Milestone #1 Sensor Measurement Project

MRE 320 Sensors and Actuators

Liam McGuire, Andrew Schalk, and Cheng Chen

February 11th, 2024

Tasked with familiarizing ourselves with and proposing testing plans for the three given sensors, this document lays out the result of our study and proposed plans.

MPU6050

The MPU6050 is a motion processing unit. With 6 degrees of freedom, it can measure linear acceleration and rotational velocity in and about all three dimensions. A triple-axis MEMS gyroscope measures rotational velocity and a triple-axis MEMS accelerometer measures linear acceleration. The accelerometer has four selectable ranges from ±2g to ±16g each corresponding

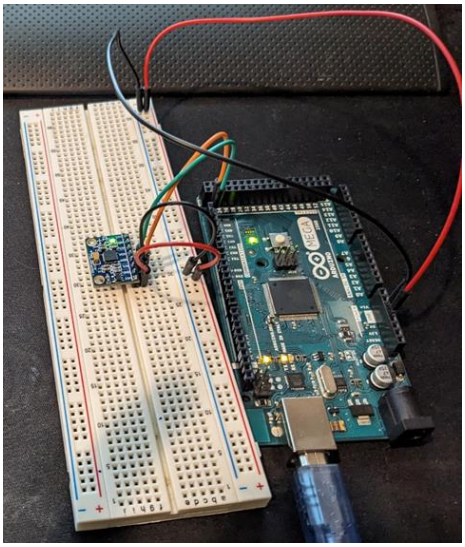


Figure 1

```
AcX = 0.00g   | AcY = 0.00g   | AcZ = 0.99g
GyX = 0.09d/s  | GyY = -0.28d/s  | GyZ = -0.25d/s

AcX = 0.01g   | AcY = 0.00g   | AcZ = 0.99g
GyX = 0.06d/s  | GyY = 0.01d/s  | GyZ = -0.20d/s

AcX = 0.00g   | AcY = 0.01g   | AcZ = 0.99g
GyX = 0.12d/s  | GyY = -0.11d/s  | GyZ = -0.08d/s

AcX = -0.00g  | AcY = 0.01g   | AcZ = 1.00g
GyX = 0.16d/s  | GyY = 0.15d/s  | GyZ = -0.02d/s

AcX = 0.01g   | AcY = 0.00g   | AcZ = 0.99g
GyX = 0.09d/s  | GyY = 0.06d/s  | GyZ = -0.03d/s

AcX = 0.01g   | AcY = -0.00g  | AcZ = 0.99g
GyX = 0.03d/s  | GyY = -0.04d/s  | GyZ = -0.20d/s

AcX = 0.00g   | AcY = 0.00g   | AcZ = 0.99g
GyX = 0.04d/s  | GyY = -0.12d/s  | GyZ = -0.05d/s
```

Figure 2

to a level of precision. Likewise, the gyroscope has four ranges from ±250 to ±2000 degrees per second. The use of I^2C is shown in figure 1 inputting data from the sensor to the microcontroller. Seen in figure 2 is a screenshot of some of the sensor readings output from this setup. The utilized code is in figure 3, this code is from the SunFounder tutorial with calibration constants modified for this setup.

```
#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+0.02); Serial.print("g ");
  Serial.print(" | AcY = "); Serial.print(AcY / 65536 * ACCELE_RANGE+.015); Serial.print("g ");
  Serial.print(" | AcZ = "); Serial.print(AcZ / 65536 * ACCELE_RANGE+0.06); 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+.3); Serial.print("d/s ");
  Serial.print(" | GyY = "); Serial.print(GyY / 65536 * GYROSC_RANGE-.25); Serial.print("d/s ");
  Serial.print(" | GyZ = "); Serial.print(GyZ / 65536 * GYROSC_RANGE+0.25); Serial.println("d/s \n");
  delay(500);
}
```

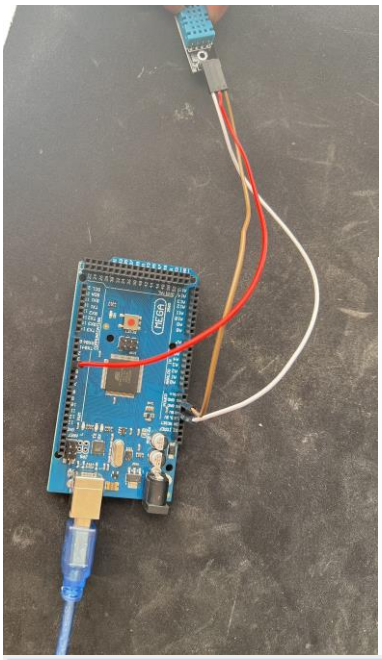
Figure 3

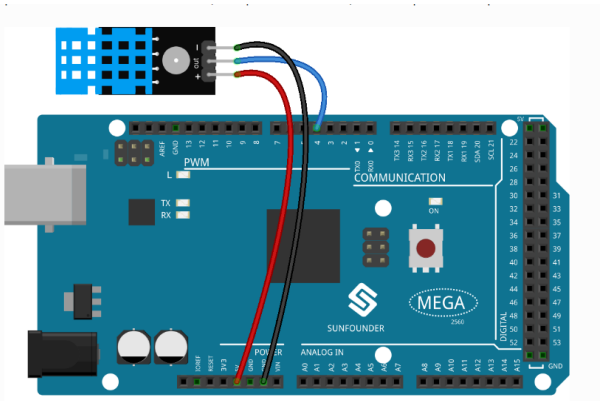
What follows is the proposed testing plan for the MPU6050. For this sensor, we will test the accelerometers and gyroscopes. The proposed static characteristics to be tested are Sensitivity, Error, Drift, and Linearity. For testing these sensors, a cell phone with accelerometers and gyroscopes will be used. The cell phone's sensors are the LSM6DSR gyroscope and the linear acceleration sensor by google. The cell phone's sensors are high accuracy and precision allowing them to act as an effective ground truth. The sensor will be attached to the phone through the sensor mounting holes to a 3D printed part. The devices will be

moved around to subject them to many accelerations and rotational velocities. Syncing both data sets in time will allow the input and output data to be compared.

DHT-11 Module

The working principle of the DHT-11 sensor is quite straightforward. It comprises a humidity sensing component and a thermistor for measuring temperature. The sensor operates by converting the surrounding air's humidity and temperature into electrical signals, which can then be interpreted by a microcontroller or similar device. The DHT-11 sensor's humidity sensing component consists of a moisture-sensitive capacitor. When the air around the sensor becomes more humid, the capacitance of this component increases. Conversely, when the air is drier, the capacitance decreases. This variation in capacitance is converted into a proportional electrical signal that represents the relative humidity of the environment. Alongside humidity measurement, the DHT-11 sensor also integrates a thermistor for temperature detection. Thermistors are temperature-sensitive resistors whose resistance changes with temperature variations. In the case of the DHT-11, the thermistor's resistance decreases as the temperature rises and increases as the temperature falls. This change in resistance is then converted into an electrical signal that corresponds to the ambient temperature. Both the humidity and temperature measurements from the DHT-11 sensor are output as digital signals, typically in the form of a serial data stream. This data can be read and processed by a microcontroller, which can then interpret and utilize the information for various applications, such as environmental monitoring, climate control, or weather forecasting. In summary, the DHT-11 sensor operates by sensing changes in humidity and temperature and converting these variations into digital signals for further analysis and application. It's simple design and reliable performance make it a popular choice for a wide range of projects requiring environmental monitoring capabilities.





Send

```
18:37:36.501 -> Temperature, Humidity = 24.00, 42.00
18:37:37.531 -> Temperature, Humidity = -999.00, -999.00
18:37:38.556 -> Temperature, Humidity = 24.00, 42.00
18:37:39.559 -> Temperature, Humidity = -999.00, -999.00
18:37:40.593 -> Temperature, Humidity = 24.00, 42.00
18:37:41.602 -> Temperature, Humidity = -999.00, -999.00
18:37:42.647 -> Temperature, Humidity = 24.00, 42.00
18:37:43.628 -> Temperature, Humidity = -999.00, -999.00
18:37:44.653 -> Temperature, Humidity = 24.00, 42.00
18:37:45.698 -> Temperature, Humidity = -999.00, -999.00
18:37:46.723 -> Temperature, Humidity = 24.00, 42.00
18:37:47.720 -> Temperature, Humidity = -999.00, -999.00
18:37:48.750 -> Temperature, Humidity = 24.00, 42.00
18:37:49.765 -> Temperature, Humidity = -999.00, -999.00
18:37:50.795 -> Temperature, Humidity = 24.00, 42.00
18:37:51.809 -> Temperature, Humidity = -999.00, -999.00
18:37:52.823 -> Temperature, Humidity = 24.00, 42.00
18:37:53.854 -> Temperature, Humidity = -999.00, -999.00
18:37:54.879 -> Temperature, Humidity = 24.00, 42.00
18:37:55.897 -> Temperature, Humidity = -999.00, -999.00
18:37:56.919 -> Temperature, Humidity = 24.00, 42.00
18:37:57.932 -> Temperature, Humidity = -999.00, -999.00
18:37:58.956 -> Temperature, Humidity = 24.00, 42.00
18:37:59.947 -> Temperature, Humidity = -999.00, -999.00
```

Code:

2.32_dhtModule.ino

```ino
#include "DHT.h"

#define DHTPIN 4  // Set the pin connected to the DHT11 data pin
#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);

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

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 temperature = dht.readTemperature();

  // Check if any reads failed and exit early (to try again).
  if (isnan(humidity) || isnan(temperature)) {
    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: ");
  Serial.print(temperature);
  Serial.println(" *C");
}
```

Range:

Utilize an oven for temperature adjustments and a humidifier for humidity control. Use a high-precision thermometer for reference temperature measurements. Humidity Control: utilize a spray bottle for localized humidity adjustments. Data Logging and Analysis: Develop a data logging system to continuously record sensor readings Evaluate the minimum and maximum values accurately measured by the DHT-11 sensor under different environmental conditions.

Sensitivity:

Utilize an oven for temperature adjustments and a humidifier for humidity control. Small Changes Introduction: Gradually introduce small changes in temperature and humidity within the controlled environment. Monitor the sensor's response to ensure it accurately detects and records these changes. Data Logging and Analysis: Implement a data logging system to record the sensor's responses to small changes in temperature and humidity. Analyze the sensor's accuracy in detecting these changes to assess its sensitivity accurately.

Resolution:

Utilize an oven for temperature adjustments and a humidifier for humidity control. Small Changes Detection: Gradually adjust the temperature and humidity within the controlled environment to introduce small variations. Monitor the sensor's readings to determine the smallest temperature and humidity changes it can detect. Data Logging and Analysis: Employ a data logging system to record the sensor's responses to small changes in temperature and humidity. Analyze the recorded data to determine the minimum temperature and humidity differences that the sensor can accurately distinguish, thereby assessing its resolution.

Ultrasonic Ranging Module

The Ultrasonic Ranging module is a sensor that provides measurements ranging from 2cm (about 0.79 in) to 400cm (about 13.12 ft) without contact. Its accuracy is claimed to be within 3mm. According to the sensor’s description, it can keep a stable signal within 5m. Beyond that it begins to weaken until fading after 7m. The way this module works is that it it sends out an ultrasonic signal, in a frequency unable to be heard by human ears. When the signal reaches another object it echos a signal and sends it back to the sensor. The sensor then calculates that time and converts it to a distance. According to the Sunfounder Module, the formula for such a calculation is: (time * sound speed(340m/s)/2).

Configuration:

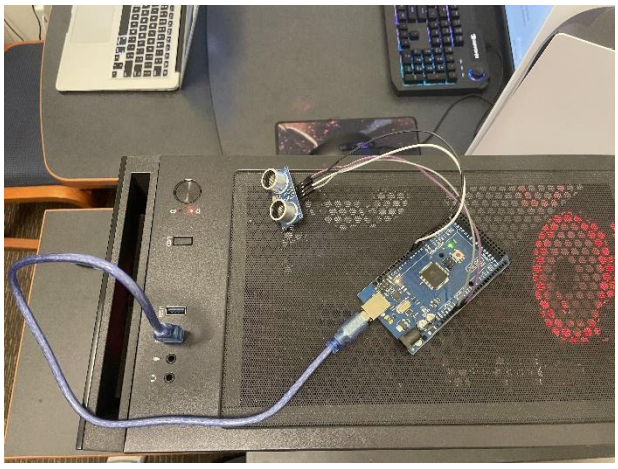
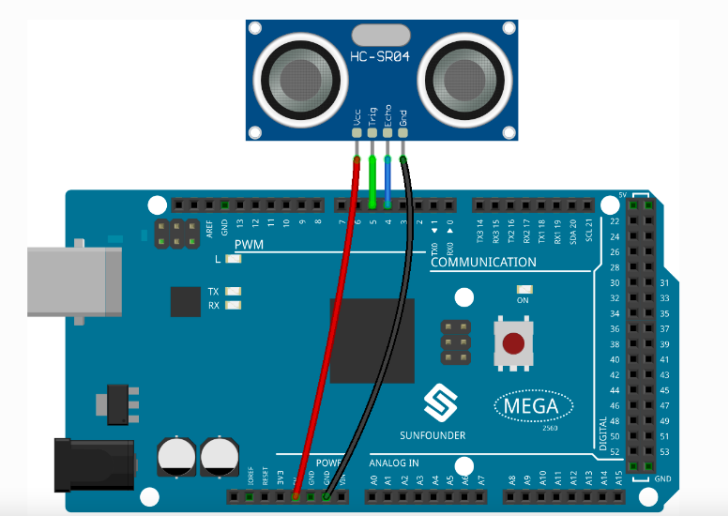


Diagram:



Code:

```
const int echoPin = 4;
const int trigPin = 5;
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);
  Serial.println("Ultrasonic sensor:");
}

void loop() {
  // put your main code here, to run repeatedly:
 float distance = readSensorData();
  Serial.print(distance);
  Serial.println(" cm");
  delay(400);
}

float readSensorData(){
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  float distance = pulseIn(echoPin, HIGH)/58.00;  //Equivalent to (340m/s*1us)/2
  return distance;

}
```

Sensor Readings:

```
1   const int echoPin = 4;
2   const int trigPin = 5;
3   void setup() {
4     // put your setup code here, to run once:
5     Serial.begin(9600);
6     pinMode(echoPin, INPUT);
7     pinMode(trigPin, OUTPUT);
8     Serial.println("Ultrasonic sensor:");
9   }
10
11  void loop() {
12    // put your main code here, to run repeatedly:
13   float distance = readSensorData();
14    Serial.print(distance);
15    Serial.println(" cm");
```

Output Serial Monitor ×

Message (Enter to send message to 'Arduino Mega or Mega 2560' on 'COM3')

```
12.43 cm
12.33 cm
12.33 cm
12.33 cm
12.33 cm
12.33 cm
12.45 cm
12.43 cm
12.38 cm
12.33 cm
12.33 cm
12.33 cm
```

Tests: We are going to test the measurement capabilities of the Ultrasound Ranging Module. These will consist of testing various static characteristics. The characteristics we will be testing are Range, Static Error, Repeatability, and Accuracy.

- Range: the site description of the sensor claims the sensor has a maximum stable measuring distance of 5m. We will test the distance by placing the sensor against a wall in a room with a length longer than 5m. We will measure increments along that floor and set an object x meters away from the sensor to see if it gives us an accurate measurement. This process will be repeated from less than 2cm to greater than 5m.

- Static Error: Record multiple data points at the same distance. Repeat this for multiple distances within its range of stability. From this data, you can find linearity and sensitivity.

- Repeatability: To measure the sensor's repeatability, we will simply repeat our range test for values within the stable range, then compare our new values with the values we measured in the first test. This could be repeated to gain a 3rd set of data points.

- Accuracy and Precision: We can use our data collected from range and repeatability in this step and determine if the sensor is measuring accurately and precisely.