// StdLib

#include <cstdint>

#include <cstring>

#include <limits>

// Mbed and STM32 Drivers

#include "mbed.h"

#include "drivers/LCD\_DISCO\_F429ZI.h"

#include "stm32f4xx.h"

// Local Files

#include "macros.hpp"

#include "dtw\_distance.hpp"

#include "utilities.hpp"

// Gyroscope registers

#define L3GD20\_CTRL\_REG1        0x20

#define L3GD20\_CTRL\_REG4        0x23

#define L3GD20\_OUT\_X\_L          0x28

#define L3GD20\_OUT\_X\_H          0x29

#define L3GD20\_OUT\_Y\_L          0x2A

#define L3GD20\_OUT\_Y\_H          0x2B

#define L3GD20\_OUT\_Z\_L          0x2C

#define L3GD20\_OUT\_Z\_H          0x2D

// Gyroscope scaling factor

#define GYRO\_SCALE\_FACTOR       (17.5f \* 0.017453292519943295769236907684886f / 1000.0f)

// Recording duration in ms

#define RECORDING\_DURATION      2000

// Tolerance for gesture comparison

#define GESTURE\_TOLERANCE     90

// Press duration for recording (in ms)

#define LONG\_PRESS\_DURATION     2000

// Samples per second and total samples

#define SAMPLES\_PER\_SECOND 50

#define NUM\_SAMPLES (RECORDING\_DURATION / 1000 \* SAMPLES\_PER\_SECOND)

// --- Register Addresses and Configuration Values ---

#define CTRL\_REG1 0x20               // Control register 1 address

#define CTRL\_REG1\_CONFIG 0b01'10'1'1'1'1 //0110 1111  // Configuration: ODR=100Hz, Enable X/Y/Z axes, power on

#define CTRL\_REG4 0x23               // Control register 4 address

#define CTRL\_REG4\_CONFIG 0b0'0'01'0'00'0  // Configuration: High-resolution, 2000dps sensitivity

// SPI communication completion flag

#define SPI\_FLAG 1

// Address of the gyroscope's X-axis output lower byte register

#define OUT\_X\_L 0x28

// Scaling factor for converting raw sensor data in dps (deg/s) to angular velocity in rps (rad/s)

// Combines sensitivity scaling and conversion from degrees to radians

#define DEG\_TO\_RAD (17.5f \* 0.0174532925199432957692236907684886f / 1000.0f)

// EventFlags object to synchronize asynchronous SPI transfers

EventFlags flags;

// --- SPI Transfer Callback Function ---

// Called automatically when an SPI transfer completes

void spi\_cb(int event) {

    flags.set(SPI\_FLAG);  // Set the SPI\_FLAG to signal that transfer is complete

}

using namespace std::chrono;

using namespace std;

Timer t;

// LCD

LCD\_DISCO\_F429ZI lcd;

// BUTTON for recording and entering key

DigitalIn button(PA\_0);

// LED for indicating success

DigitalOut led(LED1);

I2C\_HandleTypeDef hi2c;

// Variables to store key gestures

float key\_vals[MAX\_ARRAY\_SIZE][3];

float gyro\_vals[MAX\_ARRAY\_SIZE][3];

bool key\_recorded = false;

// Delay function

void delay\_ms(uint32\_t ms)

{

    HAL\_Delay(ms);

}

// Function to Read Gyroscope Data

void read\_gyro(int row, uint8\_t write\_buf[32], uint8\_t read\_buf[32], SPI \*spi, EventFlags \*flags, float arr[MAX\_ARRAY\_SIZE][3]) {

    uint16\_t raw\_gx, raw\_gy, raw\_gz;

    float gx, gy, gz;

    // Function prototypes

    // Prepare to read gyroscope output starting at OUT\_X\_L

    // - write\_buf[0]: register address with read (0x80) and auto-increment (0x40) bits set

    write\_buf[0] = OUT\_X\_L | 0x80 | 0x40; // Read mode + auto-increment

    // Perform SPI transfer to read 6 bytes (X, Y, Z axis data)

    // - write\_buf[1:6] contains dummy data for clocking

    // - read\_buf[1:6] will store received data

    spi->transfer(write\_buf, 7, read\_buf, 7, spi\_cb);

    flags->wait\_all(SPI\_FLAG);  // Wait until the transfer completes

    // --- Extract and Convert Raw Data ---

    // Combine high and low bytes

    raw\_gx = (((uint16\_t)read\_buf[2]) << 8) | read\_buf[1];

    raw\_gy = (((uint16\_t)read\_buf[4]) << 8) | read\_buf[3];

    raw\_gz = (((uint16\_t)read\_buf[6]) << 8) | read\_buf[5];

    gx = raw\_gx \* DEG\_TO\_RAD;

    gy = raw\_gy \* DEG\_TO\_RAD;

    gz = raw\_gz \* DEG\_TO\_RAD;

    arr[row][0] = gx;

    arr[row][1] = gy;

    arr[row][2] = gz;

    delay\_ms(1000 / SAMPLES\_PER\_SECOND);

}

//Displaying the unlocked message

void display\_unlock\_message(bool success) {

    // Set the font to a larger size

    lcd.SetFont(&Font24);

    // Clear the screen and display the appropriate message

    if (success) {

        lcd.Clear(LCD\_COLOR\_GREEN); // Set background to green

        lcd.DisplayStringAt(0, LINE(7), (uint8\_t \*)"   UNLOCK SUCCESS", CENTER\_MODE);

    } else {

        lcd.Clear(LCD\_COLOR\_RED); // Set background to red

        lcd.DisplayStringAt(0, LINE(7), (uint8\_t \*)"UNLOCK FAILED", CENTER\_MODE);

    }

    // Wait for 2 seconds to allow the user to see the message

    delay\_ms(2000);

    // Reset the font to default (optional)

    lcd.SetFont(NULL);

}

// Main program

int main() {

    // --- SPI Initialization ---

    SPI spi(PF\_9, PF\_8, PF\_7, PC\_1, use\_gpio\_ssel);

    // Buffers for SPI data transfer:

    // - write\_buf: stores data to send to the gyroscope

    // - read\_buf: stores data received from the gyroscope

    uint8\_t write\_buf[32], read\_buf[32];

    // Configure SPI interface:

    // - 8-bit data size

    // - Mode 3 (CPOL = 1, CPHA = 1): idle clock high, data sampled on falling edge

    spi.format(8, 3);

    // Set SPI communication frequency to 1 MHz

    spi.frequency(1'000'000);

    // --- Gyroscope Initialization ---

    // Configure Control Register 1 (CTRL\_REG1)

    // - write\_buf[0]: address of the register to write (CTRL\_REG1)

    // - write\_buf[1]: configuration value to enable gyroscope and axes

    write\_buf[0] = CTRL\_REG1;

    write\_buf[1] = CTRL\_REG1\_CONFIG;

    spi.transfer(write\_buf, 2, read\_buf, 2, spi\_cb);  // Initiate SPI transfer

    flags.wait\_all(SPI\_FLAG);  // Wait until the transfer completes

    // Configure Control Register 4 (CTRL\_REG4)

    // - write\_buf[0]: address of the register to write (CTRL\_REG4)

    // - write\_buf[1]: configuration value to set sensitivity and high-resolution mode

    write\_buf[0] = CTRL\_REG4;

    write\_buf[1] = CTRL\_REG4\_CONFIG;

    spi.transfer(write\_buf, 2, read\_buf, 2, spi\_cb);  // Initiate SPI transfer

    flags.wait\_all(SPI\_FLAG);  // Wait until the transfer completes

    // Initialize the gyroscope and LCD

    lcd.DisplayOn();

    lcd.Clear(LCD\_COLOR\_WHITE);

    // Initialize Key and Entered Gesture value

    memset(key\_vals, 0, sizeof key\_vals);

    memset(gyro\_vals, 0, sizeof gyro\_vals);

    // Main loop

    while (1) {

        float dtw\_distance = std::numeric\_limits<float>::infinity();

        if (button.read() != 1) {

            continue;

        }

        uint32\_t press\_time = HAL\_GetTick(); // Check button press

        while (button.read() == 1) {} // Wait for button release

        // Determine if it was a short press or long press

        if ((HAL\_GetTick() - press\_time) >= LONG\_PRESS\_DURATION) {

            // Long press - Record key gesture

            memset(key\_vals, 0, sizeof key\_vals);

            lcd.Clear(LCD\_COLOR\_BLUE);

            lcd.DisplayStringAt(

                0, LINE(7), (uint8\_t \*)"RECORDING !!", CENTER\_MODE

            );

            delay\_ms(1000);  // Give user a chance to get ready

            for (int i = 0; i < MAX\_ARRAY\_SIZE; i++) {

                read\_gyro(i, write\_buf, read\_buf, &spi, &flags, key\_vals);

                delay\_ms(1000 / SAMPLES\_PER\_SECOND);

            }

            standard\_scaler(key\_vals); //Normalize the data

            key\_recorded = true;

            lcd.Clear(LCD\_COLOR\_WHITE);

            lcd.DisplayStringAt(0, LINE(7), (uint8\_t \*)"RECORDING COMPLETE", CENTER\_MODE);

            delay\_ms(1000);

        } else {

            // Short press - Enter key

            if (!key\_recorded) {

                lcd.Clear(LCD\_COLOR\_RED);

                lcd.DisplayStringAt(0, LINE(7), (uint8\_t \*)"NO KEY RECORDED", CENTER\_MODE);

                delay\_ms(1000);

                continue;

            }

            lcd.Clear(LCD\_COLOR\_BLUE);

            lcd.DisplayStringAt(0, LINE(7), (uint8\_t \*)"ENTER KEY", CENTER\_MODE);

            delay\_ms(1000);  // Starts recording after 1second

            memset(gyro\_vals, 0, sizeof gyro\_vals);

            for (int i = 0; i < MAX\_ARRAY\_SIZE; i++) {

                read\_gyro( i, write\_buf, read\_buf, &spi, &flags,

                    gyro\_vals

                    );

                delay\_ms(1000 / SAMPLES\_PER\_SECOND);

            }

            standard\_scaler(gyro\_vals);

            dtw\_distance = dtw\_distance\_only(key\_vals, MAX\_ARRAY\_SIZE, 3, gyro\_vals, MAX\_ARRAY\_SIZE, 3, 2);

            printf("%f\n\n", dtw\_distance);

            printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

            if (dtw\_distance <= GESTURE\_TOLERANCE)

            {

                // Successful unlock

                display\_unlock\_message(true);

            }

            else

            {

                // Unsuccessful unlock

                display\_unlock\_message(false);

            }

        }

    }

}