

Universidad Internacional SEK

Ingeniería de Software

4to Semestre

Proyecto Final:

**"Sistema de Detección, Monitoreo y Alerta de
Fugas de Gases y llamas con ESP32 e
integración IoT con Ubidots y Telegram"**

Programación de Sistemas Embebidos

Estudiante: Bryan Enrique Garay Benavidez

Docente: MSc. Bustamante Villagomez Diego Fernando

Fecha: 14 de febrero de 2024

Contenido

1. Objetivo del Proyecto	3
2. Marco Teórico	4
2.1. El microcontrolador ESP32	4
2.2. Sensores de gas MQ2, de flama LM393 y de temperatura DHT11	5
2.3. Ubidots como plataforma para el IoT	6
2.4. Telegram Bot para alarmas.....	7
3. Planificación y materiales.....	8
4. Herramientas Digitales	9
5. Diseño del proyecto y Arquitectura.....	10
6. Circuito y Flujograma	12
7. Programación	14
8. Resultados	20
9. Conclusión	22
10. Referencias	23

"Sistema de Detección, Monitoreo y Alerta de Fugas de Gases y llamas con ESP32 e integración IoT con Ubidots y Telegram"

1. Objetivo del Proyecto

El objetivo principal de este proyecto es desarrollar un prototipo funcional para la detección, monitoreo y alerta de fugas de gases y llamas utilizando microcontroladores, específicamente el ESP32. Se busca integrar tecnologías de Internet de las cosas (IoT) para ofrecer una solución integral, moderna y que atienda problemas que pueden ser comunes en un ambiente doméstico. Los objetivos secundarios identificados son:

1. **Detección y Monitoreo Preciso:** Implementar sensores de buena precisión, como el sensor de gas MQ2 y el sensor de llama, para detectar de manera confiable la presencia de gases peligrosos y posibles incendios en el entorno, implicando una correcta calibración de acuerdo con la necesidad.
2. **Alertas en Tiempo Real:** Utilizar la conectividad WiFi del ESP32 para enviar alertas en tiempo real a través de la plataforma de mensajería Telegram. Esto permite a los usuarios recibir notificaciones instantáneas sobre situaciones de riesgo, incluso cuando no están físicamente presentes en el lugar.
3. **Integración con Plataforma IoT:** Integrar el sistema con la plataforma Ubidots para el monitoreo remoto, recepción de datos, control y visualización. Esto proporciona a los usuarios una interfaz intuitiva para supervisar el estado de los sensores, sus resultados y controlar las alarmas.

2. Marco Teórico

Los proyectos de sistemas embebidos tienen un impacto significativo en la vida diaria. Estos sistemas ofrecen soluciones inteligentes que mejoran la seguridad, la eficiencia y la comodidad. Estas aplicaciones se pueden encontrar en casi cualquier hogar moderno abarcando sistemas de videovigilancia, control de humedad y temperatura, seguros de puertas automáticos, control de luces, etc. Los sistemas embebidos optimizan las tareas domésticas y hacen que las viviendas sean más seguras y cómodas para sus habitantes.

Dentro de las aplicaciones domésticas las fugas de gases o incendios son también de relevancia. En la publicación del diario (El Universo, 2023) se reporta que: “En el año 2023 se han producido 9 deflagraciones por fugas de gas en la capital, mientras que en 2022 hubo 22. En tanto que se atendieron 466 fugas de gas en lo que va del año versus las 594 del año pasado.” Enfocado en esta problemática surge la necesidad de construir un prototipo que permita una alerta temprana ante fugas de gas o un posible

Para lograr la construcción de un sistema de estas características hay que comprender los componentes claves del sistema que se detallan a continuación.

2.1. El microcontrolador ESP32

El ESP32 es un microcontrolador de bajo costo y alto rendimiento desarrollado por Espressif Systems. Integra capacidades de conectividad Wi-Fi y Bluetooth, lo que lo hace ideal para aplicaciones de Internet de las cosas (IoT) y proyectos embebidos.

De acuerdo con lo leído en (Babuich, et al., 2019), este microcontrolador ha ganado popularidad en la comunidad de desarrollo debido a su potencia, versatilidad y bajo costo. Permite a los desarrolladores crear sistemas IoT avanzados con capacidades de conectividad inalámbrica y procesamiento de datos.

Para **configurar** el ESP32, se puede utilizar el IDE de **Arduino** junto con la instalación de la biblioteca **ESP32** en el IDE. Luego, se selecciona el tipo de placa ESP32 y se configuran las librerías necesarias y se carga el código escrito.

Ventajas de la ESP32:

- Conectividad Wi-Fi y Bluetooth integrada para comunicación inalámbrica.
- Potente procesador de doble núcleo que permite ejecutar aplicaciones complejas.
- Amplia gama de periféricos y interfaces disponibles para la conexión de sensores y actuadores.
- Bajo consumo de energía en modo de reposo, lo que lo hace adecuado para dispositivos alimentados por batería.

2.2. Sensores de gas MQ2, de flama LM393 y de temperatura DHT11

- El sensor de gas MQ2 es un dispositivo que detecta la presencia de varios gases como metano, propano, alcohol y humo.
- El sensor de flama LM393 es un sensor que detecta la presencia de llamas o fuego.
- El sensor de temperatura DHT11 es un sensor que mide la temperatura y la humedad relativa del ambiente.

Según (Mota, 2014) estos sensores son fundamentales para la detección temprana de peligros como fugas de gas e incendios, lo que contribuye a la seguridad y prevención de accidentes en diversos entornos. Proporcionan datos ambientales clave para el monitoreo y control de condiciones en aplicaciones de automatización, climatización y calidad del aire.

Cómo se configuran: Los sensores se conectan al microcontrolador a través de pines digitales o analógicos, según el tipo de sensor. Se utilizan

bibliotecas específicas para cada sensor en el IDE de Arduino para acceder y procesar los datos de los sensores.

2.3. Ubidots como plataforma para el IoT

En el artículo de (Kanakaraja, et al., 2021) definen a Ubidots como una plataforma en la nube diseñada para la gestión y visualización de datos en proyectos de Internet de las cosas (IoT). Permite la recolección, almacenamiento, análisis y visualización de datos en tiempo real de dispositivos conectados.

Importancia:

- Ubidots simplifica el desarrollo y despliegue de aplicaciones IoT al proporcionar herramientas para la gestión de datos, la creación de paneles de control personalizados y la integración con otras plataformas y servicios.
- Facilita el monitoreo remoto, la toma de decisiones basada en datos y la optimización de procesos en una amplia variedad de aplicaciones industriales, comerciales y residenciales.

Configuración:

Para utilizar Ubidots, se crea una cuenta en la plataforma y se configuran dispositivos y variables para la recopilación de datos.

Se utilizan bibliotecas específicas en el IDE de Arduino para enviar datos desde el microcontrolador a la plataforma Ubidots a través de protocolos de comunicación como MQTT o HTTP.

Cuando ya se llega a un entorno más complejo en donde se necesita suscribirse a varias variables declaradas en Ubidots se puede requerir de una configuración más específica, para lo cual se tomó en consideración el tutorial de (Sepúlveda, 2021) recuperado del siguiente foro de la plataforma oficial de Ubidots:

<https://ubidots.com/community/t/how-to-subscribe-to-multiple-variables-using-esp32-mqtt-and-what-is-the-difference-between-things-ubidots-and-industrial-ubidots/3469/2>

2.4. Telegram Bot para alarmas

Un bot de Telegram es un programa automatizado que interactúa con los usuarios a través de la plataforma de mensajería Telegram. En este contexto, se utiliza para enviar notificaciones y alertas sobre eventos importantes, como alarmas de seguridad.

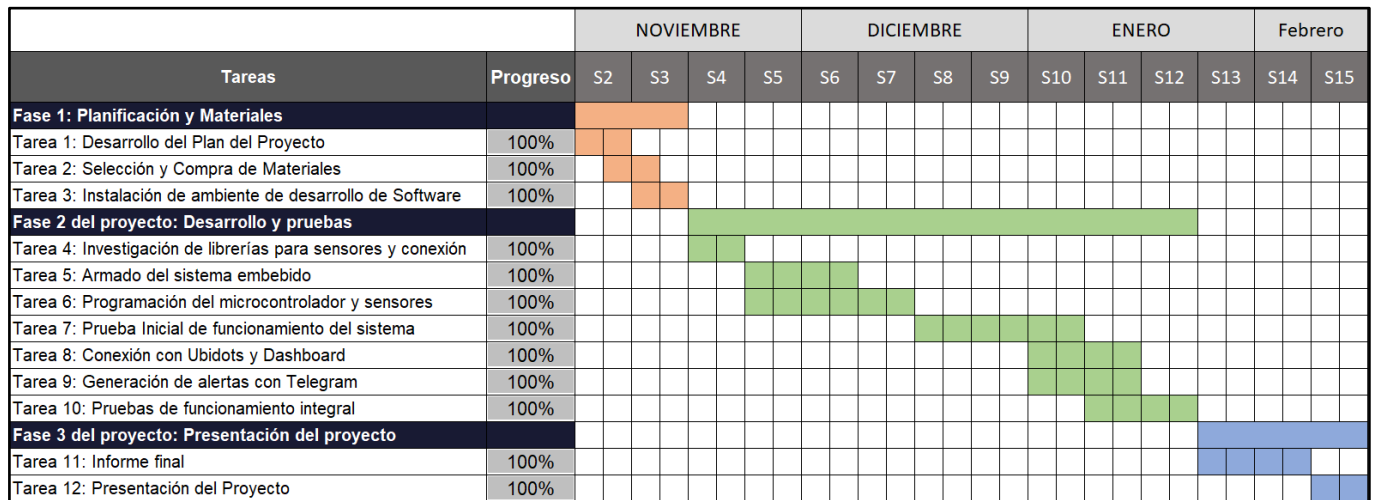
Cómo se configura:

Para poder lograr una conexión exitosa se siguió el tutorial de (Santos, 2020) <https://randomnerdtutorials.com/telegram-esp32-motion-detection-arduino/>. Entre los puntos más relevantes están:







- Se crea un bot en Telegram utilizando el BotFather, el bot oficial de Telegram para la creación y gestión de bots.
- Se obtiene un token de acceso único para el bot, que se utiliza para configurar la comunicación entre el microcontrolador y el bot.


3. Planificación y materiales

La planificación seguida para el desarrollo del proyecto mediante un **Diagrama de Gantt** fue la siguiente:



El listado de materiales físicos, sensores, dispositivos de salida, conexiones y fuente de poder utilizados en el proyecto es:

Material	Imagen referencial
- Microcontrolador ESP32	
- Gas sensor MQ2 (300 - 10000 ppm)	
- Flame sensor LM393	
- Active Buzzer Module (TMB12A05)	
- Led naranja de alto brillo	
- Temperature sensor DHT11	

- Transmisor y receptor de señales infrarrojas	
- Foco de colores controlado por señales infrarrojas y su control respectivo	
- Fuente de alimentación de protoboard 3.3 y 5V.	
- Cables jumper macho/hembra	
- Protoboard	

4. Herramientas Digitales

El código del proyecto fue realizado con el lenguaje de programación C++ en conjunto con las siguientes herramientas de desarrollo.

- Entorno de programación: Arduino IDE
- Repositorio de código: Github
- Servicio de mensajería: Telegram
- Plataforma IoT: Ubidots

Las librerías utilizadas fueron:

- WiFi, WiFiClientSecure
- UniversalTelegramBot
- ArduinoJson
- UbidotsEsp32Mqtt
- DHT
- IRremoteESP8266 y IRsend

5. Diseño del proyecto y Arquitectura

La idea principal del proyecto es crear un prototipo de un sistema automatizado que tenga alarmas automáticas cuando se detecte una concentración alta de gas (principalmente butano) o si se detecta una llama y el registro complementario de temperatura.

Diseño por capas

Para la capa **Física** tenemos entonces 3 sensores: **DHT11**, **MQ2** y **LM393** para flamas. Los dispositivos de salida serían las alarmas físicas incluyendo el **Buzzer** para la alarma sonora, el led y el foco de colores como alarma visual. Todos estos dispositivos físicos se conectan con el microcontrolador ESP32 para enviar y recibir las señales tanto **digitales** como **analógicas** en el caso del MQ2 o el DHT11.

Lo siguiente es la capa **Local** y protocolos de comunicación. En la capa local tenemos a nuestro entorno de Arduino IDE en donde recibimos mediante el Serial Monitor los logs del funcionamiento de la ESP32. Mediante las librerías de comunicación a internet como son **Wifi**, **WifiClientSecure** y el **Broker** para el protocolo de comunicación **MQTT** de Ubidots: **UbidotsEsp32Mqtt**, enviamos las señales en la capa Global.

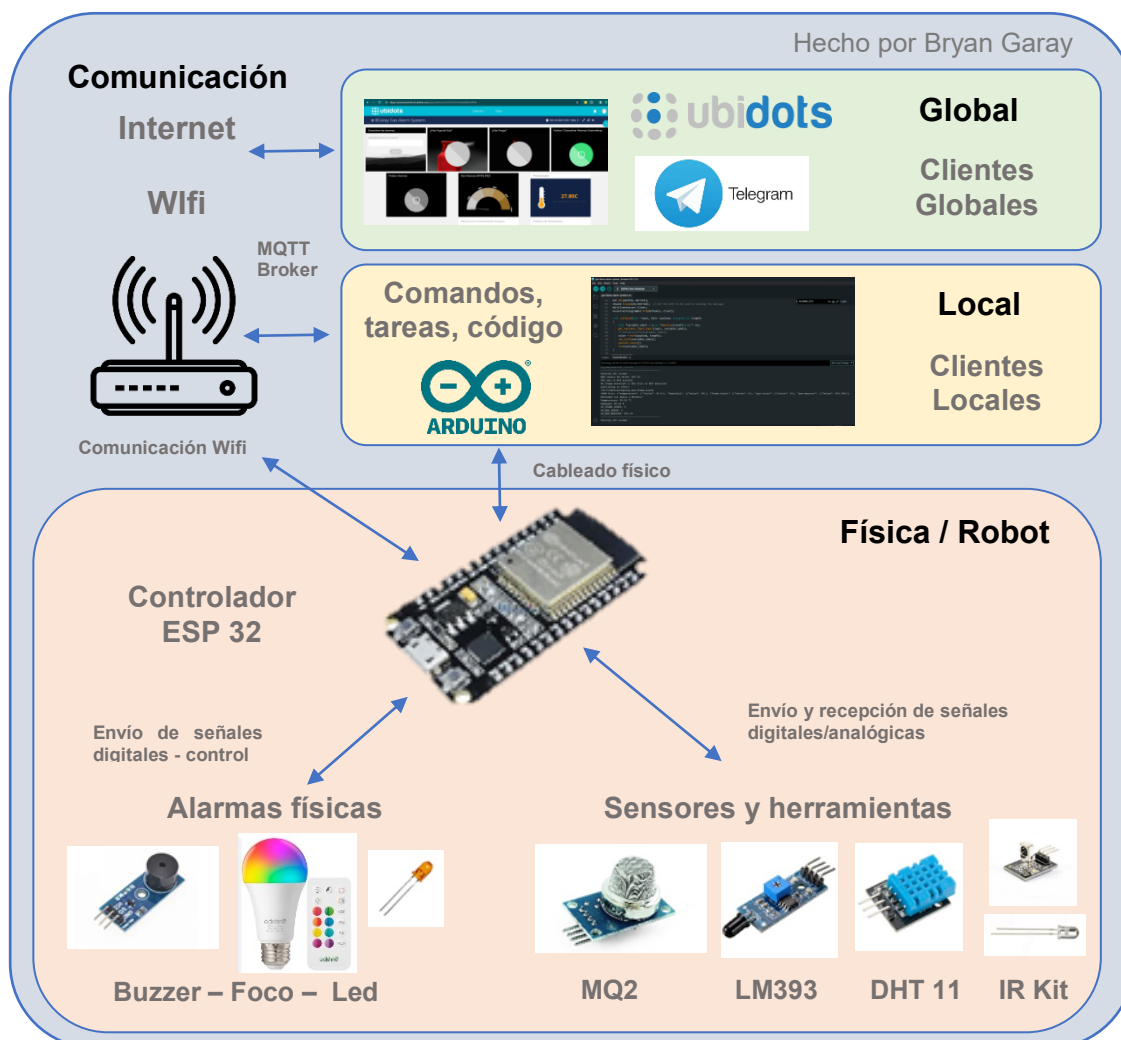
Por último, en la capa **Global** encontramos nuestra plataforma de **IoT** de **Ubidots** en el cual se levantó un Dashboard de monitoreo, visualización de datos y control de diferentes aspectos del sistema. Así mismo, en la parte Global accesible desde cualquier parte tenemos nuestro sistema de mensajería para alertas en línea **Telegram** mediante Bots y su librería de comunicación universal.

Modos de funcionamiento: Con estas capas definidas los modos de funcionamiento del sistema serían los siguientes:

- **Modo Manual:** Las alarmas pueden ser probadas manualmente o activadas/desactivadas a través de un botón en el Dashboard de Ubidots.

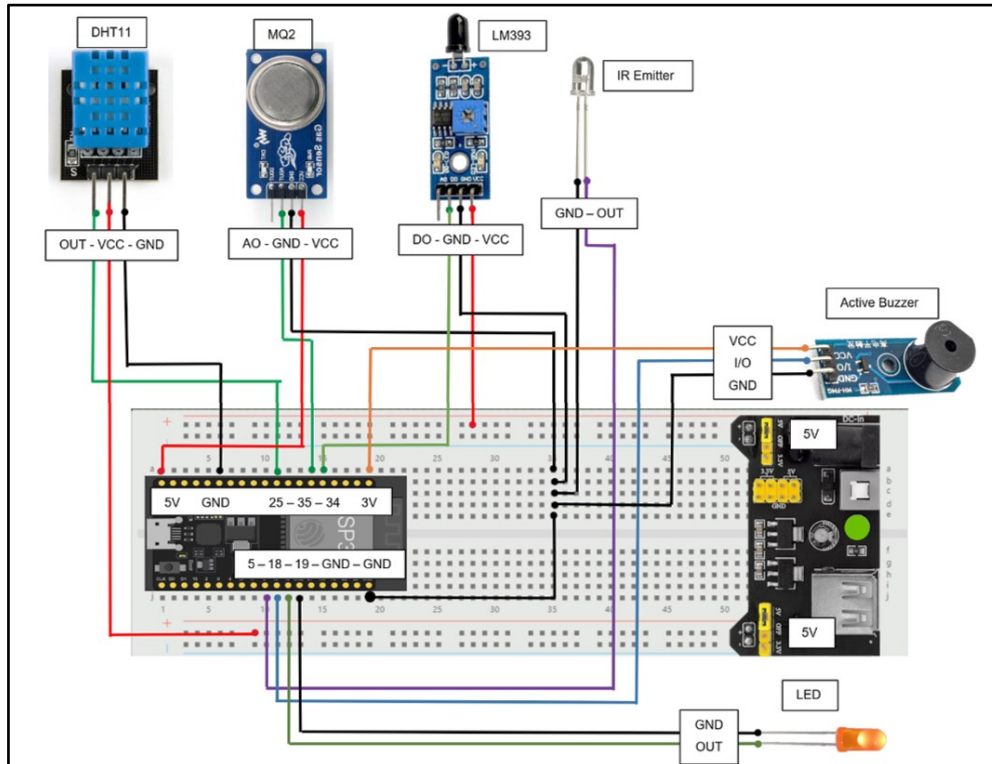
- **Modo Automático:** Si las alarmas se encuentran activas y se detecta una flama o alta concentración de gases, definido el límite permitido en 2000 ppm de acuerdo con la calibración realizada para encendedores domésticos, entonces se activarán las alarmas visuales, sonoras y digitales por Telegram. Si se dejan de detectar alguna de las dos condiciones físicas, se regresa al estado normal de las alarmas que es apagado y se envía un mensaje en Telegram indicando el estado fuera de peligro.

Arquitectura referencial del sistema

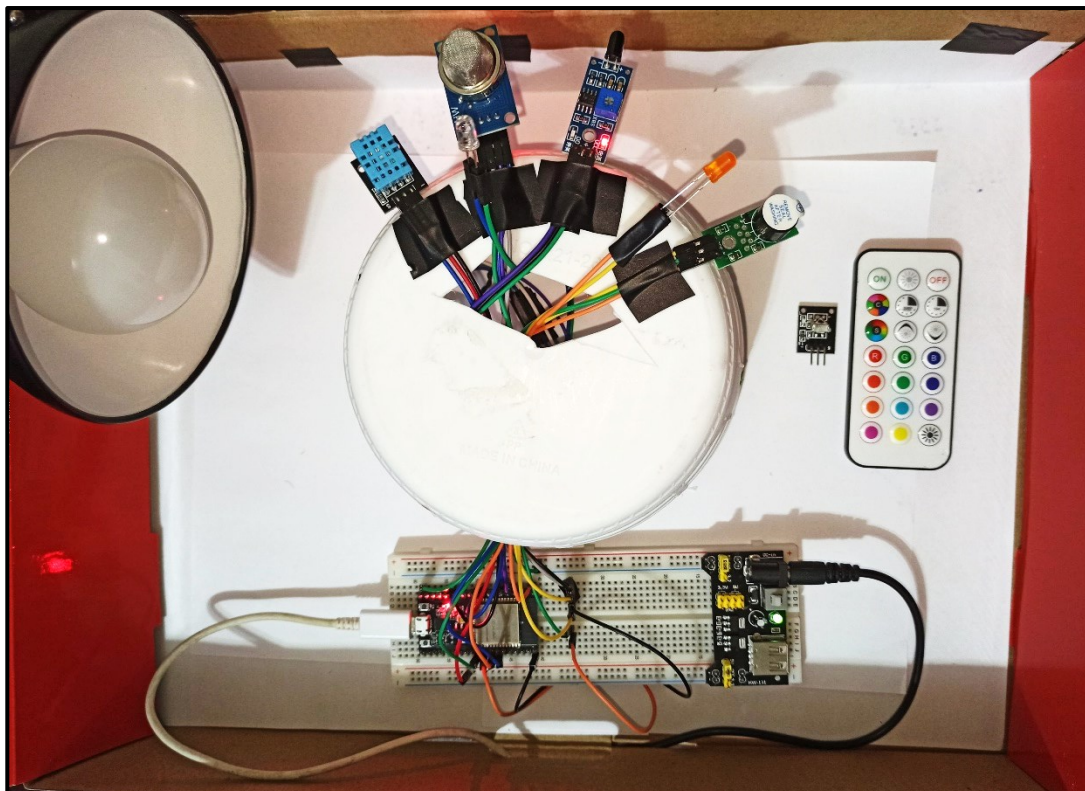


6. Circuito y Flujograma

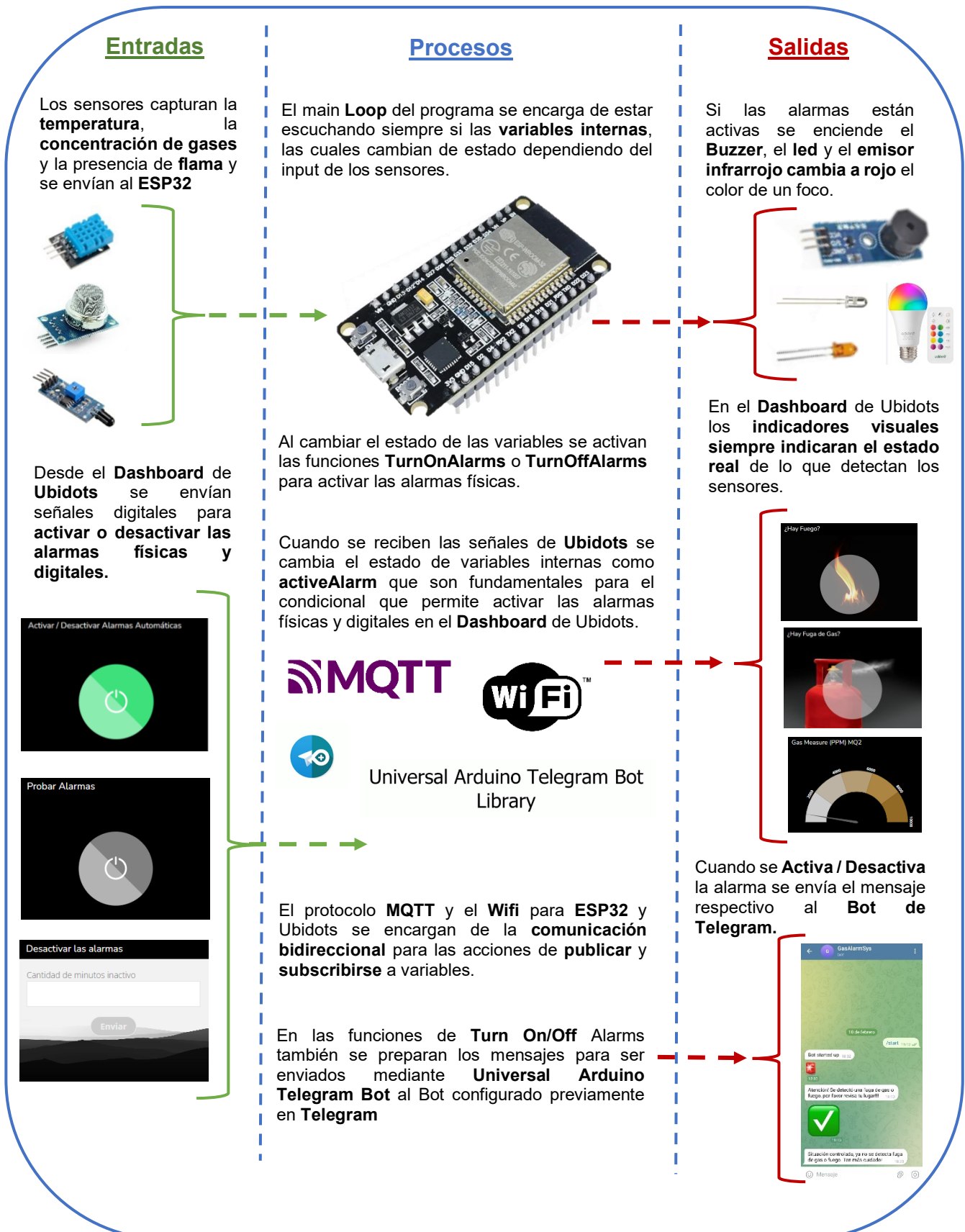
Diagrama de conexión de sensores y alarmas



Evidencia del prototipo armado



Flujo gráfico de procesos



7. Programación

El código completo con su respectivo **README.md** informativo, assets, librerías y códigos complementarios como la forma en la que se puede decodificar un control infrarrojo se encuentran en el siguiente repositorio: <https://github.com/BryanGaray99/esp32-ubidots-gas-leakage-fire-alert-system>

En esta sección se explorarán las partes más relevantes del código del programa. Este fue realizado en el lenguaje de programación **C++** en el entorno de **Arduino IDE**.

- **Librerías:** Esto es lo primero que se ubica en el código ya que bajo las librerías se construye el código, al poder usar los objetos, métodos y demás funciones que proveen las librerías como las de DHT, MQTT o Telegram.

```

1  #include <Arduino.h>
2  #include <WiFi.h>
3  #include <WiFiClientSecure.h>
4  #include <UniversalTelegramBot.h>
5  #include <ArduinoJson.h>
6  #include "UbidotsEsp32Mqtt.h"
7  #include "DHT.h"
8  #include <IRremoteESP8266.h>
9  #include <IRsend.h>

```

- **Credenciales:** Estas son fundamentales para el correcto funcionamiento del programa y las conexiones con los clientes globales, si las contraseñas no coinciden, no se establecerá conexión y el programa puede fallar o solo funcionar para el entorno local.

```

19  /* Telegram credentials */
20  #define BOTtoken ":" // your Bot Token (Get from Botfather)
21  #define CHAT_ID ""
22
23  /* Define the Token provided by the Ubidots platform to make the connection */
24  const char *UBIDOTS_TOKEN = "";
25  /* Define SSID and PASSWORD of your WiFi network */
26  const char *WIFI_SSID = "";
27  const char *WIFI_PASS = "";
28  /* Define the name of your device, which will appear on the Ubidots platform */
29  const char *DEVICE_LABEL = "";

```

- Definición de variables y constantes:** Estas son tanto para el funcionamiento interno del programa como para manejar estados de la aplicación, control de eventos o almacenamiento de información que será procesada en las diferentes partes del programa. Podemos identificar definición de Pines de conexión con la ESP32 que serán constantes, también constantes son los nombres de las variables creadas en Ubidots. Luego tenemos otros valores que pueden cambiar a lo largo de los diferentes estados del sistema como son **activeAlarm**, **TOGGLE_ALARMS_VAR** o **INACTIVITY_TIME** que almacenan los valores recibidos de Ubidots.

```

11  #define AO_MQ2_PIN 35
12  #define DO_FLAME_PIN 34
13  #define DHTPIN 25
14  #define IR_EMITTER 5
15  #define BUZZER_PIN 18
16  #define ALARM_LED 19
17  #define DHTTYPE DHT11
  
```

```

31  /* Define the variables to be measured and published on the Ubidots platform */
32  const char *DO_MQ2 = "gas-state";
33  const char *AO_MQ2 = "gas-measure";
34  const char *DO_FLAME = "flame-state";
35  const char *TEMP_LABEL = "temperature";
36  const char *HUMIDITY_LABEL = "humidity";
37  const char *ALARMS_INACTIVITY_TIME = "alarms-inactivity-time";
38  const char *TOGGLE_ALARMS = "toggle-alarm";
39  const char *TEST_ALARMS = "test-alarms";
40
41  const int PUBLISH_FREQUENCY = 1000;
42  const int gasThreshold = 2000;
43
44  const uint8_t NUMBER_OF_VARIABLES = 3; // Number of variables to subscribe to
45  char *variable_labels[NUMBER_OF_VARIABLES] = {"alarms-inactivity-time", "toggle-alarm", "test-alarms"};
46  float value; // To store incoming value
47  uint8_t variable; // To keep track of the state machine
  
```

```

49  int alarmsActive = 1;
50  int manualAlarm = 0;
51  static unsigned long CURRENT_INACTIVITY_TIME = 0;
52  static unsigned long INACTIVITY_TIME = 0; // Variable to store inactivity time
53  int TOGGLE_ALARMS_VAR; // Variable to toggle alarms
54  int TEST_ALARMS_VAR;
55  String colorLight = "";
56  const int ERROR_VALUE = 65535; // Error value
  
```

- **Funciones auxiliares:** Estas permiten acciones como subscripción al valor de las variables recibidas de Ubidots o la publicación de valores. Estas son **void Callback**, **get_variable_label_topic**, **btotf**, **set_state**. Estas básicamente no necesitan ser modificadas.
- **Switch de ejecución de casos:** Definido por la función **void execute_cases()**, es parte central de la ejecución al responder de forma automática a la variable y su valor recibido desde Ubidots, en el caso del programa son 3: **"alarms-inactivity-time"**, **"toggle-alarm"**, **"test-alarms"**. En el orden que las definamos en:

char *variable_labels[NUMBER_OF_VARIABLES]

Será la ejecución de casos, es decir que si desde Ubidots se envía un valor para la variable con el **label toggle-alarm**, entonces se ejecutará el caso 1, por su posición en el array.

```
gas-flame-alarm-system.ino
128 // Function with switch case to determine which variable changed and assign the value
129 void execute_cases()
130 {
131     switch (variable)
132     {
133     case 0:
134         // Serial.println("Tiempo de inactividad recibido:" + String(value));
135         INACTIVITY_TIME = value * 60 * 1000;
136         CURRENT_INACTIVITY_TIME = millis();
137         Serial.println("Inactivity time started:" + String(CURRENT_INACTIVITY_TIME));
138         alarmsActive = 0;
139         break;
140     case 1:
141         // Serial.println("value:" + String(value));
142         TOGGLE_ALARMS_VAR = value;
143         if (TOGGLE_ALARMS_VAR == 1.0) // User activates alarms
144         {
145             alarmsActive = 1;
146         }
147         else // User deactivates alarms
148         {
149             alarmsActive = 0;
150         }
151         break;
152     case 2:
153         // Serial.println("value:" + String(value));
154         TEST_ALARMS_VAR = value;
155         if (TEST_ALARMS_VAR == 1.0) // User activates alarms
156         {
157             manualAlarm = 1;
158         }
159         else // User deactivates alarms
160         {
161             manualAlarm = 0;
162         }
163         break;
164     case ERROR_VALUE:
165         Serial.println("Error");
166         Serial.println();
167         break;
168     default:
169         Serial.println("Default");
170         Serial.println();
171     }
172 }
```


- **Setup:** Esta función se ejecutará una vez al iniciar el programa y establecerá las conexiones necesarias con Ubidots, Telegram, Wifi, definirá el uso de los pines de la ESP32 entre otras acciones. Es fundamental antes inicializar las clases necesarios afuera del **void setup**:

```

58  Ubidots ubidots(UBIDOTS_TOKEN);
59  DHT dht(DHTPIN, DHTTYPE);
60  IRsend irsend(IR_EMITTER); // Set the GPIO to be used to sending the message.
61  WiFiClientSecure client;
62  UniversalTelegramBot bot(BOTtoken, client);

```

```

178 void setup() {
179     Serial.begin(115200);
180     Serial.println("Measurement initiated");
181     Serial.println("DHT11 and MQ-2 Turned On");
182     irsend.begin();
183     dht.begin();
184     pinMode(DO_FLAME_PIN, INPUT);
185     pinMode(BUZZER_PIN, OUTPUT);
186     pinMode(ALARM_LED, OUTPUT);
187
188     // Telegram connection
189     // Attempt to connect to Wifi network:
190     Serial.print("Connecting Wifi: ");
191     Serial.println(WIFI_SSID);
192
193     WiFi.mode(WIFI_STA);
194     WiFi.begin(WIFI_SSID, WIFI_PASS);
195     client.setCACert(TELEGRAM_CERTIFICATE_ROOT);
196
197     while (WiFi.status() != WL_CONNECTED) {
198         Serial.print(".");
199         delay(500);
200     }

```

```

201
202     Serial.println("");
203     Serial.println("Wi-Fi connected");
204     Serial.print("IP address: ");
205     Serial.println(WiFi.localIP());
206
207     bot.sendMessage(CHAT_ID, "Bot started up", "");
208
209     // Ubidots connection
210     ubidots.setDebug(true); // Uncomment this to make debug messages
211     ubidots.connectToWifi(WIFI_SSID, WIFI_PASS);
212     ubidots.setCallback(callback);
213     ubidots.setup();
214     ubidots.reconnect();
215     ubidots.add(TOGGLE_ALARMS, alarmsActive);
216     ubidots.publish(DEVICE_LABEL);
217     for (uint8_t i = 0; i < NUMBER_OF_VARIABLES; i++)
218     {
219         ubidots.subscribeLastValue(DEVICE_LABEL, variable_labels[i]);
220         delay(100);
221     }
222 }

```

- Loop:** Este contiene la lógica principal de la aplicación. Cada ciclo del **void Loop** dura 0.5 segundos para asegurarnos de tener actualizado el estado de las variables que permiten activar condicionalmente las alarmas. Lo primero del ciclo es reestablecer la conexión con Ubidots si por alguna razón nos desconectamos. Luego validando si han pasado los 0.5 segundos vamos a leer los valores de los sensores y empezamos mandando la señal al Monitor Serial, luego publicamos los valores en Ubidots y finalmente si existe fuga de gas o llamas se activan las funciones que encienden y apagan las alarmas físicas y digitales en el caso de Telegram.

```

224 void loop() {
225     static unsigned long timer = millis(); // Declare 'timer' as static
226     if (!ubidots.connected())
227     {
228         ubidots.reconnect();
229         for (uint8_t i = 0; i < NUMBER_OF_VARIABLES; i++)
230         {
231             ubidots.subscribeLastValue(DEVICE_LABEL, variable_labels[i]);
232             delay(100);
233         }
234     }
235
236     if (abs(static_cast<long>(millis() - timer)) > PUBLISH_FREQUENCY) {
237         float humidity = dht.readHumidity();
238         float temperature = dht.readTemperature();
239         int analogValue = analogRead(AO_MQ2_PIN);
240         float gasMeasure = (analogValue / 4095.0) * (10000 - 300) + 300;
241         int flame_stateSignal = digitalRead(DO_FLAME_PIN);
242         int gasState;
243         int flameState;
244
245         Serial.print("MQ2 sensor AO value: ");
246         Serial.print(gasMeasure);
247         Serial.println("\t");

```

```

248
249         if (gasMeasure > gasThreshold) {
250             gasState = 1;
251             Serial.println("The gas is present");
252         } else {
253             gasState = 0;
254             Serial.println("The gas is NOT present");
255         }
256
257         if (flame_stateSignal == HIGH) {
258             flameState = 0;
259             Serial.println("No flame detected => The fire is NOT detected");
260         } else if (flame_stateSignal == LOW) {
261             flameState = 1;
262             Serial.println("Flame detected => The fire is detected");
263         }
264
265         ubidots.add(TEMP_LABEL, temperature);
266         ubidots.add(HUMIDITY_LABEL, humidity);
267         ubidots.add(DO_FLAME, flameState);
268         ubidots.add(DO_MQ2, gasState);
269         ubidots.add(AO_MQ2, gasMeasure);
270
271         ubidots.publish(DEVICE_LABEL);
272     }

```

```

277 Serial.println("DO_MQ2_STATE: " + String(gasState));
278 Serial.println("AO_MQ2_MEASURE: " + String(gasMeasure));
279 Serial.println("-----");
280
281 // Check if it's time to start the inactivity timer
282 if (INACTIVITY_TIME != 0 && abs(static_cast<long>(millis() - CURRENT_INACTIVITY_TIME)) > INACTIVITY_TIME) {
283   alarmsActive = 1;
284   INACTIVITY_TIME = 0;
285   Serial.println("Activating alarms");
286 }
287 if (alarmsActive == 1 && (gasMeasure > gasThreshold || flameState == 1)){
288   turnOnAlarms();
289 }
290 else if (manualAlarm == 1) {
291   turnOnAlarms();
292 }
293 else {
294   turnOffAlarms();
295 }
296 timer = millis();
297 }
298
299 delay(500);
300 ubidots.loop();
301 }

```

- Funciones para activar / desactivar las alarmas:** Cuando se llama a la función **turnOnAlarms** si es que ya no está activada la luz roja, se manda una señal con el **emisor IR** para cambiar el color del foco según el **código hex** decodificado previamente. Se adjunta en el **repositorio** el código usado para decodificar el control. Luego se manda también un mensaje a **Telegram** de alarma, se ubica dentro del condicional para que se envíe solo un mensaje por evento. Luego se enciende y apaga de manera intermitente el buzzer y el led. La función **turnOffAlarms**, cambia el foco a blanco, apaga el led, el buzzer y manda un mensaje a Telegram indicando el estado controlado de la situación.

```

303 void turnOnAlarms() {
304   Serial.println("Turning on alarms");
305   if (colorLight != "red")
306   {
307     bot.sendMessage(CHAT_ID, "\xF0\x9F\x9A\xA8", "");
308     bot.sendMessage(CHAT_ID, "Attention! Gas leak or fire detected, please check your place!!!", "");
309     colorLight = "red";
310     irsend.sendNEC(0xFF6897, 32); // Send the hex code for the color red
311   }
312   digitalWrite(BUZZER_PIN, LOW); // Turn on the buzzer
313   digitalWrite(ALARM_LED, HIGH); // Turn on the LED
314   delay(250);
315   digitalWrite(BUZZER_PIN, HIGH); // Turn off the buzzer
316   digitalWrite(ALARM_LED, LOW); // Turn off the LED
317   delay(250);
318   digitalWrite(BUZZER_PIN, LOW); // Turn on the buzzer
319   digitalWrite(ALARM_LED, HIGH); // Turn on the LED
320 }
321
322 void turnOffAlarms() {
323   Serial.println("Turning off alarms");
324   if (colorLight != "white")
325   {
326     bot.sendMessage(CHAT_ID, "\xE2\x9C\x85", "");
327     bot.sendMessage(CHAT_ID, "Situation under control, no gas leak or fire detected. Be more careful!", "");
328     colorLight = "white";
329     irsend.sendNEC(0xFF52AD, 32); // Send the hex code for the color white
330   }
331   digitalWrite(BUZZER_PIN, HIGH); // Turn off the buzzer
332   digitalWrite(ALARM_LED, LOW); // Turn off the LED
333 }

```

8. Resultados

Logs en Serial Monitor: Los logs del Serial Monitor que se imprimen en el entorno de Arduino IDE permiten debuggear el programa y ver el estado de los sensores o la comunicación con Ubidots y Telegram.

```

gas-flame-alarm-system | Arduino IDE 2.3.0
File Edit Sketch Tools Help
ESP32 Dev Module
gas-flame-alarm-system.ino
18
19 /*Telegram credentials*/
Output Serial Monitor x
Message (Enter to send message to 'ESP32 Dev Module' on 'COM3')

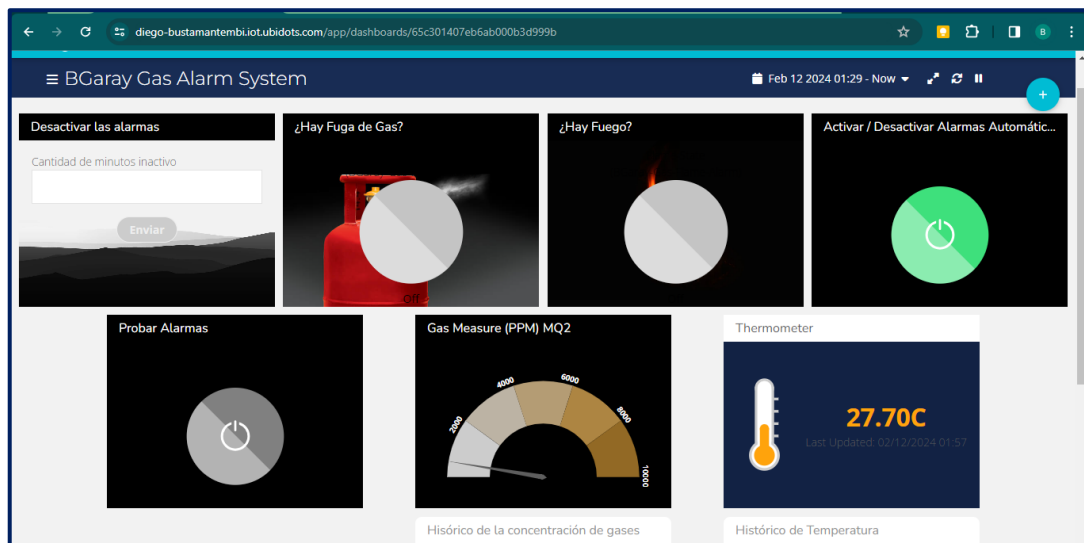
Turning off alarms
MQ2 sensor AO value: 802.17
The gas is NOT present
No flame detected => The fire is NOT detected
publishing to TOPIC:
/v2.0/devices/bqaray-gas-flame-alarm
JSON dict: {"temperature": [{"value": 27.7}], "humidity": [{"value": 73}], "flame-state": [{"value": 0}], "gas-state": [{"value": 0}],
Enviando los datos a Ubidots:
Temperatura: 27.70 °C
Humedad: 73.00 %
DO FLAME_STATE: 0
DO MQ2_STATE: 0
AO MQ2_MEASURE: 802.17

Turning off alarms
MQ2 sensor AO value: 809.28
The gas is NOT present
No flame detected => The fire is NOT detected
publishing to TOPIC:
/v2.0/devices/bqaray-gas-flame-alarm
JSON dict: {"temperature": [{"value": 27.7}], "humidity": [{"value": 73}], "flame-state": [{"value": 0}], "gas-state": [{"value": 0}],
Enviando los datos a Ubidots:
Temperatura: 27.70 °C
  
```

Alarmas físicas: Las alarmas físicas se activan cuando se detecta gas o fuego y se puede ver cómo cambia el color de la luz a rojo, se activa el buzzer y el led.



Dashboard en Ubidots: En el Dashboard podemos ver los indicadores, las estadísticas de los sensores y los botones que permiten probar, activar o desactivar las alarmas por una cantidad de minutos específica.



Telegram: En el chat Bot de Telegram podemos ver un mensaje automático cuando se detecta gas o llamas y cuando se controla la situación se envía otro mensaje.



9. Conclusión

La realización de este proyecto fue una experiencia integral que permitió poner en práctica diferentes habilidades tecnológicas de programación, electrónica, construcción de circuitos, manejo de repositorios y tecnologías IoT.

Se pudo evidenciar que con una correcta planificación tanto teórica como técnica se pueden abordar problemas cotidianos y dar una solución desde el ámbito del IoT y la programación de Sistemas Embebidos, lo que permite al estudiante poder aportar activamente a mejorar el bienestar de su sociedad mediante la prevención, monitoreo y alerta temprana.

Se pudieron cumplir los objetivos del proyecto ya que se construyó un prototipo funcional con el microcontrolador ESP32, capaz de monitorear, detectar y alertar en tiempo real fugas de gases o llamas. Así mismo, se logró integrar tecnologías del IoT como Ubidots con un Dashboard que se configuró de manera tal que permitió la entrada y salida de información. Finalmente se llevó el proyecto más allá de las tecnologías vistas a lo largo del curso, haciendo una integración en Telegram con un Bot para enviar mensajes y alertar de manera práctica y moderna al usuario del sistema.

10. Referencias

- Babiuch, M., Foltýnek, P., & Smutný, P. (2019, May). Using the ESP32 microcontroller for data processing. In 2019 20th International Carpathian Control Conference (ICCC) (pp. 1-6). IEEE. Recuperado de: <https://ieeexplore.ieee.org/abstract/document/8765944>
- El Universo (2023). Una persona herida y daños materiales por acumulación de gas en Quito, este sábado. Quito, Ecuador. Recuperado de: <https://www.eluniverso.com/noticias/ecuador/quito-vivenda-afectada-explosion-gas-nota/>
- Kanakaraja, P., Sundar, P. S., Vaishnavi, N., Reddy, S. G. K., & Manikanta, G. S. (2021). IoT enabled advanced forest fire detecting and monitoring on Ubidots platform. Materials Today: Proceedings, 46, 3907-3914. Recuperado de: <https://www.sciencedirect.com/science/article/abs/pii/S2214785321014449>
- Mota, V. S. (2014). Sistema de monitoreo mediante una Red de Sensores. Universidad de Pamplona. , ISSN 0122-820X, ISSN-e 2422-5053, Vol. 25, N°. 1, 2020. Recuperado de: <https://dialnet.unirioja.es/servlet/articulo?codigo=7611541>