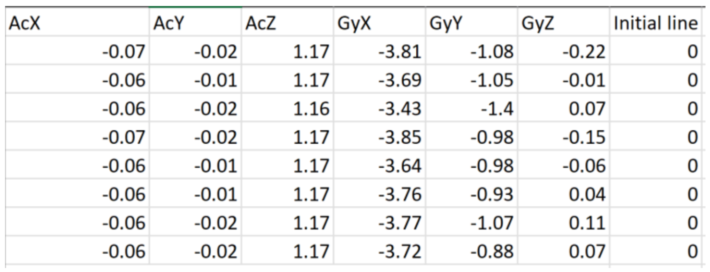
MRE 320

Sensors and Actuators

Segment 1 {Methodology of Sensor Testing}

I. MPU6050 Module {Accelerometer}

A. Sensor Reading {@ Stationary}



AcX	AcY	AcZ	GyX	GyY	GyZ	Initial line
-0.07	-0.02	1.17	-3.81	-1.08	-0.22	0
-0.06	-0.01	1.17	-3.69	-1.05	-0.01	0
-0.06	-0.02	1.16	-3.43	-1.4	0.07	0
-0.07	-0.02	1.17	-3.85	-0.98	-0.15	0
-0.06	-0.01	1.17	-3.64	-0.98	-0.06	0
-0.06	-0.01	1.17	-3.76	-0.93	0.04	0
-0.06	-0.02	1.17	-3.77	-1.07	0.11	0
-0.06	-0.02	1.17	-3.72	-0.88	0.07	0

Figure 1 – Accelerometer Readings {@ Stationary}

B. Arduino Code

```
#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);
}
```

C. Wiring Diagram {Hardware}

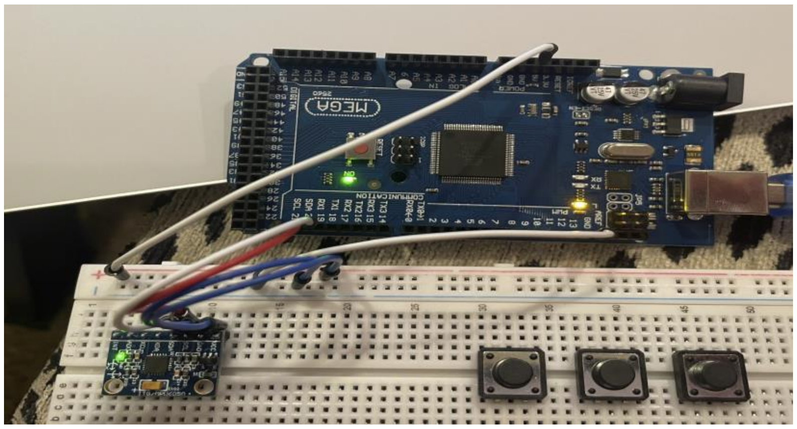


Figure 2 – Accelerometer Wiring

D. Summary of Findings

   i. While stationary, the sensor picks up readings of acceleration and angular velocity that is not present. Appears to be quite precise, however the sensor lags in its accuracy.

E. Static Characteristics {@ Stationary Position}

   b. Range

      i. According to data sheets,

         - Accelerometer ranges: ±2, ±4, ±8, ±16g
         - Gyroscope ranges: ± 250, 500, 1000, 2000 °/s

   c. Drift

      i. Initial reading

      AcX = 0.01g | AcY = 0.01g | AcZ = 1.18g

      GyX = -3.63d/s | GyY = -1.04d/s | GyZ = 0.10d/s

      ii. Reading after 5 minutes from initial reading

      AcX = 0.00g | AcY = 0.01g | AcZ = 1.18g

      GyX = -3.45d/s | GyY = -0.93d/s | GyZ = 0.30d/s

   The values remain almost identical at the Stationary position regardless of how much time passes.

   d. Sensitivity

      i. As seen in the below values, we can see that when moving the device up and down repeatedly, we get a change in value for our angular velocities. However, the acceleration in the x, y, and z did not change. This means that the sensitivity for the acceleration is NOT as sensitive as the angular velocities which is sensitive.

      AcX = -0.02g | AcY = 0.00g | AcZ = 1.18g

      GyX = 7.38d/s | GyY = -0.68d/s | GyZ = 0.99d/s

      AcX = 0.00g | AcY = -0.01g | AcZ = 1.16g

      GyX = -2.14d/s | GyY = -4.51d/s | GyZ = -2.05d/s

   e. Accuracy {@ Stationary 0}

      i. Since the sensor is not moving, the reading should output 0 for all accelerations and angular velocities. Since the values seen below are far from 0, the conclusion is the sensor is not accurate.

AcX = 0.01g | AcY = 0.01g | AcZ = 1.17g

GyX = -3.65d/s | GyY = -1.14d/s | GyZ = 0.04d/s

AcX = 0.00g | AcY = 0.01g | AcZ = 1.18g

GyX = -3.36d/s | GyY = -0.95d/s | GyZ = -0.16d/s

AcX = 0.01g | AcY = 0.01g | AcZ = 1.18g

GyX = -3.69d/s | GyY = -1

AcX = 0.01g | AcY = 0.01g | AcZ = 1.18g

GyX = -3.63d/s | GyY = -0.85d/s | GyZ = 0.04d/s

AcX = 0.01g | AcY = 0.00g | AcZ = 1.19g

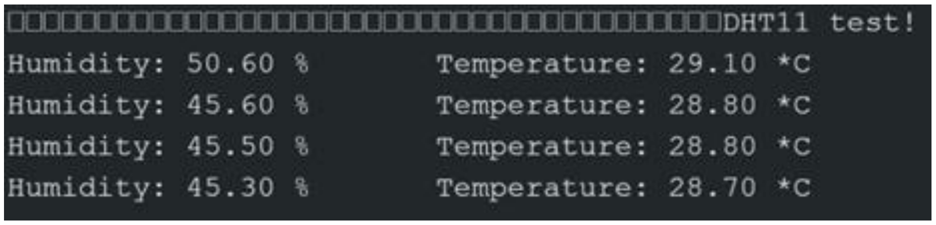
GyX = -3.75d/s | GyY = -0.71d/s | GyZ = 0.69d/s

f. Precision {@ Stationary 0}
   i. Once again looking at the data above, we see that although the values are not 0 leading to an inaccurate sensor, the sensor still reads with good precision.

F. Testing methodology
   i. Using an external battery, we can attach the sensor to a moving object such as a rover or a drone to test the angular velocities and acceleration of the sensor.
      ii. GROUND TRUTH: Deriving formula for constant acceleration.

II. DHT-11 {Temperature}

A. Sensor Reading {@ Stationary}



```
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□DHT11 test!
Humidity: 50.60 %       Temperature: 29.10 *C
Humidity: 45.60 %       Temperature: 28.80 *C
Humidity: 45.50 %       Temperature: 28.80 *C
Humidity: 45.30 %       Temperature: 28.70 *C
```

Figure 3 – DHT-11 Readings

B. Arduino Code

```
#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");
  }
```

C. Wiring Diagram {Hardware}

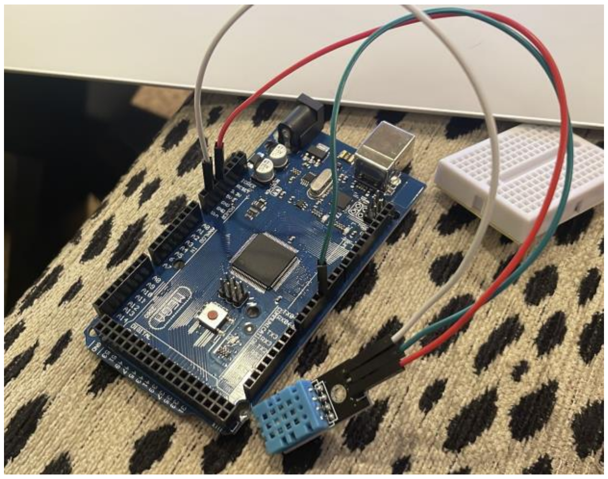


Figure 3 – DHT-11 {Temperature Sensor}

D. Summary of Findings
   i. The DHT sensor picks up readings that are easily repeatable with little drift and from the data collected the temperature readings are accurate although not completely accurate, yet the readings are precise.

E. Static Characteristics
   a. Repeatability
      i. The sensor steadily outputs the same value when repeatedly put into the same environment. There are occasionally a few variances in values, yet they are not too significant in value.
   b. Drift
      i. There is little drift present, and the sensor consistently outputs the same value when left in an environment.
   c. Accuracy
      i. The data collected is a few degrees off from the actual temperature in a room.
   d. Precision

        i. The data collected are all within a degree higher or lower from an initial temperature or humidity reading.

F. Testing Methodology
    ii. We can put a sensor into a could environment such a refrigerator then put it back into room temperature to determine the temperature and humidity changes.
    iii. Could turn on hot shower and let the bathroom steam due to the warm temperature. Putting sensor into the steamed room should increase the humidity.
    iv. GROUND TRUTH: Thermostat reading and humidifier.

III. Ultrasonic Ranging Module

A. Sensor Reading

First Reading Cycle	Second Reading Cycle	Second Reading Cycle
168.93cm	169.33cm	169.12cm
169.05cm	169.38cm	169.19cm
169.03cm	169.76cm	169.17cm
169.31cm	169.76cm	169.07cm
168.95cm	169.41cm	169.10cm
168.95cm	169.43cm	169.07cm
168.95cm	169.41cm	169.52cm
169.03cm	169.36cm	169.21cm
169.41cm	169.74cm	169.19cm

B. Arduino Code

```
const int echoPin = 4;
const int trigPin = 5;
void setup(){
  Serial.begin(9600);
  pinMode(echoPin, INPUT);
  pinMode(trigPin, OUTPUT);
```

```
 Serial.println("Ultrasonic sensor:");
}

void loop(){
 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;
}
```

C. Wiring Diagram {Hardware}

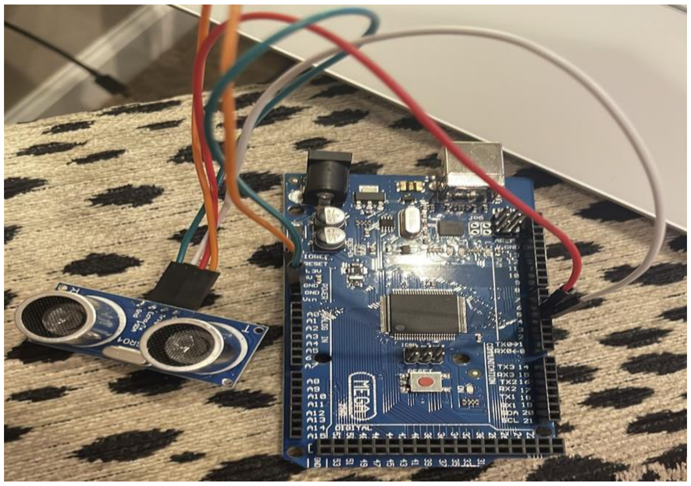


Figure 4 – Ultrasonic Ranging Module

D. Summary of Findings

i. Overall, the sensor is accurate and precise. According to the actual measurement of the tape measure, the distance was 173.25cm (about 5.68 ft). As seen in the above table, we averaged approximately 169.5cm (about 5.56 ft). This value is close to the actual measured value. This gives us an accurate sensor. When looking at the above table again, we see the values are all quite if not identical close. This means we also have a precise sensor.

E. Static Characteristics
    b. Range
        i. 2-400cm
    c. Saturation
        i. Approximately 2.1cm (about 0.83 in)

Reading
2.12cm
2.10cm
2.10cm
2.12cm
2.12cm
2.11cm

    d. Sensitivity
        i. Very sensitive. Even the quickest wave of a hand through the path of the sensor causes the sensor to output a reasonable output value. This does decrease as the distance from the sensor increases. Max range in 400cm (about 13.12 ft). Approximately 180-200cm (about 6.56 ft) has a significant drop off. Will still detect, but not as precise.
    e. Repeatability
        i. Very good. Repeats very accurately as seen below.

Test 1	Test 2
3.09cm	3.10cm
3.10cm	3.09cm
3.09cm	3.09cm
3.09cm	3.09cm
2.98cm	3.09cm
3.09cm	3.09cm
3.09cm	2.98cm

F. Testing Methodology
        ii. Using a notecard and measuring tape, we can determine the actual distance of the notecard from the sensor using the

measuring tape, then compare these values with the sensor output.

iii. Using a tape measure once again, we can use a wide container/box that may be able to aid in the accuracy of the reading. This will enable us to check if external noise tampers with the output of the sensor readings. We will use the same procedure as the note card, only now with a container/box.