



The MPU6050 sensor provides raw data for three axes (X, Y, and Z) from both the accelerometer and the gyroscope. These raw values can be used to calculate the pitch, roll, and yaw of an object. Here's how they relate:

1. Understanding Gyroscope Output:

- The gyroscope measures the rate of rotation around the three axes: X, Y, and Z. These rates are provided in degrees per second (dps).
- Gyroscope outputs:
 - g_x : Rate of rotation around the X-axis.
 - g_y : Rate of rotation around the Y-axis.
 - g_z : Rate of rotation around the Z-axis.

2. Pitch, Roll, and Yaw Definitions:

- **Pitch:** Rotation around the Y-axis. It tilts the object forward or backward.
- **Roll:** Rotation around the X-axis. It tilts the object side to side.
- **Yaw:** Rotation around the Z-axis. It rotates the object left or right on the horizontal plane.

3. Calculating Pitch, Roll, and Yaw from Gyroscope Data:

- To compute pitch, roll, and yaw angles from the gyroscope data, you need to integrate the gyroscope output over time. This is because the gyroscope measures the rate of change of the angle (angular velocity), not the angle itself.

4. Relation Between Gyroscope Data and Euler Angles (Pitch, Roll, Yaw):

- **Pitch (θ):**
 - Calculated primarily using the accelerometer data but also corrected over time using the gyroscope to avoid drift.
 - In a simple scenario (ignoring drift correction), pitch can be calculated using:

$$\text{Pitch} = \text{Pitch}_{\text{previous}} + g_x \times \Delta t$$

$$\text{Pitch} = \text{Pitch}_{\text{previous}} + g_x \times \Delta t$$
 - Where g_x is the angular velocity around the X-axis and Δt is the time interval.
- **Roll (ϕ):**
 - Similarly, roll can be calculated as:

$$\text{Roll} = \text{Roll}_{\text{previous}} + g_y \times \Delta t$$

$$\text{Roll} = \text{Roll}_{\text{previous}} + g_y \times \Delta t$$
 - Where g_y is the angular velocity around the Y-axis.
- **Yaw (ψ):**
 - Yaw is computed using the gyroscope data around the Z-axis:

$$\text{Yaw} = \text{Yaw}_{\text{previous}} + g_z \times \Delta t$$

$$\text{Yaw} = \text{Yaw}_{\text{previous}} + g_z \times \Delta t$$
 - Where g_z is the angular velocity around the Z-axis.

5. Complementary Filter Approach:

- A complementary filter can be used to combine the accelerometer and gyroscope data. The accelerometer data is used to correct for drift in the pitch and roll calculations, while the gyroscope data provides smoother, more responsive measurements.
- For example:
$$\text{Pitch} = (1 - \alpha) \times (\text{Pitch} + g_x \times \Delta t) + \alpha \times \text{AccelPitch}$$

$$\text{Pitch} = (1 - \alpha) \times (\text{Pitch} + g_x \times \Delta t) + \alpha \times \text{AccelPitch}$$

$$\text{Roll} = (1 - \alpha) \times (\text{Roll} + g_y \times \Delta t) + \alpha \times \text{AccelRoll}$$

$$\text{Roll} = (1 - \alpha) \times (\text{Roll} + g_y \times \Delta t) + \alpha \times \text{AccelRoll}$$
- Here, AccelPitch and AccelRoll are the angles calculated using the accelerometer data, and α is the filter coefficient.

6. Accelerometer Contribution:

- The accelerometer helps determine the orientation of the device relative to the Earth's surface, which is crucial for determining the pitch and roll angles:

$$\text{Pitch (Accel)} = \text{atan2}(a_y, \sqrt{a_x^2 + a_z^2})$$

$$\text{Pitch (Accel)} = \text{atan2}(a_y, \sqrt{a_x^2 + a_z^2})$$

$$\text{Roll (Accel)} = \text{atan2}(-a_x, a_z)$$

$$\text{Roll (Accel)} = \text{atan2}(-a_x, a_z)$$

7. Yaw and Drift:

- Unlike pitch and roll, yaw is more challenging to calculate accurately because it can drift over time. Without a magnetometer to correct for this drift, the yaw angle can accumulate errors.

Sensor Data Display

MPU6050 #1 Data

Acceleration X: -0.006755371
Acceleration Y: -0.00159668
Acceleration Z: 0.999528825
Gyroscope X: 0.050228719
Gyroscope Y: 0.030687012
Gyroscope Z: 0.020381743
Displacement X: -14.03 m
Displacement Y: 0.41 m
Displacement Z: 4420.88 m

MPU6050 #2 Data

Acceleration X: -0.001308594
Acceleration Y: 0.001572266
Acceleration Z: 1.001137733
Gyroscope X: -0.012366401
Gyroscope Y: -0.023129843
Gyroscope Z: -0.007633494
Displacement X: -1.99 m
Displacement Y: 7.32 m
Displacement Z: 4423.79 m

ADXL345 Data

Acceleration X: -2.412436485
Acceleration Y: -5.851037979
Acceleration Z: 7.178860664

Toggle MPU6050 #1 ON

Toggle MPU6050 #2 ON

Toggle ADXL345 ON

MPU6050 #1
Pitch: -0.09°
Roll: 0.39°
Yaw: 1.17°
MPU6050 #2
Pitch: 0.09°
Roll: 0.07°
Yaw: -0.44°



Activate Windows
Go to Settings to activate Windows.

Arduino Code:

```
#include <Wire.h>
```

```
#include <MPU6050.h>
```

```
#include <Adafruit_Sensor.h>
```

```
#include <Adafruit_ADXL345_U.h>
```

```
#include <ArduinoJson.h>
```

```
MPU6050 mpu1(0x68); // Default I2C address
```

```
MPU6050 mpu2(0x69); // Alternate I2C address
```

```
Adafruit_ADXL345_Unified adxl = Adafruit_ADXL345_Unified(12345);
```

```
const int numReadings = 100; // Number of readings for the moving average, increased  
for better smoothing
```

```
// Arrays to store readings for moving average
```

```
float mpu1_axReadings[numReadings];
```

```
float mpu1_ayReadings[numReadings];
```

```
float mpu1_azReadings[numReadings];
```

```
float mpu1_gxReadings[numReadings];
```

```
float mpu1_gyReadings[numReadings];
```

```
float mpu1_gzReadings[numReadings];
```

```
float mpu2_axReadings[numReadings];
```

```
float mpu2_ayReadings[numReadings];
```

```
float mpu2_azReadings[numReadings];
```

```
float mpu2_gxReadings[numReadings];
```

```
float mpu2_gyReadings[numReadings];
```

```
float mpu2_gzReadings[numReadings];
```

```
float adxl_axReadings[numReadings];
```

```
float adxl_ayReadings[numReadings];
```

```
float adxl_azReadings[numReadings];
```

```
int readIndex = 0;
```

```
float total_mpu1_ax = 0;
```

```
float total_mpu1_ay = 0;
```

```
float total_mpu1_az = 0;
```

```
float total_mpu1_gx = 0;
```

```
float total_mpu1_gy = 0;
```

```
float total_mpu1_gz = 0;
```

```
float total_mpu2_ax = 0;
```

```
float total_mpu2_ay = 0;
```

```
float total_mpu2_az = 0;
```

```
float total_mpu2_gx = 0;
```

```
float total_mpu2_gy = 0;
```

```
float total_mpu2_gz = 0;
```

```
float total_adxl_ax = 0;
```

```
float total_adxl_ay = 0;
```

```
float total_adxl_az = 0;
```

```
bool mpu1Enabled = true;
```

```
bool mpu2Enabled = true;
```

```
bool adxlEnabled = true;
```

```
void setup() {
```

```
    Serial.begin(9600);
```

```
    while (!Serial) {
```

```
        ; // Wait for serial port to connect
```

```
    }
```

```
    Wire.begin();
```

```
    // Adding a delay to ensure the serial connection is fully established
```

```
    delay(1000);
```

```
// Initialize first MPU6050 sensor

mpu1.initialize();

if (mpu1.testConnection()) {

    sendInitStatus("MPU6050 #1 connection successful");

} else {

    sendInitStatus("MPU6050 #1 connection failed");

}


// Initialize second MPU6050 sensor

mpu2.initialize();

if (mpu2.testConnection()) {

    sendInitStatus("MPU6050 #2 connection successful");

} else {

    sendInitStatus("MPU6050 #2 connection failed");

}


// Initialize ADXL345 sensor

if (adxl.begin()) {

    sendInitStatus("ADXL345 connection successful");

} else {

    sendInitStatus("ADXL345 connection failed");

}


// Calibrate the first sensor

if (mpu1.testConnection()) {

    mpu1.CalibrateAccel(6);

    mpu1.CalibrateGyro(6);

    sendInitStatus("Calibration completed for MPU6050 #1");

}
```

```
}
```

```
// Calibrate the second sensor
```

```
if (mpu2.testConnection()) {
```

```
    mpu2.CalibrateAccel(6);
```

```
    mpu2.CalibrateGyro(6);
```

```
    sendInitStatus("Calibration completed for MPU6050 #2");
```

```
}
```

```
// Initialize readings arrays to zero
```

```
for (int i = 0; i < numReadings; i++) {
```

```
    mpu1_axReadings[i] = 0;
```

```
    mpu1_ayReadings[i] = 0;
```

```
    mpu1_azReadings[i] = 0;
```

```
    mpu1_gxReadings[i] = 0;
```

```
    mpu1_gyReadings[i] = 0;
```

```
    mpu1_gzReadings[i] = 0;
```

```
    mpu2_axReadings[i] = 0;
```

```
    mpu2_ayReadings[i] = 0;
```

```
    mpu2_azReadings[i] = 0;
```

```
    mpu2_gxReadings[i] = 0;
```

```
    mpu2_gyReadings[i] = 0;
```

```
    mpu2_gzReadings[i] = 0;
```

```
    adxl_axReadings[i] = 0;
```

```
    adxl_ayReadings[i] = 0;
```

```
    adxl_azReadings[i] = 0;
```

```
}  
}
```

```
void sendInitStatus(const char* status) {  
    StaticJsonDocument<200> doc;  
    doc["status"] = status;  
    String output;  
    serializeJson(doc, output);  
    Serial.print("<JSON>");  
    Serial.print(output);  
    Serial.println("</JSON>");  
}
```

```
void sendSensorData() {  
    // Read accelerometer and gyroscope data from first MPU6050  
    if (mpu1Enabled) {  
        int16_t ax1, ay1, az1, gx1, gy1, gz1;  
        mpu1.getMotion6(&ax1, &ay1, &az1, &gx1, &gy1, &gz1);  
        float g_ax1 = ax1 / 16384.0;  
        float g_ay1 = ay1 / 16384.0;  
        float g_az1 = az1 / 16384.0;  
        float dps_gx1 = gx1 / 131.0;  
        float dps_gy1 = gy1 / 131.0;  
        float dps_gz1 = gz1 / 131.0;  
  
        // Update moving average arrays  
        total_mpu1_ax -= mpu1_axReadings[readIndex];  
        total_mpu1_ay -= mpu1_ayReadings[readIndex];
```



```

total_mpu1_az -= mpu1_azReadings[readIndex];
total_mpu1_gx -= mpu1_gxReadings[readIndex];
total_mpu1_gy -= mpu1_gyReadings[readIndex];
total_mpu1_gz -= mpu1_gzReadings[readIndex];

mpu1_axReadings[readIndex] = g_ax1;
mpu1_ayReadings[readIndex] = g_ay1;
mpu1_azReadings[readIndex] = g_az1;
mpu1_gxReadings[readIndex] = dps_gx1;
mpu1_gyReadings[readIndex] = dps_gy1;
mpu1_gzReadings[readIndex] = dps_gz1;

total_mpu1_ax += mpu1_axReadings[readIndex];
total_mpu1_ay += mpu1_ayReadings[readIndex];
total_mpu1_az += mpu1_azReadings[readIndex];
total_mpu1_gx += mpu1_gxReadings[readIndex];
total_mpu1_gy += mpu1_gyReadings[readIndex];
total_mpu1_gz += mpu1_gzReadings[readIndex];
}

// Read accelerometer and gyroscope data from second MPU6050
if (mpu2Enabled) {
    int16_t ax2, ay2, az2, gx2, gy2, gz2;
    mpu2.getMotion6(&ax2, &ay2, &az2, &gx2, &gy2, &gz2);
    float g_ax2 = ax2 / 16384.0;
    float g_ay2 = ay2 / 16384.0;
    float g_az2 = az2 / 16384.0;
    float dps_gx2 = gx2 / 131.0;

```

```

float dps_gy2 = gy2 / 131.0;

float dps_gz2 = gz2 / 131.0;


// Update moving average arrays

total_mpu2_ax -= mpu2_axReadings[readIndex];
total_mpu2_ay -= mpu2_ayReadings[readIndex];
total_mpu2_az -= mpu2_azReadings[readIndex];
total_mpu2_gx -= mpu2_gxReadings[readIndex];
total_mpu2_gy -= mpu2_gyReadings[readIndex];
total_mpu2_gz -= mpu2_gzReadings[readIndex];


mpu2_axReadings[readIndex] = g_ax2;
mpu2_ayReadings[readIndex] = g_ay2;
mpu2_azReadings[readIndex] = g_az2;
mpu2_gxReadings[readIndex] = dps_gx2;
mpu2_gyReadings[readIndex] = dps_gy2;
mpu2_gzReadings[readIndex] = dps_gz2;


total_mpu2_ax += mpu2_axReadings[readIndex];
total_mpu2_ay += mpu2_ayReadings[readIndex];
total_mpu2_az += mpu2_azReadings[readIndex];
total_mpu2_gx += mpu2_gxReadings[readIndex];
total_mpu2_gy += mpu2_gyReadings[readIndex];
total_mpu2_gz += mpu2_gzReadings[readIndex];
}


// Read sensor data from ADXL345

if (adxlEnabled) {

```

```

sensors_event_t event;

adxl_getEvent(&event);

// Update moving average arrays

total_adxl_ax -= adxl_axReadings[readIndex];
total_adxl_ay -= adxl_ayReadings[readIndex];
total_adxl_az -= adxl_azReadings[readIndex];

adxl_axReadings[readIndex] = event.acceleration.x;
adxl_ayReadings[readIndex] = event.acceleration.y;
adxl_azReadings[readIndex] = event.acceleration.z;

total_adxl_ax += adxl_axReadings[readIndex];
total_adxl_ay += adxl_ayReadings[readIndex];
total_adxl_az += adxl_azReadings[readIndex];
}

readIndex++;

if (readIndex >= numReadings) {
    readIndex = 0;
}

float avg_mpu1_ax = total_mpu1_ax / numReadings;
float avg_mpu1_ay = total_mpu1_ay / numReadings;
float avg_mpu1_az = total_mpu1_az / numReadings;
float avg_mpu1_gx = total_mpu1_gx / numReadings;
float avg_mpu1_gy = total_mpu1_gy / numReadings;
float avg_mpu1_gz = total_mpu1_gz / numReadings;

```

```
float avg_mpu2_ax = total_mpu2_ax / numReadings;  
float avg_mpu2_ay = total_mpu2_ay / numReadings;  
float avg_mpu2_az = total_mpu2_az / numReadings;  
float avg_mpu2_gx = total_mpu2_gx / numReadings;  
float avg_mpu2_gy = total_mpu2_gy / numReadings;  
float avg_mpu2_gz = total_mpu2_gz / numReadings;
```

```
float avg_adxl_ax = total_adxl_ax / numReadings;  
float avg_adxl_ay = total_adxl_ay / numReadings;  
float avg_adxl_az = total_adxl_az / numReadings;
```

```
// Create a JSON document
```

```
StaticJsonDocument<800> doc;  
doc["mpu1"]["ax"] = avg_mpu1_ax;  
doc["mpu1"]["ay"] = avg_mpu1_ay;  
doc["mpu1"]["az"] = avg_mpu1_az;  
doc["mpu1"]["gx"] = avg_mpu1_gx;  
doc["mpu1"]["gy"] = avg_mpu1_gy;  
doc["mpu1"]["gz"] = avg_mpu1_gz;
```

```
doc["mpu2"]["ax"] = avg_mpu2_ax;  
doc["mpu2"]["ay"] = avg_mpu2_ay;  
doc["mpu2"]["az"] = avg_mpu2_az;  
doc["mpu2"]["gx"] = avg_mpu2_gx;  
doc["mpu2"]["gy"] = avg_mpu2_gy;  
doc["mpu2"]["gz"] = avg_mpu2_gz;
```

```

doc["adxl"]["ax"] = avg_adxl_ax;
doc["adxl"]["ay"] = avg_adxl_ay;
doc["adxl"]["az"] = avg_adxl_az;

// Serialize JSON to a String
String data;
serializeJson(doc, data);

// Print JSON string to serial monitor
Serial.print("<JSON>");
Serial.print(data);
Serial.println("</JSON>");
}

void loop() {
  // Continuously send sensor data
  sendSensorData();
  delay(50); // Adjust delay as needed
}

```

Server Code

```

const express = require('express');
const http = require('http');
const WebSocket = require('ws');
const bodyParser = require('body-parser');
const { SerialPort, ReadlineParser } = require('serialport');

```

```
const path = require('path');

const app = express();
app.use(bodyParser.json());

// Serve the HTML file
app.use(express.static(path.join(__dirname, 'public')));

const server = http.createServer(app);
const wss = new WebSocket.Server({ server });

let wsClient = null;

wss.on('connection', (ws) => {
  console.log('WebSocket client connected');
  wsClient = ws;

  ws.on('message', (message) => {
    console.log('Received message from client:', message);
    if (message === 'toggleMpu1') {
      port.write('toggleMpu1\n');
    } else if (message === 'toggleMpu2') {
      port.write('toggleMpu2\n');
    } else if (message === 'toggleAdxl') {
      port.write('toggleAdxl\n');
    }
  });
});
```

```

const serialPortPath = 'COM3'; // Update with your serial port path
const port = new SerialPort({ path: serialPortPath, baudRate: 9600 });
const parser = port.pipe(new ReadlineParser({ delimiter: '\n' }));

parser.on('data', (data) => {
  console.log('Received sensor data:', data);

  // Extract JSON data between <JSON> and </JSON> markers
  const jsonData = data.match(/<JSON>(.*?)</JSON>/);
  if (jsonData && jsonData[1]) {
    try {
      const sensorData = JSON.parse(jsonData[1].trim());

      if (sensorData.status) {
        console.log('Initialization status:', sensorData.status);
      } else {
        // Calculate displacement for MPU1 and MPU2

        sensorData.mpu1.displacement =
calculateDisplacement(sensorData.mpu1.ax, sensorData.mpu1.ay,
sensorData.mpu1.az, 'mpu1');

        sensorData.mpu2.displacement =
calculateDisplacement(sensorData.mpu2.ax, sensorData.mpu2.ay,
sensorData.mpu2.az, 'mpu2');

        if (wsClient && wsClient.readyState === WebSocket.OPEN) {
          wsClient.send(JSON.stringify(sensorData));
        }
      }
    } catch (error) {

```

```
        console.error('Error parsing JSON:', error);
    }
} else {
    console.error('Non-JSON data received, ignoring:', data);
}
});
```

```
server.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

```
// Function to calculate displacement based on acceleration
```

```
let prevAx = { mpu1: 0, mpu2: 0 }, prevAy = { mpu1: 0, mpu2: 0 }, prevAz = { mpu1: 0,
mpu2: 0 };
```

```
let velX = { mpu1: 0, mpu2: 0 }, velY = { mpu1: 0, mpu2: 0 }, velZ = { mpu1: 0, mpu2: 0 };
```

```
let dispX = { mpu1: 0, mpu2: 0 }, dispY = { mpu1: 0, mpu2: 0 }, dispZ = { mpu1: 0, mpu2: 0
};
```

```
// Filtered displacement variables
```

```
let filteredDispX = { mpu1: 0, mpu2: 0 };
```

```
let filteredDispY = { mpu1: 0, mpu2: 0 };
```

```
let filteredDispZ = { mpu1: 0, mpu2: 0 };
```

```
const dt = 0.01; // Time step in seconds
```

```
const alphaDisp = 0.1; // Low-pass filter constant
```

```
function calculateDisplacement(ax, ay, az, mpu) {
```

```
    // Calculate velocity from acceleration
```

```
    velX[mpu] += 0.5 * (prevAx[mpu] + ax) * dt;
```

```
    velY[mpu] += 0.5 * (prevAy[mpu] + ay) * dt;
```



```

    velZ[mpu] += 0.5 * (prevAz[mpu] + az) * dt;

    // Calculate displacement from velocity
    dispX[mpu] += velX[mpu] * dt;
    dispY[mpu] += velY[mpu] * dt;
    dispZ[mpu] += velZ[mpu] * dt;

    // Apply low-pass filter to displacement
    filteredDispX[mpu] = alphaDisp * dispX[mpu] + (1 - alphaDisp) * filteredDispX[mpu];
    filteredDispY[mpu] = alphaDisp * dispY[mpu] + (1 - alphaDisp) * filteredDispY[mpu];
    filteredDispZ[mpu] = alphaDisp * dispZ[mpu] + (1 - alphaDisp) * filteredDispZ[mpu];

    // Update previous acceleration values
    prevAx[mpu] = ax;
    prevAy[mpu] = ay;
    prevAz[mpu] = az;

    return { x: filteredDispX[mpu], y: filteredDispY[mpu], z: filteredDispZ[mpu] };
}

```

HTML Code

```

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Sensor Data Display</title>

  <style>

    #container {

```

```
display: flex;

justify-content: space-around;

align-items: center;
}

#sensorData {

width: 30%;

padding-right: 20px;

text-align: left;
}

#canvasContainer {

width: 40%;

display: flex;

align-items: center;

justify-content: center;

position: relative;
}

#threeCanvas {

width: 100%;

height: 100%;
}

.button-container {

margin: 10px 0;

text-align: center;
}

.button-container button {

margin-right: 10px;

padding: 10px;

border: none;
```

```
    color: white;
    cursor: pointer;
}
.on {
    background-color: green;
}
.off {
    background-color: red;
}
.angle-display {
    position: absolute;
    left: 10px;
    top: 10px;
    color: black;
    font-size: 20px;
    text-align: left;
    background-color: rgba(255, 255, 255, 0.5);
    padding: 10px;
    border-radius: 5px;
}
.angle-section {
    margin-bottom: 20px;
}
</style>

<script src="https://cdn.jsdelivr.net/npm/three@0.129.0/build/three.min.js"></script>
</head>
<body>
<h1>Sensor Data Display</h1>
```

```
<div class="button-container">

  <button id="toggleMpu1" class="on">Toggle MPU6050 #1 ON</button>

  <button id="toggleMpu2" class="on">Toggle MPU6050 #2 ON</button>

  <button id="toggleAdxl" class="on">Toggle ADXL345 ON</button>

</div>
```

```
<div id="container">

  <div id="sensorData"></div>

  <div id="canvasContainer">

    <canvas id="threeCanvas"></canvas>

    <div class="angle-display">

      <div id="mpu1Angles" class="angle-section"></div>

      <div id="mpu2Angles" class="angle-section"></div>

    </div>

  </div>

</div>
```

```
<script>

  const socket = new WebSocket('ws://localhost:3000');

  let mpu1Enabled = true;

  let mpu2Enabled = true;

  let adxlEnabled = true;

  const toggleButton = (button, state) => {

    button.classList.toggle('on', state);

    button.classList.toggle('off', !state);

    button.textContent = button.textContent.replace(/ (ON|OFF)$/, '') + (state ? ' ON' : ' OFF');

  };

</script>
```

```
const mpu1Button = document.getElementById('toggleMpu1');
const mpu2Button = document.getElementById('toggleMpu2');
const adxlButton = document.getElementById('toggleAdxl');
const mpu1Angles = document.getElementById('mpu1Angles');
const mpu2Angles = document.getElementById('mpu2Angles');
```

```
mpu1Button.addEventListener('click', () => {
  mpu1Enabled = !mpu1Enabled;
  toggleButton(mpu1Button, mpu1Enabled);
  socket.send('toggleMpu1');
});
```

```
mpu2Button.addEventListener('click', () => {
  mpu2Enabled = !mpu2Enabled;
  toggleButton(mpu2Button, mpu2Enabled);
  socket.send('toggleMpu2');
});
```

```
adxlButton.addEventListener('click', () => {
  adxlEnabled = !adxlEnabled;
  toggleButton(adxlButton, adxlEnabled);
  socket.send('toggleAdxl');
});
```

```
socket.addEventListener('open', function (event) {
  console.log('WebSocket connection opened');
});
```

```
socket.addEventListener('message', function (event) {  
    const sensorData = JSON.parse(event.data);  
    updateSensorData(sensorData);  
    updateHand(  
        sensorData.mpu1.ax, sensorData.mpu1.ay, sensorData.mpu1.az,  
sensorData.mpu1.gx, sensorData.mpu1.gy, sensorData.mpu1.gz,  
        sensorData.mpu2.ax, sensorData.mpu2.ay, sensorData.mpu2.az,  
sensorData.mpu2.gx, sensorData.mpu2.gy, sensorData.mpu2.gz,  
        sensorData.adxl.ax, sensorData.adxl.ay, sensorData.adxl.az  
    );  
});
```

```
socket.addEventListener('close', function (event) {  
    console.log('WebSocket connection closed');  
});
```

```
socket.addEventListener('error', function (event) {  
    console.error('WebSocket error:', event);  
});
```

```
function updateSensorData(sensorData) {  
    const sensorDataDiv = document.getElementById('sensorData');  
    sensorDataDiv.innerHTML = ""; // Clear previous data  
  
    if (mpu1Enabled) {  
        const ax1 = sensorData.mpu1.ax;  
        const ay1 = sensorData.mpu1.ay;  
        const az1 = sensorData.mpu1.az;
```

```

const gx1 = sensorData.mpu1.gx;
const gy1 = sensorData.mpu1.gy;
const gz1 = sensorData.mpu1.gz;

const sensorDiv1 = document.createElement('div');
sensorDiv1.innerHTML = `<h2>MPU6050 #1 Data</h2>

    <p>Acceleration X: ${ax1}</p>
    <p>Acceleration Y: ${ay1}</p>
    <p>Acceleration Z: ${az1}</p>
    <p>Gyroscope X: ${gx1}</p>
    <p>Gyroscope Y: ${gy1}</p>
    <p>Gyroscope Z: ${gz1}</p>
    <p>Displacement X: ${sensorData.mpu1.displacement.x.toFixed(2)}
m</p>
    <p>Displacement Y: ${sensorData.mpu1.displacement.y.toFixed(2)}
m</p>
    <p>Displacement Z: ${sensorData.mpu1.displacement.z.toFixed(2)}
m</p>`;

    sensorDataDiv.appendChild(sensorDiv1);

}

if (mpu2Enabled) {
    const ax2 = sensorData.mpu2.ax;
    const ay2 = sensorData.mpu2.ay;
    const az2 = sensorData.mpu2.az;
    const gx2 = sensorData.mpu2.gx;
    const gy2 = sensorData.mpu2.gy;
    const gz2 = sensorData.mpu2.gz;

```

```

const sensorDiv2 = document.createElement('div');
sensorDiv2.innerHTML = `<h2>MPU6050 #2 Data</h2>

    <p>Acceleration X: ${ax2}</p>
    <p>Acceleration Y: ${ay2}</p>
    <p>Acceleration Z: ${az2}</p>
    <p>Gyroscope X: ${gx2}</p>
    <p>Gyroscope Y: ${gy2}</p>
    <p>Gyroscope Z: ${gz2}</p>
    <p>Displacement X: ${sensorData.mpu2.displacement.x.toFixed(2)}
m</p>
    <p>Displacement Y: ${sensorData.mpu2.displacement.y.toFixed(2)}
m</p>
    <p>Displacement Z: ${sensorData.mpu2.displacement.z.toFixed(2)}
m</p>`;

sensorDataDiv.appendChild(sensorDiv2);
}

if (adxlEnabled) {
    const adxl_ax = sensorData.adxl.ax;
    const adxl_ay = sensorData.adxl.ay;
    const adxl_az = sensorData.adxl.az;

    const sensorDiv3 = document.createElement('div');
    sensorDiv3.innerHTML = `<h2>ADXL345 Data</h2>

        <p>Acceleration X: ${adxl_ax}</p>
        <p>Acceleration Y: ${adxl_ay}</p>
        <p>Acceleration Z: ${adxl_az}</p>`;

    sensorDataDiv.appendChild(sensorDiv3);
}

```



```
}
```

```
// Set up the Three.js scene
```

```
const scene = new THREE.Scene();
```

```
const camera = new THREE.PerspectiveCamera(75, window.innerWidth /  
window.innerHeight, 0.1, 1000);
```

```
const renderer = new THREE.WebGLRenderer({ canvas:  
document.getElementById('threeCanvas'), antialias: true });
```

```
renderer.setSize(window.innerWidth / 2, window.innerHeight);
```

```
document.getElementById('canvasContainer').appendChild(renderer.domElement);
```

```
const hand1 = new THREE.Group();
```

```
const hand2 = new THREE.Group();
```

```
const hand3 = new THREE.Group(); // Hand for ADXL345
```

```
const armGeometry = new THREE.BoxGeometry(0.5, 0.3, 0.5);
```

```
const armMaterial = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
```

```
const arm1 = new THREE.Mesh(armGeometry, armMaterial);
```

```
arm1.position.set(0, -2.6, 0);
```

```
hand1.add(arm1);
```

```
const arm2 = new THREE.Mesh(armGeometry, armMaterial);
```

```
arm2.position.set(0, -2.6, 0);
```

```
hand2.add(arm2);
```

```
const arm3 = new THREE.Mesh(armGeometry, armMaterial);
```

```
arm3.position.set(0, -2.6, 0);
```

```
hand3.add(arm3);
```

```
const fingerGeometry = new THREE.BoxGeometry(2.5, 0.3, 2.5);
```

```
const fingerMaterial = new THREE.MeshBasicMaterial({ color: 0xff0000 });
```

```
const finger1 = new THREE.Mesh(fingerGeometry, fingerMaterial);
finger1.position.set(0, -3, 0);
hand1.add(finger1);

const finger2 = new THREE.Mesh(fingerGeometry, fingerMaterial);
finger2.position.set(0, -3, 0);
hand2.add(finger2);

const finger3 = new THREE.Mesh(fingerGeometry, fingerMaterial);
finger3.position.set(0, -3, 0);
hand3.add(finger3);
```

```
hand1.position.set(0, 3, 0);
hand2.position.set(0, 0, 0);
hand3.position.set(0, -3, 0);
```

```
scene.add(hand1);
scene.add(hand2);
scene.add(hand3);
```

```
camera.position.z = 10;
```

```
const animate = function () {
  requestAnimationFrame(animate);
  renderer.render(scene, camera);
};
```

```
animate();
```

```
const alpha = 0.02; // Smoothing factor for angles
```

```
let filteredAx1 = 0, filteredAy1 = 0, filteredAz1 = 0;
let filteredGx1 = 0, filteredGy1 = 0, filteredGz1 = 0;
let filteredAx2 = 0, filteredAy2 = 0, filteredAz2 = 0;
let filteredGx2 = 0, filteredGy2 = 0, filteredGz2 = 0;
let filteredAdxlAx = 0, filteredAdxlAy = 0, filteredAdxlAz = 0;
```

```
function calculateEulerAngles(ax, ay, az) {
  const pitch = Math.atan2(ay, Math.sqrt(ax * ax + az * az)) * (180 / Math.PI);
  const roll = Math.atan2(-ax, az) * (180 / Math.PI);
  return { pitch, roll };
}
```

```
function updateHand(ax1, ay1, az1, gx1, gy1, gz1, ax2, ay2, az2, gx2, gy2, gz2, adxl_ax,
adxl_ay, adxl_az) {
```

```
  let { pitch: pitch1, roll: roll1 } = calculateEulerAngles(ax1, ay1, az1);
  let { pitch: pitch2, roll: roll2 } = calculateEulerAngles(ax2, ay2, az2);
```

```
  const yaw1 = gz1 * (180 / Math.PI);
  const yaw2 = gz2 * (180 / Math.PI);
```

```
  if (mpu1Enabled) {
    filteredAx1 = alpha * ax1 + (1 - alpha) * filteredAx1;
    filteredAy1 = alpha * ay1 + (1 - alpha) * filteredAy1;
    filteredAz1 = alpha * az1 + (1 - alpha) * filteredAz1;
```

```
    hand1.rotation.x = filteredAx1 * 2;
    hand1.rotation.y = filteredAy1 * 2;
    hand1.rotation.z = filteredAz1 * 2;
```

```
hand1.rotation.x += gx1 * Math.PI / 180;
```

```
hand1.rotation.y += gy1 * Math.PI / 180;
```

```
hand1.rotation.z += gz1 * Math.PI / 180;
```

```
mpu1Angles.innerHTML =
```

```
`<p>MPU6050 #1</p>
```

```
<p>Pitch: ${pitch1.toFixed(2)}°</p>
```

```
<p>Roll: ${roll1.toFixed(2)}°</p>
```

```
<p>Yaw: ${yaw1.toFixed(2)}°</p>`;
```

```
} else {
```

```
mpu1Angles.innerHTML = ""; // Clear the display if MPU1 is disabled
```

```
}
```

```
if (mpu2Enabled) {
```

```
filteredAx2 = alpha * ax2 + (1 - alpha) * filteredAx2;
```

```
filteredAy2 = alpha * ay2 + (1 - alpha) * filteredAy2;
```

```
filteredAz2 = alpha * az2 + (1 - alpha) * filteredAz2;
```

```
hand2.rotation.x = filteredAx2 * 2;
```

```
hand2.rotation.y = filteredAy2 * 2;
```

```
hand2.rotation.z = filteredAz2 * 2;
```

```
hand2.rotation.x += gx2 * Math.PI / 180;
```

```
hand2.rotation.y += gy2 * Math.PI / 180;
```

```
hand2.rotation.z += gz2 * Math.PI / 180;
```

```
mpu2Angles.innerHTML =
```

```

`<p>MPU6050 #2</p>

<p>Pitch: ${pitch2.toFixed(2)}°</p>

<p>Roll: ${roll2.toFixed(2)}°</p>

<p>Yaw: ${yaw2.toFixed(2)}°</p>`;

} else {

    mpu2Angles.innerHTML = ""; // Clear the display if MPU2 is disabled

}

if (adxlEnabled) {

    filteredAdxlAx = alpha * adxl_ax + (1 - alpha) * filteredAdxlAx;

    filteredAdxlAy = alpha * adxl_ay + (1 - alpha) * filteredAdxlAy;

    filteredAdxlAz = alpha * adxl_az + (1 - alpha) * filteredAdxlAz;

    hand3.rotation.x = filteredAdxlAx * 2;

    hand3.rotation.y = filteredAdxlAy * 2;

    hand3.rotation.z = filteredAdxlAz * 2;

}

}

</script>

</body>

</html>

```