Memoria do Proxecto: Pulsador Intelixente

1.- Introdución

O proxecto "Pulsador Intelixente" está enmarcado dentro do programa "Talentos Inclusivos" e foi desenvolvido polos alumnos do segundo curso de FP Básica de Electrónica, xunto co grupo de fp Básica de adultos Retorna Talento. O obxectivo deste proxecto foi crear un pulsador sen contacto que permita a usuarios con parálise cerebral comunicarse facilmente mediante un cambio de cor nun LED e o envío dunha mensaxe a unha pantalla e unha aplicación. O sistema está baseado en tecnoloxía MQTT e utiliza microcontroladores ESP32 e ESP8266 para a súa implementación.

2.- Descrición Xeral do Sistema

O sistema consiste nun pulsador intelixente que detecta a presenza dun usuario a través dun sensor ultrasónico. Cando o usuario está presente durante un tempo determinado programable, o sistema cambia a cor dun LED, amosa unha mensaxe nunha pantalla LED e envía unha mensaxe a unha aplicación mediante o protocolo MQTT. O sistema inclúe dous módulos ESP32 para manexar a pantalla e conectarse á rede Wi-Fi, e un ESP8266 para o sensor ultrasónico. Ambos elementos

van a comunicarse con MQTT a través dun servidor gratuito, neste caso test.mosquitto.org, pero desenvolveuse tamén un servidor local nunha Raspberry pi para maior robustez do sistema.

2.1.- Funcionamento do Sensor

O pulsador intelixente é un dispositivo deseñado para poder avisasr desde a sala ata o comedor e ao movil da coidadora dentro de ASPACE. Está composto por un sensor ultrasónico HC-SR04, un microcontrolador ESP8266, e un LED RGB. Todo o sistema vaise a alimentar mediante un simple alimentador USB tipo C. A continuación, descríbese detalladamente o funcionamento deste sistema:

O sensor ultrasónico HC-SR04 está conectado ao ESP8266 a través dos pinos "trig" e "echo". O pin "trig" é un pin de saída que envía pulsos ultrasónicos, mentres que o pin "echo" é un pin de entrada que recibe o eco destes pulsos, permitindo calcular a distancia ao obxecto detectado.

Inicialmente, o ESP8266 configura os pins para o sensor ultrasónico e o LED RGB, establece a conexión á rede Wi-Fi utilizando as credenciais proporcionadas e configura a conexión co broker MQTT, especificando a dirección do servidor e o porto. O ESP8266 actúa como o núcleo do sistema, controlando tanto o sensor ultrasónico como o LED RGB, e manexando a comunicación a través da rede.

O funcionamento do sensor ultrasónico comeza co envío dun pulso de 10 microsegundos a través do pin "trig", provocando que o sensor emita unha onda sonora a 40 kHz. Esta onda viaxa polo aire e reflíctese ao atopar un obxecto, regresando ao sensor. O pin "echo" recibe o eco da onda reflectida, e o ESP8266 mide o tempo que tarda en regresar. O tempo medido multiplícase por 0.034 e divídese entre 2 para obter a distancia en centímetros, baseado na velocidade do son no aire (aproximadamente 343 metros por segundo).

Unha vez calculada a distancia, o sistema verifica se esta está dentro do rango predefinido (3 a 10 cm). Este rango foi contrastado nas instalacións de ASPACE na segunda visita, e é programable, polo que pode cambiarse cando se desexe. Se a distancia está dentro deste rango, considérase que hai unha presenza diante do sensor. Neste caso, o LED cambia de cor a azul. Se a presenza permanece constante durante 2 segundos (2000 milisegundos), o ESP8266 marca unha bandeira indicando que se debe enviar unha mensaxe de axuda.

Cando a bandeira está marcada, o ESP8266 forma unha mensaxe de texto indicando que un usuario necesita axuda. Esta mensaxe publícase no tópic MQTT predefinido ("TalentosInclusivos"). Despois de enviar a mensaxe, a bandeira réiniciase e o LED cambia de cor a verde durante 5 segundos para indicar que a mensaxe foi enviada con éxito.

Se a distancia medida non está dentro do rango predefinido (3 a 10 cm), o LED cambia de cor a vermello, o temporizador de presenza réiniciase, e a bandeira de publicación desactívase. O ESP8266 verifica constantemente se a conexión co broker MQTT está activa. Se a conexión

pérdese, o sistema tenta restablecela automaticamente, informando do estado de reconexión a través do monitor serie.

O código fonte proporciona unha implementación detallada deste funcionamento. As principais seccións do código inclúen a configuración inicial dos pinos e da conexión Wi-Fi, a medición da distancia mediante pulsos ultrasónicos, o control do LED RGB e a publicación de mensaxes no tópic MQTT cando se detecta unha presenza continua durante 2 segundos, así como a revisión constante da conexión MQTT e o seu restablecemento en caso de desconexión.

O código completo está no apartado 4.1.1.

2.2.-Funcionamento da Pantalla

A pantalla xestiónase cun ESP32, cun programa que ten como obxectivo conectar este microcontrolador a unha rede Wi-Fi, subscribirse a un tópic MQTT, e mostrar as mensaxes recibidas nunha pantalla LED utilizando a biblioteca ESP32-HUB75-MatrixPanel-I2S-DMA. A continuación, descríbese detalladamente o funcionamento deste sistema.

O primeiro paso é establecer a conexión do ESP32 á rede Wi-Fi. Utilízase a biblioteca WiFi.h para manexar esta conexión. As credenciais da rede Wi-Fi (SSID e contrasinal) defínense ao comezo do código. O ESP32 tenta conectarse á rede nun bucle, imprimindo puntos no monitor serie ata que a conexión se establece con éxito. Unha vez conectado, o ESP32 notifica que a conexión Wi-Fi está lista.

Despois de establecer a conexión Wi-Fi, o ESP32 configúrase para interactuar cun servidor MQTT. Utilízase a biblioteca PubSubClient para manexar esta comunicación. A dirección do servidor MQTT e o porto defínense no código. Créase unha instancia de PubSubClient utilizando a conexión de rede Wi-Fi establecida previamente. A función de callback, chamada cando se recibe unha mensaxe, mostra a mensaxe recibida tanto no monitor serie como na pantalla LED.

A pantalla LED configúrase utilizando a biblioteca ESP32-HUB75-MatrixPanel-I2S-DMA. Configúrase un módulo de 64x32 píxeles con 1 panel de lonxitude. A pantalla inicialízase e establécese un brillo do 90%. Tamén se cambia a fonte utilizada pola pantalla a unha fonte máis grande, FreeSansBold12pt7b.

O bucle principal do programa verifica constantemente se a conexión co servidor MQTT está activa. Se a conexión pérdese, tenta restablecela utilizando a función reconnect. Esta función imprime o estado da conexión no monitor serie e tenta reconectar ao servidor MQTT ata que a conexión se restableza con éxito. Unha vez restablecida, o ESP32 volve subscribirse ao tópic MQTT predefinido.

Cando se recibe unha mensaxe no tópic MQTT, a función de callback procesa o payload da mensaxe, converténdoo nunha cadea de caracteres (String). Esta mensaxe móstrase no monitor serie e na pantalla LED mediante a función drawMessage. Esta función configúrase para desprazar a mensaxe de dereita a esquerda a través da pantalla, utilizando unha velocidade de

desprazamento configurable. A posición vertical da mensaxe calcúlase para centrar o texto na pantalla. A función actualiza a pantalla continuamente, desprazando o texto e alternando os buffers de memoria para garantir unha actualización suave.

Ademais, no bucle principal, a función drawRandomDots debuxa puntos aleatorios de cores na pantalla LED. Esta función xera coordenadas aleatorias para os puntos e asigna cores aleatorias a cada punto. A pantalla actualízase despois de debuxar todos os puntos, creando un efecto visual dinámico. Esta actividade mantén a pantalla activa e interesante cando non se están recibindo mensaxes MQTT.

O Código completo está no apartado 4.2.2.

3.- Interacción con Aspace

3.1.- Visita ao CIFP Universidade Laboral

Durante o desenvolvemento do proxecto, recibimos a visita de membros de Aspace (Asociación de Atención á Parálise Cerebral) ao noso centro, CIFP Universidade Laboral. Durante esta visita, presentamos a nosa idea do proxecto, pero o máis importante e que persoal e usuarias de ASPACE nos explicaron de primeira man qué e a parálise cerebral, qué implica e sobre todo cómo e a súa vida.

3.2.-Visitas a Aspace

Realizamos dúas unha visitas ás instalacións de ASPACE para comprender mellor as necesidades dos seus usuarios e usuarias. Esta experiencia permitiunos adaptar o noso proxecto para que fora máis útil e fácil de usar para persoas con esta discapacidade. Na primeira vista participamos nun programa de Radio que facían nas súas instalacións, cos dous grupos que participaron por parte do CIFP Universidade Laboral no programa.

Nunha segunda vista, xa fumos a probar o dispositivo, e dedir in situ as posibilidades da conectividade wifi, necesaria para o funcionamento do proxecto. Par tamén houbo, aparte da actividade técnica moita iteración con usuarios e usuarias.

3.3.- Exposición Final

Na exposición final, celebrada no centro Agora da Coruña, asistiu o grupo de retorna talento ao completo, e foi un acto moi interesante, onde os propios rapaces, xunto con usuarios e usuarias de ASPACE mostraron o proxecto e a súa funcionalidade as autoridades que visitaron a exposición.


TALENTOS INCLUSIVOS

4.- Materiais

4.1.- Programas para microcontroladores

4.1.1.- programa para a xestión do pulsador

```
//Usar placa LOLIN(WEMOS) D1 ESP-WROOM-02 para subir programa.
#include <ESP8266WiFi.h>
#include <PubSubClient.h>    //Librería counicaciones MQTT
//----------------------Definición de los pines del sensor ultrasonidos HC-SR04
#define trigPin    D4    // GPIO4 en el ESP8266
#define echoPin    D3 // GPIO0 en el ESP8266
#define ledPinR      D7 // GPIO13 en el ESP8266
#define ledPinG      D6 /
#define ledPinB      D5
//-------------------Variables para almacenar la distancia y el tiempo de duración del obstáculo
long duration;
int distance;

//-----------------Variables para llevar un registro del tiempo que el LED ha estado activo
unsigned long ledActiveStartTime = 0;
const unsigned long requiredLedActiveTime = 2000; // 2 segundos en milisegundos
// Flag para indicar si se debe publicar en el topic MQTT
bool shouldPublish = false;
// ----------Configuración WiFi
const char* ssid = "    ";          //Introduce aquí el nombre de tu red WiFi
const char* password ="    "; // Introduce aquí la contraseña de tu WiFi
// -----------Configuración el broker MQTT
```

```
const char* mqtt_server = "test.mosquitto.org"; // Dirección IP o nombre de dominio del broker
const int mqtt_port = 1883; //Puerto del Broker
const char* mqtt_topic = "TalentosInclusivos"; // Topic en el que se va a publicar
// Crea una instancia de PubSubClient
WiFiClient espClient;
PubSubClient client(espClient);
void setup() {

    pinMode(trigPin, OUTPUT);      // Envía la señal de ultrasonido
    pinMode(echoPin, INPUT);      //    Recive la señal de ultrasonido
    // Definimos los pins
    pinMode(2, OUTPUT); // Red: D4
    pinMode(4, OUTPUT); // Green: D2
    pinMode(5, OUTPUT); // Blue: D1

    Serial.begin(9600);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Conectando a WiFi...");
    }
    Serial.println("Conexión WiFi establecida");
    // Configura el servidor MQTT
    client.setServer(mqtt_server,mqtt_port);
}
```

```
void color (int R, int G, int B) {//Control colres del led RGB
    analogWrite(D7, R);
    analogWrite(D6, G);
    analogWrite(D5, B);
}
void loop() {
    if (!client.connected()) {    //mientras no estemos conectados al broker...
        reconnect();                    //...llamamos a la función que NOS INFORMA de ello por el monitor serie
    }
    client.loop(); //-------función que mantiene la conexión MQTT con el servidor activa y reconecta en caso de caída. Está en la librería PubSubClient.
    // ------------Envía un pulso ultrasónico para activar el sensor
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = duration * 0.034 / 2;
    // -----------Verifica si la distancia está dentro del rango deseado
    if (distance >= 3 && distance <= 10) {
        // Si la distancia está dentro del rango, activa el LED
        color(1023, 1023, 0); //ponemos led en azul
        if (millis() - ledActiveStartTime >= requiredLedActiveTime) {
            shouldPublish = true; // Si el LED ha estado activo durante 2 segundos seguidos, establece la bandera para publicar
```

```
        }
    } else {
        // Si la distancia no está dentro del rango, apaga el LED y reinicia el registro del tiempo
        color(0, 1023, 1023); //ponemos led en rojo
        ledActiveStartTime = millis();
        shouldPublish = false; // Reinicia la bandera para publicar
    }
    // Si la bandera para publicar está establecida, publica en el topic MQTT
    if (shouldPublish) {
        char mensaje1[50];
      //sprintf(mensaje1, "La usuaria X necesita ayuda - Hora: %02d:%02d:%02d", hour(), minute(), second()); //Sin módulo reloj no tiene sentido
        sprintf(mensaje1, "Ayuda en el Taller 2");
        //------------------------Publica Topic
        client.publish(mqtt_topic, mensaje1);
        shouldPublish = false; // Reinicia la bandera después de la publicación
        color(1023, 0, 1023); //ponemos led en verde
        delay(5000);
    }
    delay(500); // Pequeña pausa para estabilidad
}

//-----Estra función solo informa de los esfuerzos por restablecer conexión MQTT por el puerto serie
void reconnect() {
    while (!client.connected()) {
        Serial.print("Conectando al broker MQTT...");
```

```
        if (client.connect("ESP8266Client")) {
            Serial.println("Conectado al broker");

        } else {
            Serial.print("Error al conectar (rc=");
            Serial.print(client.state());
            Serial.println("). Intentando de nuevo en 5 segundos...");
            delay(5000);
        }
    }
}
```

4.1.2.- programa para a xestión da pantalla

```
//Placa: ESP32-Wroom-DA Module
//Conectar el ESP32 a una red Wi-Fi, suscribirse a un tópico MQTT y mostrar los mensajes recibidos en
//una pantalla LED utilizando la biblioteca ESP32-HUB75-MatrixPanel-I2S-DMA.
#include <SPI.h>
#include <PubSubClient.h> //
#include <WiFi.h>
#include <ESP32-HUB75-MatrixPanel-I2S-DMA.h>
#include <Fonts/FreeSansBold12pt7b.h> //Fuente más grande
// ----------Configuración WiFi
const char* ssid = "    ";          //Introduce aquí el nombre de tu red WiFi
const char* password ="    "; // Introduce aquí la contraseña de tu WiFi
//----------Configuración MQTT
```

```
const char* MQTT_SERVER = "test.mosquitto.org";

const int MQTT_PORT = 1883;

const char* topic1 = "TalentosInclusivos";

// Preparamos el dispositivo para conectarse a una red Wi-Fi y usar esa conexión para interactuar con un servidor MQTT:

WiFiClient wifiClient;                    // Crea una instancia de la clase WiFiClient. Proporciona funciones para manejar conexiones de red TCP/IP.

PubSubClient mqttClient(wifiClient); // Crea una instancia de la clase PubSubClient llamada mqttClient que utilizará la conexión de red

                                                    // gestionada por wifiClient para comunicarse con el servidor MQTT.

MatrixPanel_I2S_DMA *dma_display = nullptr; // Inicializar el puntero a nullptr, se asegura de que el puntero no contenga un valor indeterminado.

void callback(char* topic1, byte* payload, unsigned int length)

{

   payload[length] = '\0';                    // Asegura que la cadena de caracteres finalice correctamente

   String message = String((char*)payload); // Convierte el payload a una String

   // Muestra el mensaje por el puerto serie

   Serial.print("Message arrived [");

   Serial.print(topic1);

   Serial.print("] ");

   Serial.println(message);

   // Muestra el mensaje en la pantalla LED

   drawMessage(message);

}

void reconnect() {

   // Loop que nos INFORMA del estado de la conexión hasta que nos reconectemos
```

```
    while (!mqttClient.connected()) {
        Serial.print("Intentando conexión MQTT...");
            if (mqttClient.connect("arduinoClient")) {
            Serial.println("conectado");
            // Una vez conectado, suscríbete a los topics
            mqttClient.subscribe(topic1);
          } else {
            Serial.print("fallo al conectar al broker MQTT, rc=");
            Serial.print(mqttClient.state());
            Serial.println(" volviéndolo a intentar en 5 segundos");
            // Espera 5 segundos antes de volver a intentar
            delay(5000);
        }
    }
}
void setup() {
    Serial.begin(115200);

    // Inicializa el WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi conectado.");
```

```
    // Inicializa la pantalla LED
    HUB75_I2S_CFG mxconfig(
        64,      // ancho del módulo
        32,      // alto del módulo
        1         // longitud de la cadena
    );
    dma_display = new MatrixPanel_I2S_DMA(mxconfig);
    dma_display->begin();
    dma_display->setBrightness8(90); // El brillo tiene un rango de 0-255
    // Cambiar la fuente (ejemplo con una fuente diferente)
    dma_display->setFont(&FreeSansBold12pt7b); // Cambia a una fuente más grande
    // Configura el cliente MQTT
    mqttClient.setServer(MQTT_SERVER, MQTT_PORT);
    mqttClient.setCallback(callback);

    // Permite que el hardware se ajuste
    delay(1500);
}

void loop()
{
    // Reconecta si es necesario
    if (!mqttClient.connected()) {
        reconnect(); // Esta función SOLO INFORMA por monitor serie; NO RECONECTA.
        // LA IMPORTANTE ES mqttClient.loop()
```

```
    }
    mqttClient.loop(); // El cliente MQTT procesa cualquier mensaje entrante y se asegura de que la conexión MQTT se mantenga activa.
                                    // También maneja el envío de los "pings" periódicos al servidor MQTT para mantener viva la conexión.


//-------------------------------------------------------------------------------------------
    // Dibuja puntos aleatorios de colores en la pantalla
    drawRandomDots();
}
void drawRandomDots() {
    int numDots = 100; // Número de puntos aleatorios a dibujar

    for (int i = 0; i < numDots; i++) {
        int x = random(0, dma_display->width());
        int y = random(0, dma_display->height());
        uint16_t color = dma_display->color444(random(16), random(16), random(16));
        dma_display->drawPixel(x, y, color);
    }
    dma_display->flipDMABuffer(); // Alterna los búferes para actualizar la pantalla
    delay(100); // Espera antes de dibujar los siguientes puntos
}
//-------------------------------------------------------------------------------------------------------
void drawMessage(String message) {
    // Configura la velocidad de desplazamiento
    int scrollSpeed = 1; // Ajusta según sea necesario
```

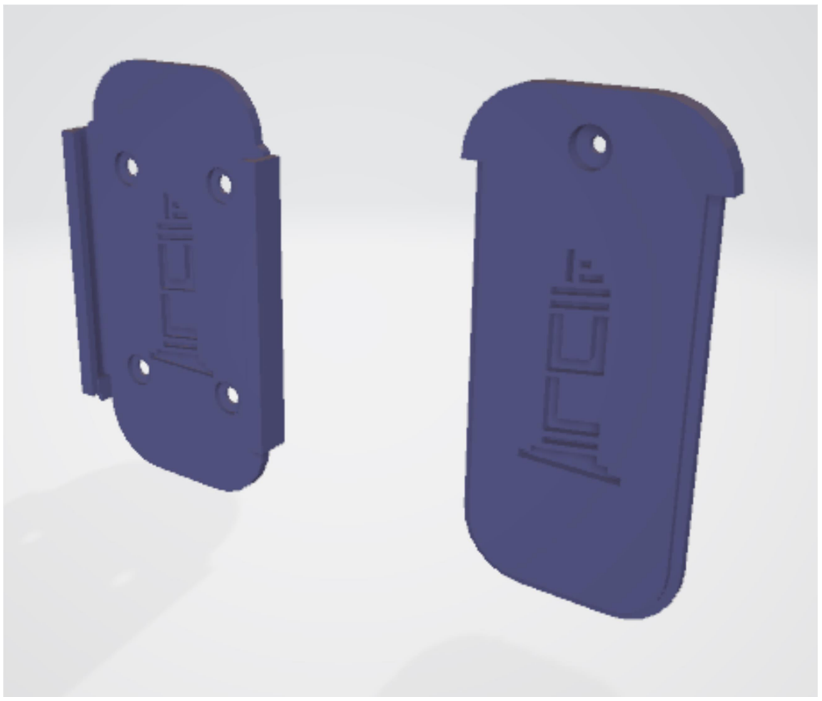
```
    // Ancho de la pantalla
  int screenWidth = dma_display->width();
    // Calcula la duración del desplazamiento basado en la velocidad y el tamaño del texto
  int messageWidth = message.length() * 12; // Ajusta este valor si la fuente es diferente
  int scrollDuration = (screenWidth + messageWidth) / scrollSpeed; // Duración suficiente para desplazarse fuera de la pantalla
  // Calcula la posición vertical media
  int yPos = (dma_display->height() - 12) / 2 + 12; // Ajusta este valor según la altura de la fuente
    // Posición inicial del texto
  int xPos = screenWidth;
  // Itera sobre el tiempo de desplazamiento
  for (int i = 0; i < scrollDuration; i++) {
    // Borra la pantalla
    dma_display->fillScreen(dma_display->color444(0, 0, 0));
        // Configura la posición del texto en la pantalla
    dma_display->setCursor(xPos, yPos);
    dma_display->setTextColor(dma_display->color444(15, 15, 15));
    dma_display->print(message);
        // Mueve el texto hacia la izquierda
    xPos -= scrollSpeed;
        // Verifica si el texto ha llegado al borde izquierdo de la pantalla
    if (xPos + messageWidth < 0) {
      // Si ha llegado al borde izquierdo, reinicia la posición del texto al borde derecho
      xPos = screenWidth;
    }
        // Actualiza la pantalla
```

```
        dma_display->flipDMABuffer(); //Alterna los dos buffer de memoria para que la pantalla se actualice suavemente (doble buffer)

        // Espera un breve período de tiempo antes de la próxima iteración

        delay(50); // Ajusta según sea necesario para la velocidad de desplazamiento deseada

    }

    dma_display->fillScreen(dma_display->color444(0, 0, 0)); //Limpiamos la pantalla

}
```

4.2. Deseños 3D

Para complementar o proxecto, creamos varios deseños en 3D utilizando ferramentas de modelado dixital. Estes deseños foron impresos en 3D e utilizados para aloxar os compoñentes electrónicos do pulsador intelixente, garantindo un aspecto profesional e funcionalidade ergonómica.

Os dous deseños que se crearon foron o soporte para albergar o ESP8266 co sensor ultrasónico e o soporte para poder instalar éste na parede.



5.- Conclusións

O proxecto "Pulsador Intelixente" foi un éxito tanto a nivel técnico como social. Aprendemos a traballar con tecnoloxías como MQTT e ESP32, e adquirimos habilidades en deseño de sistemas electrónicos e integración de sensores. Ademais, a colaboración con ASPACE permitiunos desenvolver un dispositivo que creemos que vai ser útil para as usuarias e usuarios, ainda que está pendente na data da redacción desta memoria a súa entrega e instalación definitiva.

A nivel didáctico resultou moi interesante, xa que está cláramente na liña de proxecto de educación-servizo, que se está a implementar no ensino de FP Básica no CIFP Universidade Laboral.

Desde este centro, agradecemos moito a colaboración tanto de ASPACE como do CITIC da Universidade da Coruña, e pensamos que a experiencia foi moi enriquecedora desde varios puntos de vista e claramente positiva, polo que esperamos participar en vindeiras edicións.