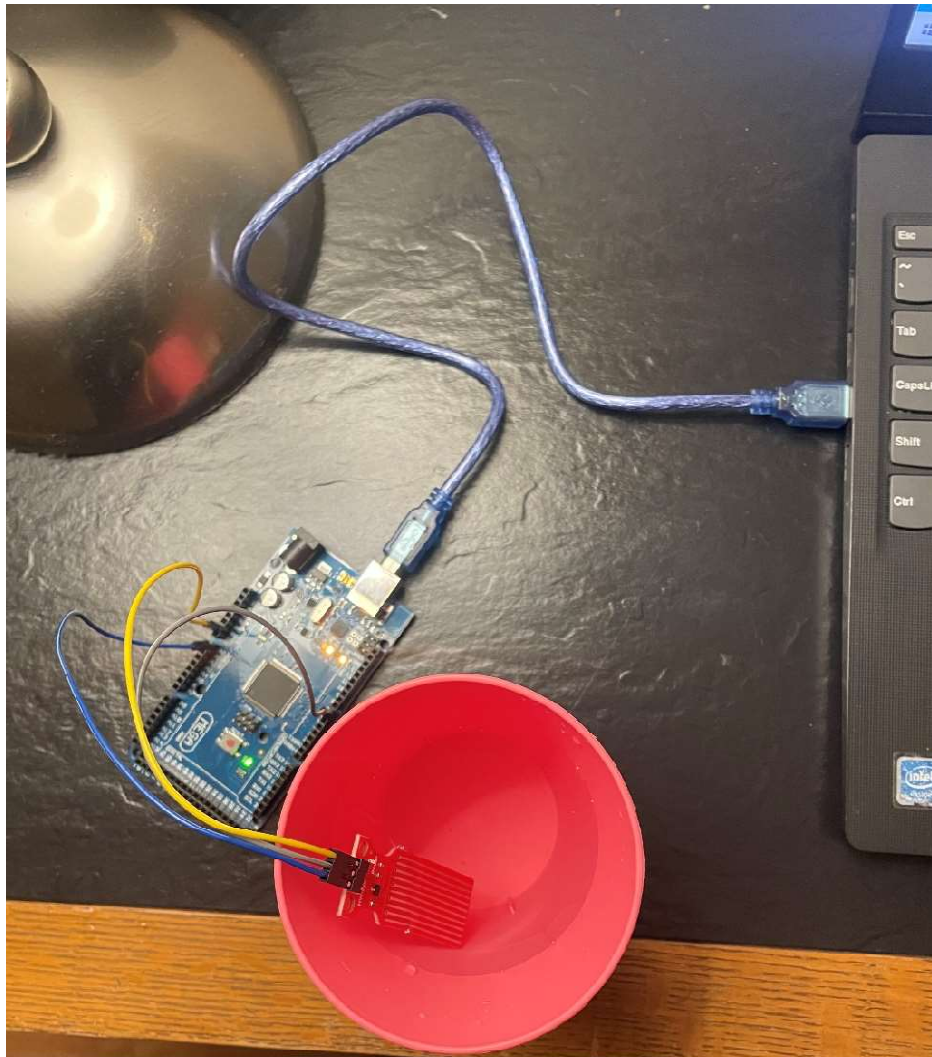


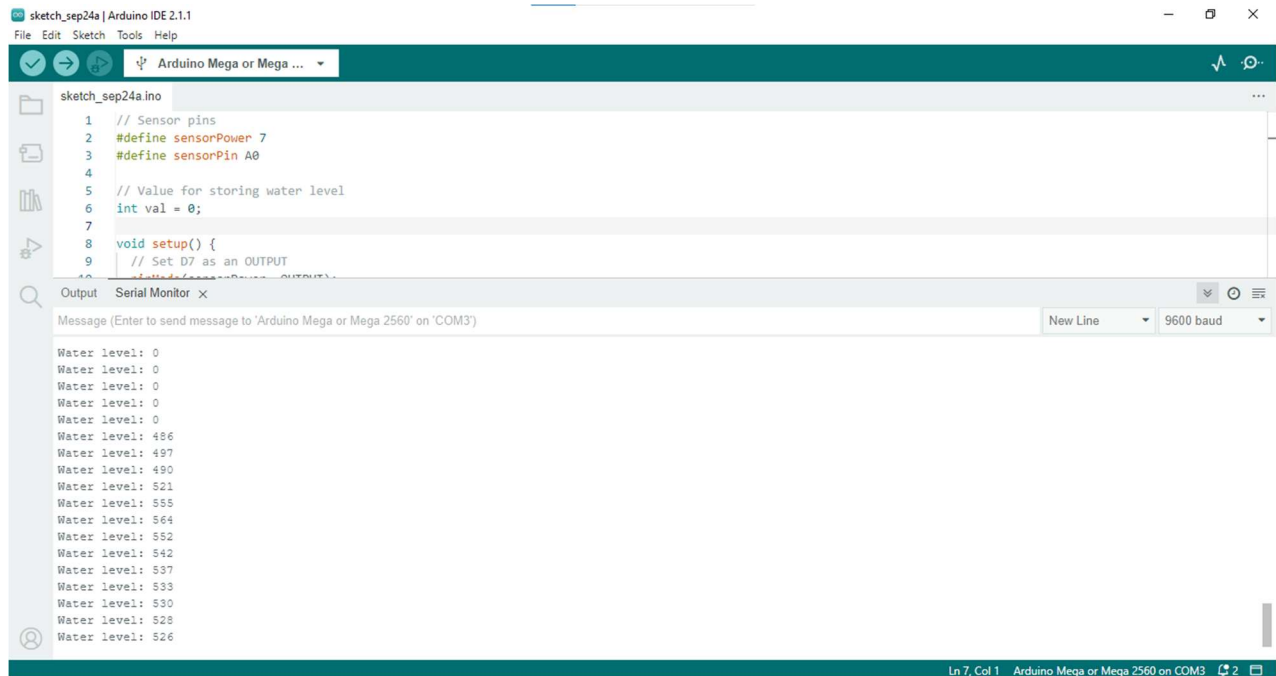
Figure 7: The Water Level Detection Sensor Module Hardware set-up

• Arduino Code

Once again to run our sensor, we need instructions for the sensor to IDE to follow! The code we used for testing our sensor is divided into two main parts as seen below. The first part which is the setup section establishes serial communication, configures the sensor's power connection to behave as an output and then sets it to low to keep the sensor off. The second part which is the loop section, turns on the sensor, reads the value and then displays it to the output! Below is the code and attached beyond that is the output displayed containing our values!

• Sensor Readings – Output

Below is an image containing the output of the sensor code which is the readings of the sensor.



The screenshot shows the Arduino IDE interface. The top menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar has icons for saving, running, and other functions. The main editor window displays the code for sketch_sep24a.ino. The code defines sensor pins, sets up a variable for water level, and prints the level in the setup function. The Serial Monitor at the bottom shows the output of the code, displaying 'Water level: 0' multiple times, followed by a series of values ranging from 486 to 528.

```
1 // Sensor pins
2 #define sensorPower 7
3 #define sensorPin A0
4
5 // Value for storing water level
6 int val = 0;
7
8 void setup() {
9   // Set D7 as an OUTPUT
10  pinMode(sensorPower, OUTPUT);
11
12  Serial.begin(9600);
13  while(1) {
14    val = analogRead(sensorPin);
15    Serial.println("Water level: " + String(val));
16    delay(1000);
17  }
18 }
```

Serial Monitor Output:

```
Water level: 0
Water level: 0
Water level: 0
Water level: 0
Water level: 0
Water level: 0
Water level: 486
Water level: 497
Water level: 490
Water level: 521
Water level: 555
Water level: 564
Water level: 552
Water level: 542
Water level: 537
Water level: 533
Water level: 530
Water level: 528
Water level: 526
```

Figure 8: Output of the Water Level Detection Sensor Module Arduino code

With these values and the testing as a whole, we were able to make informed decisions on our testing plans.

• Test Plans/Measurement Proposal

In order to test this final sensor, we came all came up with the same idea of filling a cup to a certain water level and then immersing it to get obtain readings

To get varying measurements, we plan to immerse it at different lengths of water (or even different fluids) and for constant measurements, we plan to keep it at a particular measured water level and then work around the conditions keeping in mind that we are working with metals that could corrode due to prolonged exposure to water and power.

Below are the characteristics we hope/wish to try measure alongside how we plan to extract them from our approach.

RANGE:

The range of the sensor refers to the maximum depth of water it can detect. To measure the range, we can submerge the sensor in water and gradually increase the depth until it no longer detects water. We can then compare this to the range specified by the data sheet above

SENSITIVITY:

The sensitivity of the sensor determines how well it can detect changes in water level. To measure sensitivity, we can slowly increase or decrease the water level while observing the sensor's output. We can then do this to obtain multiple readings, plot it, get the transfer function and get the sensitivity from the linear line of best fit

RESOLUTION:

The resolution of the sensor refers to its ability to detect small changes in water level. To measure resolution, we can make small incremental changes in water level and observe the corresponding changes in the sensor's output. This might not be feasible but we still plan to give it a try as we believe it is one of the easier properties to try to measure

PRECISION:

The precision of sensor is a deviation of a set of readings for a sensor of the same input. To measure this, we plan to obtain multiple readings at same water level (with the water level already premeasured) and with the help of a computing software calculate the standard deviation of the readings from each other and the premeasured (true) value!