Marmara University

Faculty of Engineering

Course Project

EE 2004 – MICROPROCESSOR SYSTEMS

3D Angle Visualization with MPU6050 and LED Matrices

	Dept	Student ID	Student Name
1	EEE	150723841	Ahmet Utku YILMAZ
2	EEE	150720035	Feyzullah OĞUZ

Date: 11.06.2025

CONTENTS

1. Introduction

This report details a project designed to visualize real-time orientation data from an MPU6050 Inertial Measurement Unit (IMU). The system is controlled by an STM32 Nucleo-L053R8 microcontroller. The primary goal is to process sensor data to calculate Euler angles (yaw, pitch, and roll) and display them dynamically on a pair of 8x8 LED matrices.

The two LED matrices are arranged perpendicularly to represent a three-dimensional coordinate space. The vertical axis of both matrices represents the pitch angle (y-axis). The horizontal axis of the first matrix represents the roll angle (x-axis), while the horizontal axis of the second matrix represents the yaw angle (z-axis).

To achieve efficient angle calculations, this project uses the MPU6050's integrated Digital Motion Processor (DMP), which offloads the complex sensor fusion algorithms from the STM32 microcontroller.

2. Hardware and Communication Protocols

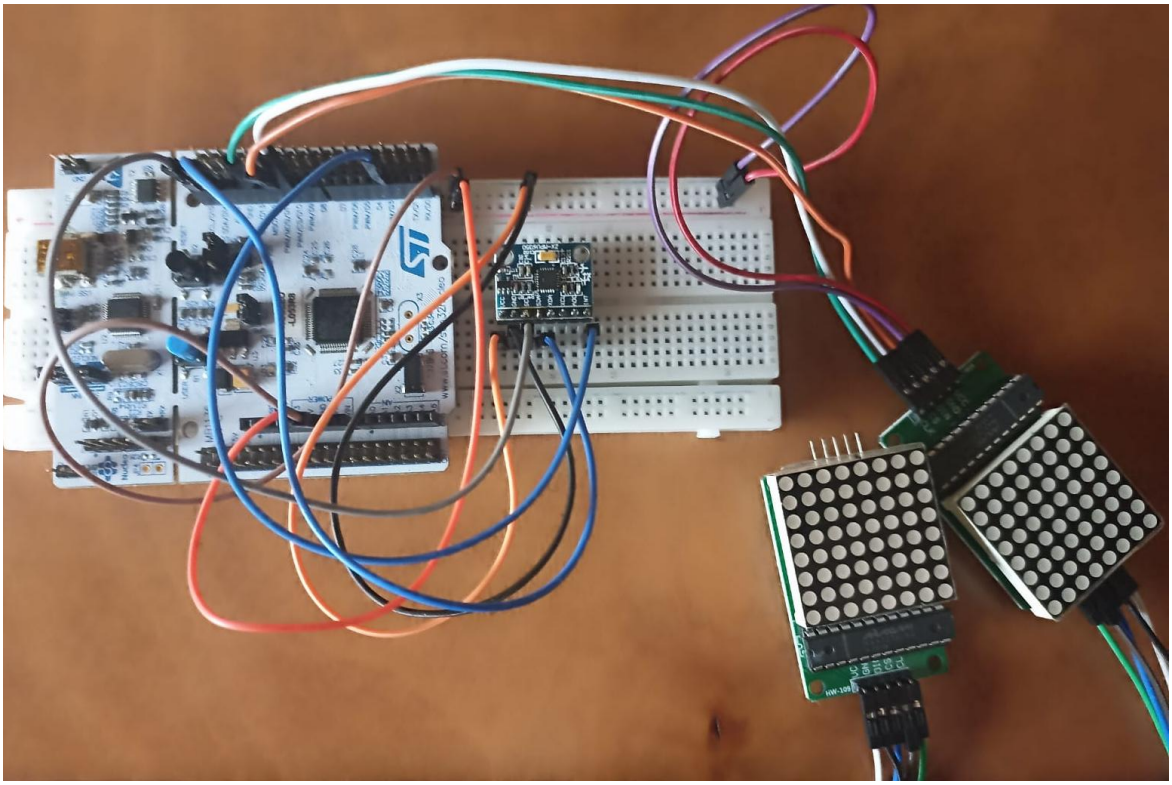

Figure 1 - Connection of the Components

The project integrates a microcontroller, a sensor, and display drivers. I2C and SPI protocol used for these. Component Connections:

- **MPU6050 to STM32:** The MPU6050 sensor communicates with the STM32 board via the I2C protocol. The SCL (Serial Clock) and SDA (Serial Data) lines of the sensor are connected to the corresponding I2C pins on the STM32. Additionally, the MPU6050's interrupt pin (INT) is connected to a digital input pin on the STM32 (D3) to signal when new data is ready for processing.
- **MAX7219 to STM32:** The two MAX7219 display drivers, which control the LED matrices, are connected to the STM32 using the SPI protocol. This requires three pins: a data pin (DATA), a clock pin (CLK), and a load/chip select pin (LOAD/CS). The drivers are daisy-chained, meaning the DOUT (Data Out) pin of the first driver is connected to the DIN (Data In) pin of the second.

Communication Protocols:

- **I2C (Inter-Integrated Circuit):** A two-wire serial bus used for communication between the STM32 (master) and the MPU6050 (slave). It is configured to run at a 400 kHz clock speed for efficient data transfer.
- **SPI (Serial Peripheral Interface):** A synchronous serial communication protocol used for sending display data from the STM32 (master) to the MAX7219 drivers (slaves). The MAX7219 functions as a shift register, which allows multiple devices to be controlled from a single SPI bus. To update the two daisy-chained displays, the LedControl library sends a 32-bit data stream. The first 16 bits are clocked into the DIN of the first driver. As these bits fill its internal register, they are simultaneously pushed out of its DOUT pin and into the DIN of the second driver. After 16 clock cycles, the second driver is full of the data intended for it. The library then sends the next 16 bits, which fill the first driver. Finally, the LOAD/CS pin is toggled, which causes both drivers to latch the data in their registers and update their respective LED displays simultaneously.

3. Code Explanation

The project's firmware is developed in the Arduino IDE, utilizing several libraries to interface with the hardware components. The code is structured to initialize the hardware, process sensor data using the DMP, and update the displays in a non-blocking, interrupt-driven manner.

3.1. Includes and Global Declarations

The code begins by including the necessary libraries:

- **I2Cdev.h and MPU6050_6Axis_MotionApps20.h:** Provide the core functionality for interfacing with the MPU6050 and its DMP.
- **Wire.h:** The standard Arduino library for I2C communication, required for compatibility with the STM32 board.
- **LedControl.h:** A library for controlling the MAX7219 display drivers.

Key global constants and variables are defined:

- SPI and interrupt pin assignments.
- Constants for mapping angle ranges -90 to +90 to the LED matrix grid (0 to 7).
- An MPU6050 object and a LedControl object are instantiated.
- Variables for managing the DMP state (DMPReady, MPUIntStatus, packetSize) and a FIFO buffer (FIFOBuffer) are declared.
- Variables to store orientation data: a Quaternion, a gravity vector, and an array for yaw, pitch, and roll (ypr).

3.2. Interrupt Service Routine (ISR)

```
volatile bool MPUInterrupt = false;

void DMPDataReady() {

  MPUInterrupt = true;

}
```

Figure 2 - DMP Interrupt Control Code

This is a critical part of the design. The DMPDataReady() function is an Interrupt Service Routine (ISR). It is triggered by a RISING edge on the MPU6050's interrupt pin. Its sole purpose is to set the MPUInterrupt flag to true. This flag signals the main loop that new data is available in the MPU's FIFO buffer, enabling an efficient, event-driven architecture.

3.3. setup() Function

The setup() function initializes all hardware and software components:

1. **I2C and Serial:** Wire.begin() starts the I2C bus, and Wire.setClock(400000) sets the communication speed to 400 kHz. Serial.begin() initializes serial communication for debugging.
2. **MPU6050 Initialization:** mpu.initialize() and mpu.testConnection() establish communication with the sensor. The program halts if the connection fails.
3. **DMP Initialization:** mpu.dmpInitialize() loads the DMP firmware onto the MPU6050. If successful, it performs a live calibration of the accelerometer and gyroscope.
4. **DMP and Interrupt Enable:** mpu.setDMPEnabled(true) activates the DMP. attachInterrupt() links the physical interrupt pin (D3) to the DMPDataReady() ISR. This is the final step that makes the DMP operational.
5. **LED Matrix Initialization:** The lc.shutdown(), lc.setIntensity(), and lc.clearDisplay() functions are called for both MAX7219 drivers to wake them from power-saving mode, set a medium brightness, and clear any initial garbage data.

3.4. loop() Function

The main loop is designed to be non-blocking and responsive.

1. **Interrupt Check:** It first checks if the DMP is ready and if an interrupt has occurred (if (!DMPReady || !MPUInterrupt)). If not, it does nothing, preventing unnecessary CPU cycles.
2. **FIFO Overflow Check:** It checks for a FIFO overflow flag from the MPU6050. If an overflow is detected, the FIFO is reset (mpu.resetFIFO()) to prevent reading corrupted data, and the current loop iteration is skipped.
3. **Data Processing:** A while loop runs if there are complete data packets in the MPU's FIFO buffer (mpu.getFIFOCount() >= packet size). Inside the loop:
    - mpu.dmpGetCurrentFIFOPacket(FIFOBuffer) retrieves one packet of data.
    - The packet is then processed using dmpGetQuaternion, dmpGetGravity, and dmpGetYawPitchRoll to calculate the final Euler angles, which are stored in the ypr array.
4. **Display Update:** If new data was processed, the displays are updated:
    - The yaw, pitch, and roll values (in radians) are converted to degrees.
    - Both LED matrices are cleared using lc.clearDisplay().
    - The roll and yaw angles are mapped to a bit pattern using the mapAngle2LedBit() helper function. The pitch angle is mapped to a row index (0-7) using the standard map() function.
    - lc.setRow() is used to light up the single LED corresponding to the current orientation on each matrix. Matrix 0 displays Roll vs. Pitch, and Matrix 1 displays Yaw vs. Pitch.

3.5. Helper Functions

- **printRollPitchYaw():** A simple debugging function to print the angle values to the Serial Monitor. It is commented out in the final code to avoid slowing down the main loop, which could lead to FIFO overflows.
- **mapAngle2LedBit(float angle):** This function maps an angle from -90 to +90 to a column index from 0 to 7. It then converts this index into a byte where only one bit is set, corresponding to the correct column. This is achieved efficiently using a bit-shift operation on the value 128 (B10000000). For example, an index of 2 results in 128 >> 2, which is 32 (B00100000), turning on the third LED in a row.

```
byte mapAngle2LedBit(float angle) {

  long index = map(angle, ANGLE_MIN, ANGLE_MAX, LED_MATRIX_MIN,
LED_MATRIX_MAX);

  return (byte)(128 >> index);

}
```

Figure 3 - Mapping Angle to Bit

4. Conclusion

This project successfully demonstrates the visualization of 3D orientation data on an LED matrix display. The system accurately captures the MPU6050's movement and represents its roll, pitch, and yaw angles in real-time.

A key challenge encountered during development was performance. An initial attempt using a software-based Madgwick sensor fusion filter proved to be too computationally intensive for the STM32's processing speed, which operated around 400-500 Hz. This was significantly lower than the filter's optimal frequency of \(2 \text{ kHz}\), resulting in sluggish and inaccurate angle calculations.

The decisive solution was to switch to the MPU6050's onboard Digital Motion Processor (DMP). By offloading the complex fusion calculations from the STM32 to the sensor itself, the main microcontroller was freed to focus on data retrieval and display updates. This architectural change, combined with an interrupt-driven approach, resolved the performance bottleneck. It also solved the related issue of FIFO buffer overflows, which occurred when the STM32 could not read data fast enough. By removing blocking functions like Serial.print() from the main loop and processing data only when signaled by an interrupt, the system became highly responsive and stable.

For a comprehensive guide, including hardware connections, setup instructions, and the complete source code, please visit the project's GitHub repository:
https://github.com/Empreon/stm32_mpu6050_1088bs