**1. The Goal**

Your dataset format is:

timestamp, acc\_x, acc\_y, acc\_z, gyro\_x, gyro\_y, gyro\_z, label

So your ESP32 project needs to:

* Read both **accelerometer** and **gyroscope** values from the MPU6050.
* Timestamp the readings.
* Output the readings in **CSV format** for training datasets.
* Still handle your **photodiode, IR LED, and Bluetooth**.

**2. The Problems You Had**

When you tried to integrate my earlier MPU6050 code with your main.cpp, you saw errors like:

* no matching function for call to 'MPU6050::MPU6050(bool)' → because your old header didn’t declare a constructor that takes bool.
* MPU6050Data was not declared → because the struct wasn’t added to the header.
* no member named 'readData' → because the function wasn’t declared in the header.
* **Ambiguous overload for Wire.requestFrom()** → ESP32 has several versions of this function, so the compiler didn’t know which one to use.

**3. What We Fixed**

**a) Added MPU6050Data**

We introduced a struct to hold **all sensor data in one object**:

struct MPU6050Data {

int16\_t acc\_x, acc\_y, acc\_z;

int16\_t gyro\_x, gyro\_y, gyro\_z;

unsigned long timestamp;

};

Now, instead of separate functions for accel and gyro, you can grab everything in one go.

**b) Added Constructor with Mock Mode**

In mpu6050.h, we added:

MPU6050(bool useMock = false);

This lets you do:

MPU6050 mpu(false); // use real hardware

MPU6050 mpu(true); // mock mode (fake data)

So you can test your program without needing the real sensor connected.

**c) Added readData()**

A new function was created:

void readData(MPU6050Data\* data);

This function:

* Reads **14 bytes** starting from register 0x3B.
* Extracts accel (6 bytes) + temperature (ignored) + gyro (6 bytes).
* Packs it into your MPU6050Data struct with a timestamp.

**d) Fixed Wire.requestFrom() Ambiguity**

On ESP32, Wire.requestFrom() has multiple overloads:

size\_t requestFrom(uint16\_t, size\_t, bool);

uint8\_t requestFrom(uint8\_t, uint8\_t, bool);

uint8\_t requestFrom(int, int, int);

Before, we had:

Wire.requestFrom(MPU6050\_ADDR, (size\_t)14, (bool)true);

That caused ambiguity.

✅ The fix is to use uint8\_t casts:

Wire.requestFrom((uint8\_t)MPU6050\_ADDR, (uint8\_t)14, (bool)true);

This explicitly picks the correct overload for ESP32.

**4. How It All Fits in main.cpp**

Now, in your loop you can do:

MPU6050Data imuData;

mpu.readData(&imuData);

Serial.print(imuData.timestamp); Serial.print(",");

Serial.print(imuData.acc\_x); Serial.print(",");

Serial.print(imuData.acc\_y); Serial.print(",");

Serial.print(imuData.acc\_z); Serial.print(",");

Serial.print(imuData.gyro\_x); Serial.print(",");

Serial.print(imuData.gyro\_y); Serial.print(",");

Serial.print(imuData.gyro\_z);

Serial.println(",0"); // label placeholder

This prints rows exactly like your dataset.

Your photodiode, IR LED, and Bluetooth still work alongside this, no changes needed.

**5. End Result**

* You now have a **clean MPU6050 driver** (.h + .cpp).
* It supports both **mock testing** and **real sensor reads**.
* It avoids the ESP32 overload issue.
* Your main.cpp outputs **proper CSV** in the same format as your dataset.
* You can replace the label column with real annotations later.