#include <Wire.h>

#include <MPU9250\_asukiaaa.h>

MPU9250\_asukiaaa mpu;

#define LED\_COUNT 8

const int ledPins[LED\_COUNT] = {12, 11, 10, 9, 8, 7, 6, 5};

const int buttonPin1 = 3;

const int buttonPin2 = 4;

// Параметры для определения движения

const float accelThreshold = 1.8; // Порог ускорения для активации отображения

const float directionThreshold = 0.8; // Порог для определения направления

bool displayActive = false;

unsigned long lastMotionTime = 0;

const unsigned long motionTimeout = 1000; // Тайм-аут для деактивации отображения (мс)

// Глобальные переменные для отслеживания времени махания

unsigned long lastDirectionChange = 0;

float wavePeriod = 0;

float waveSpeed = 0;

const float maxWavePeriod = 1000; // Максимальный период махания (мс)

const float minColumnDelay = 0.5; // Минимальная задержка столбца (мс)

const float maxColumnDelay = 2.0; // Максимальная задержка столбца (мс)

// Переменные для отслеживания направления движения

bool movingRight = true;

float lastAccelX = 0;

// Глобальные переменные для отслеживания позиции

int currentCharIndex = 0; // Текущий символ в сообщении

int currentColumnIndex = 0; // Текущий столбец в символе

bool isFirstRun = true; // Флаг первого запуска

unsigned long lastDirectionSwitch = 0;

const unsigned long directionSwitchDelay = 50; // Задержка после смены направления (мс)

// Тексты для отображения

const char\* messages[] = {"HELLO", "B", "C", "D", "E"};

int currentMessageIndex = 0;

int messageCount = 5;

// Шрифт 5x8 (5 столбцов, 8 строк)

const byte font[][5] = {

// 'A'

{B01111100, B10010000, B10010000, B10010000, B01111100},

// 'B'

{B11111100, B10100100, B10100100, B10100100, B01011100},

// 'C'

{B01111000, B10000100, B10000100, B10000100, B01001000},

// 'D'

{B11111100, B10000100, B10000100, B01000100, B01111000},

// 'E'

{B11111100, B10100100, B10100100, B10100100, B10000100},

// 'F'

{B11111100, B10010000, B10010000, B10010000, B10000000},

// 'G'

{B01111000, B10000100, B10000100, B10010100, B01011000},

// 'H'

{B11111100, B00100000, B00100000, B00100000, B11111100},

// 'I'

{B10000100, B10000100, B11111100, B10000100, B10000100},

// 'J'

{B10001000, B10000100, B10000100, B10000100, B11111000},

// 'K'

{B11111100, B00100000, B00100000, B00010000, B11001100},

// 'L'

{B11111100, B00000100, B00000100, B00000100, B00001100},

// 'M'

{B01111100, B10000000, B01100000, B10000000, B01111100},

// 'N'

{B11111100, B01000000, B00110000, B00001000, B11111100},

// 'O'

{B01111000, B10000100, B10000100, B10000100, B01111000},

// 'P'

{B11111100, B10010000, B10010000, B10010000, B01100000},

// 'Q'

{B01111000, B10000100, B10000100, B10001100, B01111100},

// 'R'

{B11111100, B10010000, B10010000, B10010000, B01101100},

// 'S'

{B01000100, B10100100, B10100100, B10100100, B10011000},

// 'T'

{B10000000, B10000000, B11111100, B10000000, B10000000},

// 'U'

{B11111000, B00000100, B00000100, B00000100, B11111000},

// 'V'

{B11110000, B00001000, B00000100, B00001000, B11110000},

// 'W'

{B11111100, B00000100, B00111000, B00000100, B11111100},

// 'X'

{B11011100, B00100000, B00100000, B00100000, B11011100},

// 'Y'

{B11000000, B00100000, B00011100, B00100000, B11000000},

// 'Z'

{B10001100, B10010100, B10110100, B10100100, B11000100},

// ' ' (space)

{B00000000, B00000000, B00000000, B00000000, B00000000},

};

// Соотношение букв и их индексов в массиве шрифта

byte getCharIndex(char c) {

if (c >= 'A' && c <= 'Z') {

return c - 'A';

} else if (c == ' ') {

return 26;

} else if (c >= '0' && c <= '9') {

return 27 + (c - '0');

}

return 26; // По умолчанию пробел

}

void setup() {

Serial.begin(9600);

// Инициализация LED пинов

for (int i = 0; i < LED\_COUNT; i++) {

pinMode(ledPins[i], OUTPUT);

digitalWrite(ledPins[i], LOW);

}

// Инициализация кнопок

pinMode(buttonPin1, INPUT\_PULLUP);

pinMode(buttonPin2, INPUT\_PULLUP);

// Инициализация MPU9250

Wire.begin();

mpu.setWire(&Wire);

mpu.beginAccel();

mpu.beginGyro();

// Калибровка датчика

calibrateSensor();

// Тестовая анимация при запуске

LEDsAnimation();

Serial.println("Initialization complete. POV Business Card ready.");

}

void calibrateSensor() {

Serial.println("Calibrating sensor... Keep device still");

// Собираем несколько показаний для установления базовой линии

float sumX = 0, sumY = 0, sumZ = 0;

const int samples = 100;

float values[samples];

for (int i = 0; i < samples; i++) {

mpu.accelUpdate();

values[i] = mpu.accelX();

sumX += values[i];

delay(10);

}

// Проверяем стабильность (дисперсию)

float meanX = sumX / samples;

float variance = 0;

for (int i = 0; i < samples; i++) {

variance += pow(values[i] - meanX, 2);

}

variance /= samples;

if (variance > 0.01) {

Serial.println("Warning: Sensor readings unstable. Please keep device still and retry.");

} else {

lastAccelX = meanX;

Serial.println("Calibration complete");

}

// Визуальное подтверждение

flashLEDs(3);

}

void loop() {

// Обработка нажатий кнопок

handleButtons();

// Получение данных с датчика

mpu.accelUpdate();

float accelX = mpu.accelX();

float accelY = mpu.accelY();

float accelZ = mpu.accelZ();

// Определяем движение по изменению ускорения

float deltaX = abs(accelX - lastAccelX);

// Определяем направление движения

if (deltaX > directionThreshold && (millis() - lastDirectionSwitch > directionSwitchDelay)) {

bool newDirection = (accelX > lastAccelX);

// Если направление поменялось, запомним это

if (movingRight != newDirection) {

movingRight = newDirection;

lastDirectionSwitch = millis();

wavePeriod = millis() - lastDirectionChange;

lastDirectionChange = millis();

wavePeriod = constrain(wavePeriod, 100, maxWavePeriod);

waveSpeed = 1000.0 / wavePeriod;

Serial.print("Wave period: ");

Serial.print(wavePeriod);

Serial.print(" ms, Speed: ");

Serial.println(waveSpeed);

Serial.print("Direction changed: ");

Serial.println(movingRight ? "RIGHT" : "LEFT");

}

lastMotionTime = millis();

if (!displayActive) {

displayActive = true;

isFirstRun = true; // Сбрасываем флаг при новой активации

Serial.println("Display activated");

}

}

// Обновляем предыдущее ускорение

lastAccelX = accelX;

// Проверяем тайм-аут для деактивации отображения

if (displayActive && (millis() - lastMotionTime > motionTimeout)) {

displayActive = false;

Serial.println("Display deactivated due to inactivity");

clearLEDs();

}

// Отображаем сообщение при активном состоянии

if (displayActive) {

float columnDelay = map(wavePeriod, 100, maxWavePeriod, minColumnDelay, maxColumnDelay);

displayCurrentMessage(columnDelay);

}

delay(1); // Уменьшаем задержку для более быстрого отклика

}

// Проверяет, содержит ли строка только прописные буквы

bool isUpperCaseString(const char\* str) {

for (int i = 0; str[i] != '\0'; i++) {

if (str[i] < 'A' || str[i] > 'Z') {

return false;

}

}

return true;

}

void handleButtons() {

// Кнопка 1 - следующее сообщение

if (digitalRead(buttonPin1) == LOW) {

currentMessageIndex = (currentMessageIndex + 1) % messageCount;

Serial.print("Message changed to: ");

Serial.println(messages[currentMessageIndex]);

// Сбрасываем позицию при смене сообщения

isFirstRun = true;

// Визуальное подтверждение смены сообщения

flashLEDs(2);

delay(300);

}

// Кнопка 2 - предыдущее сообщение

if (digitalRead(buttonPin2) == LOW) {

currentMessageIndex = (currentMessageIndex + messageCount - 1) % messageCount;

Serial.print("Message changed to: ");

Serial.println(messages[currentMessageIndex]);

// Сбрасываем позицию при смене сообщения

isFirstRun = true;

// Визуальное подтверждение смены сообщения

flashLEDs(2);

delay(300);

}

}

void displayCurrentMessage(float columnDelay) {

const char\* message = messages[currentMessageIndex];

int messageLength = strlen(message);

if (currentCharIndex >= 0 && currentCharIndex < messageLength) {

char currentChar = message[currentCharIndex];

byte charIndex = getCharIndex(currentChar);

int actualColumn = movingRight ? currentColumnIndex : 4 - currentColumnIndex;

byte columnData = font[charIndex][actualColumn];

for (int led = 0; led < LED\_COUNT; led++) {

digitalWrite(ledPins[led], bitRead(columnData, 7 - led) ? HIGH : LOW);

}

delayMicroseconds((int)(columnDelay \* 1000)); // Задержка в микросекундах

if (movingRight) {

currentColumnIndex++;

if (currentColumnIndex > 4) {

currentColumnIndex = 0;

currentCharIndex++;

clearLEDs();

delayMicroseconds((int)(columnDelay \* 1000));

}

if (currentCharIndex >= messageLength) {

currentCharIndex = 0;

clearLEDs();

delayMicroseconds((int)(columnDelay \* 3000));

}

} else {

currentColumnIndex--;

if (currentColumnIndex < 0) {

currentColumnIndex = 4;

currentCharIndex--;

clearLEDs();

delayMicroseconds((int)(columnDelay \* 1000));

}

if (currentCharIndex < 0) {

currentCharIndex = messageLength - 1;

clearLEDs();

delayMicroseconds((int)(columnDelay \* 3000));

}

}

}

}

void clearLEDs() {

for (int i = 0; i < LED\_COUNT; i++) {

digitalWrite(ledPins[i], LOW);

}

}

void flashLEDs(int times) {

for (int i = 0; i < times; i++) {

for (int j = 0; j < LED\_COUNT; j++) {

digitalWrite(ledPins[j], HIGH);

}

delay(50);

for (int j = 0; j < LED\_COUNT; j++) {

digitalWrite(ledPins[j], LOW);

}

delay(50);

}

}

void LEDsAnimation() {

// Перебегающий огонек

for (int i = 0; i < LED\_COUNT; i++) {

clearLEDs();

digitalWrite(ledPins[i], HIGH);

delay(60);

}

for (int i = LED\_COUNT - 1; i >= 0; i--) {

clearLEDs();

digitalWrite(ledPins[i], HIGH);

delay(60);

}

for (int i = 0; i < LED\_COUNT; i++) {

digitalWrite(ledPins[i], HIGH);

digitalWrite(ledPins[7-i], HIGH);

delay(100);

digitalWrite(ledPins[i], LOW);

digitalWrite(ledPins[7-i], LOW);

}

clearLEDs();

}