



Escuela Politécnica

Samuel Díaz  
Padilla

# PROYECTO ARDUINO

Sensor con aviso a Telegram y  
con alarma integrada

Fecha: 8/11/2024



SISTEMAS DIGITALES – 3º –

## **ÍNDICE**

1. MOTIVACIÓN DEL DESARROLLO DEL PROYECTO
2. MATERIALES Y ESQUEMA DEL HARDWARE
3. EXPLICACIÓN DEL CÓDIGO EMPLEADO Y FUNCIONAMIENTO
4. RESULTADO FINAL
5. IMPLEMENTACIÓN, FUTURAS MEJORAS Y CONCLUSIÓN
6. BIBLIOGRAFÍA

## **1- MOTIVACIÓN DEL DESARROLLO DEL PROYECTO**

Hoy en día, la seguridad es esencial y realmente importante para la protección de nuestros hogares. Por eso mismo, el diseño de un sensor de movimiento con aviso a Telegram puede ser útil además de interesante.

Este diseño con aviso a Telegram puede ir desde el propio sensor que se ha realizado en este proyecto hasta cámaras, sensores de humedad, de humo, magnéticos en puertas o de temperatura, capaces de monitorear desde el más remoto rincón de un domicilio.

Por último, el hecho de aprender temas de ciberseguridad es gratamente satisfactorio, al realizar algo sencillo y práctico con tus propios conocimientos.

## **2- MATERIALES Y ESQUEMA DEL HARDWARE**

Se detallará los materiales empleados y uso de cada uno de ellos en el proyecto:

- Chip ESP32 **(1)**: Encargado del funcionamiento completo, con WI-FI y Bluetooth integrado. Es el microprocesador que se programará, es decir, el cerebro del circuito. Al ESP32 irán conectados pines, fuentes de voltaje y tierra para el resto de las componentes.
- LED **(2)**: Diodo emisor de luz que se encenderá cuando se active el pin declarado en el código, teniendo este un funcionamiento intrascendente.
- Resistencia de  $1k\Omega$  **(3)**: Usada para que el LED tenga un voltaje dentro de su rango, y evitar así su destrozo.
- Módulo PIR (Passive Infrared) **(4)**: Módulo detector de infrarrojos responsable de la detección del movimiento
- Buzzer **(5)**: Dispositivo capacitado para convertir una señal eléctrica a una onda sonora. Será el encargado de emitir un sonido cuando así lo indiquemos.
- Cables de tipo “hembra-hembra” y “macho-hembra” **(6)**: Empleados para conectar los elementos del circuito.

**(1)**



**(2)**



**(3)**



**(4)**



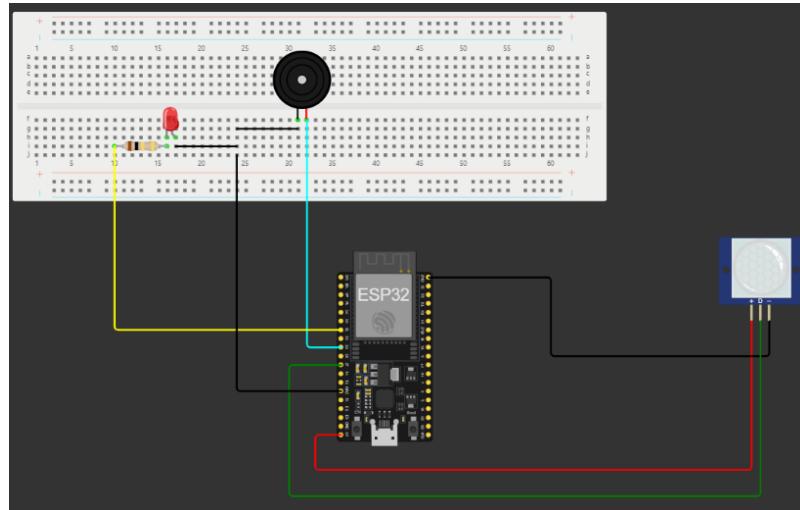
**(5)**



**(6)**



A continuación, se muestra el circuito utilizado con todos los elementos previamente citados:



Puede ser que el patillaje del módulo PIR de la figura no sea exactamente idéntico con el que se ha utilizado en el proyecto, pues depende explícitamente del fabricante.

Una vez concluido el montaje falta detallar lo más difícil e imprescindible, la implementación del código.

### 3- EXPLICACIÓN DEL CÓDIGO EMPLEADO Y FUNCIONAMIENTO

La mejor manera de explicar el código será por partes y viendo módulo a módulo.

#### LIBRERÍAS EMPLEADAS

```
//Incluimos las librerías necesarias, importando la librería  
UniversalTelegramBot en zip  
#include <WiFi.h>  
#include <WiFiClientSecure.h>  
#include <UniversalTelegramBot.h>  
#include <ArduinoJson.h>
```

Comenzaremos en primera instancia descargando desde el propio programa de Arduino la librería **ArduinoJson.h** (Java Script Object Notation).

Los beneficios de utilizar esta librería es la facilidad de transferencia de datos que hacemos con el ESP32, para descomponer datos recibidos por webs o aplicaciones, en este caso Telegram.

También se deberá instalar otra librería llamada esp32, esta se encargará de que el programa lea correctamente el microcontrolador utilizado, debido a que hay una cantidad de modelos.

Respecto a **Wifi.h** y **WiFiClientSecure.h** son librerías esenciales para que nuestro microprocesador establezca una conexión fácil y segura con el servidor deseado, y así, poder enviar mensajes. No tienen mucho más que destacar, simplemente hacen lo más sencillo e importante.

Por último, comentaremos la librería **UniversalTelegramBot**, que se descargará de manera externa a Arduino y agregaremos añadiendo simplemente su archivo .zip

### WIFI Y BOT DE TELEGRAM

```
// Introducir la red WIFI y contraseña
const char* ssid = "Mitelefono";
const char* password = "HXBWSJHCVWjhbstjxb900";

// Iniciamos el BOT de telegram introduciendo el token
#define BOTtoken "8132305778:AAGYGDsYhYmfSikUZBxYmnubStfQshX9bXc"

// Introducimos nuestra ID chat de telegram
#define CHAT_ID "1313718119"

WiFiClientSecure client;
UniversalTelegramBot bot(BOTtoken, client);
```

Mediante estas instrucciones el ESP32 se conectará a la red que le hayamos indicado una vez que le actualicemos el código, más adelante en el setup se comprobará que esté correctamente conectado.

También definimos el bot de Telegram mediante el BOTtoken y el usuario mediante el CHAT\_ID. Finalmente creamos un enlace seguro para que el bot de telegram se comunique con el microprocesador.

### DECLARAMOS VARIABLES

```
//Introducimos las variables necesarias para el funcionamiento del código,
tanto temas de patillaje como estados.
const int buzPin = 25; //Definimos el buzzer
const int led = 32; //Definimos el LED
const int PIR = 27; // Sensor PIR
bool motionDetected = false;
bool alarmaActiva = false; // Controlamos el estado de la alarma
bool sensorActivo = false; // Variable para controlar si el sensor está
activo o no
// Variables de retardo para la verificación del movimiento
unsigned long lastDetectionTime = 0; // Tiempo de la última detección
const unsigned long detectionDelay = 100; // Retardo de verificación en
milisegundos
```

```
// Variable para control del tiempo del último mensaje enviado, lo utilizaremos para que no mande continuamente mensajes
unsigned long lastMessageTime = 0; // Tiempo del último mensaje de movimiento enviado
const unsigned long messageInterval = 30000; // Intervalo mínimo entre mensajes en milisegundos (30 segundos)
```

Aunque no este claro la implementación de estas variables se detallará de manera escueta su futuro funcionamiento.

Las tres primeras son el buzzer, el led y el sensor PIR respectivamente, y les otorgamos el pin al que irán conectadas al microcontrolador de manera minuciosa.

Las tres siguientes son variables booleanas que ayudaran a saber el estado en el que se encuentra en sensor y la alarma/buzzer.

Para terminar las variables `lastDetectionTime` y `detectionDelay` las utilizaremos en un módulo para confirmar la detección de movimiento. Por otra parte, `lastMessageTime` y `messageInterval` se destinarán a que el bot no mande más mensajes en un rango definido de tiempo.

### PRIMER MÓDULO, COMPROBACIÓN DE MOVIMIENTO

```
// Indica cuando se detecta movimiento
void IRAM_ATTR detectsMovement() {
    // Solo detecta si el sensor está activo y ha pasado el retardo de tiempo agregado arriba en las variables
    if (sensorActivo && (millis() - lastDetectionTime > detectionDelay)) {
        motionDetected = true;
        lastDetectionTime = millis();
    }
}
```

El presente módulo se encarga de comprobar si el módulo PIR ha detectado movimiento. Marca el movimiento como detectado (`motionDetected = true`) si se el sensor ha recibido información de movimiento (`sensorActivo`) y ha transcurrido un tiempo mínimo impuesto con anterioridad, para así evitar múltiples activaciones seguidas.

Se debe destacar el uso de la palabra clave `IRAM_ATTR`. Esta función es de suma utilidad, realiza una actualización rápida a la memoria RAM de nuestro microprocesador ESP32, por tanto, ayudará a verificar este módulo velozmente.

El siguiente bloque se detallará por partes, siendo este sencillo de comprender, pero extenso.

## PARTE 1 - INICIACIÓN Y COMPROBACIÓN DE USUARIO

```
//Módulo que se encarga de enviar mensajes dependiendo de la situación en  
la que nos encontremos  
void handleNewMessages(int numNewMessages) {  
//Bucle for que hace de intermediario entre telegram(los comandos que  
indicamos) y nuestro código actual  
    for (int i = 0; i < numNewMessages; i++) {  
        String chat_id = String(bot.messages[i].chat_id);  
  
    //Verificamos que el ID de usuario de telegram sea correcto con el móvil que  
    tenemos, de lo contrario el acceso será denegado  
        if (chat_id != CHAT_ID) {  
            bot.sendMessage(chat_id, "ACCESO DENEGADO", "");  
            continue; //Reiniciamos para evaluar la condición del bucle nuevamente  
    }
```

Lo primero de todo será destacar el funcionamiento del módulo. La función **handleNewMessages** es la encargada de detectar comandos enviados por Telegram y gestionarlos mediante el ESP32, teniendo en cuenta que el usuario tiene que estar autorizado.

En segundo lugar, el bucle **for** es el encargado de recopilar los mensajes enviados desde Telegram y extraer su chat ID, para que el usuario sea el introducido en el inicio del programa, tal y como hemos especificado antes. En caso de que el ID no sea el indicado imprimirá por el móvil un mensaje de error.

## PARTE 2 – ENVÍO Y USO DE COMANDOS

```
String text = bot.messages[i].text;
```

Extraemos los mensajes y los mensajes que recibimos y los asociamos a la variable **text**, que utilizaremos para comparar el comando con las diversas funcionalidades que se hayan agregado a nuestro código.

```
//Este if hace activar la alarma con un pitido y un led  
if (text == "/alarma_on") {  
    bot.sendMessage(chat_id, "Alarma activada", "");  
    alarmaActiva = true; // Activa la alarma  
    digitalWrite(buzPin, HIGH); // Enciende el buzzer  
    digitalWrite(led, HIGH);  
}  
  
//Apagamos la alarma  
if (text == "/alarma_off") {  
    bot.sendMessage(chat_id, "Alarma desactivada", "");  
    alarmaActiva = false; // Desactiva la alarma  
    digitalWrite(buzPin, LOW); // Apaga el buzzer  
    digitalWrite(led, LOW); // Apaga el LED
```

```
}

    if (text == "/activar") {
        bot.sendMessage(chat_id, "Sensor activado", "");
        sensorActivo = true; // Activa el sensor PIR
        attachInterrupt(digitalPinToInterrupt(PIR),detectsMovement,RISING);
        // Reactiva el sensor
    }

    //Desactivamos el sensor de manera permanente hasta que no se indique
    //lo contrario con /activar
    if (text == "/desactivar") {
        bot.sendMessage(chat_id, "Sensor desactivado", "");
        sensorActivo = false; // Desactiva el sensor PIR
        detachInterrupt(digitalPinToInterrupt(PIR)); // Desactiva el sensor
    }
}
}
```

Se han declarado distintos comandos útiles para dar funcionalidad a la alarma. El primer comando es [/alarma\\_on](#) que hará que el LED se encienda de manera permanente y el buzzer emita un pitido hasta que no se indique que este proceso de detenga mediante el comando opuesto [/alarma\\_off](#).

El sensor PIR en un inicio estará desactivado, es decir, no detectará movimiento. En el supuesto de que se quiera activar el sensor se tendrá que utilizar [/activar](#), y [/desactivar](#) para que el sensor vuelva a dejar de trabajar. Esto es verdaderamente efectivo para cuando el cliente llegue o salga de la casa.

#### MÓDULO DE CONFIGURACIÓN – SETUP

```
void setup() {
    Serial.begin(9600);

    // Configuración del sensor PIR, buzzer y led
    pinMode(PIR, INPUT_PULLUP);
    pinMode(buzPin, OUTPUT);
    pinMode(led, OUTPUT);

    // Conexión WiFi, NECESARIO LAS BIBLIOTECAS
    Serial.print("Connecting Wifi: ");
    Serial.println(ssid);

    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    client.setCACert(TELEGRAM_CERTIFICATE_ROOT); // Agrega el certificado
    //raíz para api.telegram.org
```

```

//Verificamos que la conexión al WIFI se haya realizado correctamente,
de lo contrario escribirá "." continuamente
while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
}

//Indicamos la conexión correcta del WIFI
Serial.println("");
Serial.println("WiFi connected");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

//Nada más iniciar el bot, mandará un mensaje de bienvenida con los
comandos de los que dispone el usuario
String welcomeMessage = "Bot iniciado\n\n";
welcomeMessage += "Comandos disponibles:\n";
welcomeMessage += "/alarma_on - Activa la alarma y el buzzer\n";
welcomeMessage += "/alarma_off - Desactiva la alarma y el buzzer\n";
welcomeMessage += "/activar - Activa el sensor de movimiento\n";
welcomeMessage += "/desactivar - Desactiva el sensor de movimiento\n";
bot.sendMessage(CHAT_ID, welcomeMessage, "");
}

```

El módulo de configuración se ejecuta una única vez cuando el ESP32 se enciende o reinicia. Se encarga de configurar el hardware del sistema.

Definimos la velocidad del monitor serie y declaramos los pines. Es interesante ver como el sensor PIR está siendo utilizado como una resistencia de pull-up “INPUT\_PULLUP”, encargado de elevar la tensión de salida. Posteriormente agregamos el certificado de Telegram y verificamos por pantalla al monitor serie que la conexión WIFI se ha realizado correctamente, escribiendo “.” si no se ha conseguido efectuar la conexión.

Finalmente, el bot comenzará dando un mensaje de bienvenida con los comandos definidos en el sistema.

	/activar	/desactivar	/alarma_on	/alarma_off
SENSOR	✓	✗	-	-
ALARMA	-	-	✓	✗

Símbolo “-” representa que puede estar activo o no.

Símbolo ✓ significa que está activo.

Símbolo ✗ significa que no está activo.

La alarma y el sensor se activan y desactivan manualmente, son elementos independientes.

### BLOQUE LOOP

```
void loop() {
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    //Bucle que está continuamente detectando la entrada de nuevos mensajes
    while (numNewMessages) {
        handleNewMessages(numNewMessages);
        numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }

    //Verificamos el movimiento
    if (motionDetected) {
        unsigned long currentMillis = millis();
        // Solo envía el mensaje si han pasado al menos 30 segundos desde el
        // último mensaje, esto hará que el usuario no reciba muchos mensajes
        if (currentMillis - lastMessageTime >= messageInterval) {
            bot.sendMessage(CHAT_ID, "Movimiento detectado!!", "");
            lastMessageTime = currentMillis; // Actualiza el tiempo del último
            mensaje
        }
        motionDetected = false; // Reinicia la variable de movimiento
        detectado
        Serial.println("Movimiento detectado");
    } else {
        Serial.println("No se detecta movimiento"); //Indicamos que no se
detecta movimiento por el monitor serie
    }
}
```

El módulo LOOP es el que se estará en continuo funcionamiento una vez que el microcontrolador se haya configurado a la perfección.

La función a la que llamamos `numNewMessages` es la encargada de recibir los mensajes, que a posteriori se comprobarán en el bucle `while`, llamando a la función `handleNewMessages` para ver cuál es el comando que el usuario está reclamando.

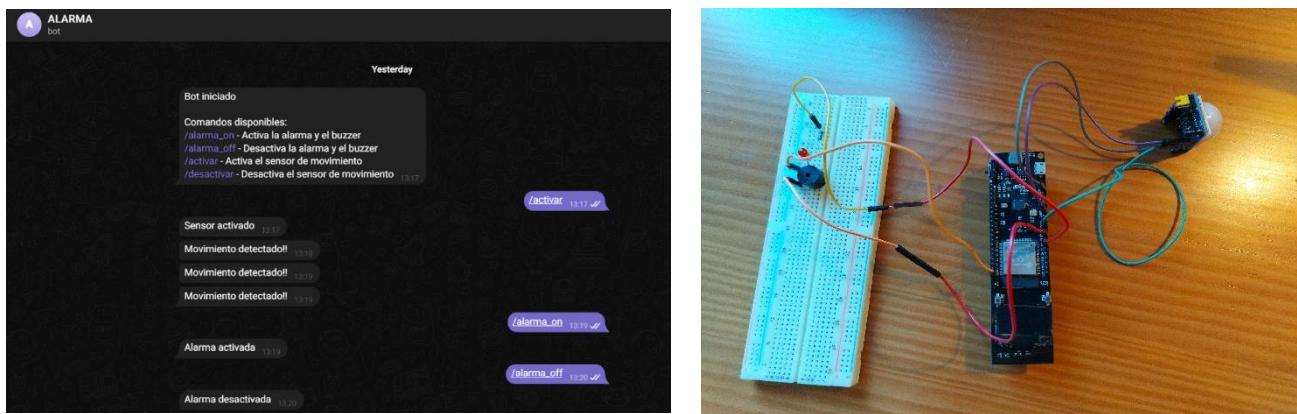
El control de detección de movimiento se realiza mediante un “if”, el cual se emplea para tener un control de frecuencia de mensajes gracias a la variable `messageInterval`, definida al comienzo del código.

Si se detecta movimiento se imprimirá por pantalla “`Movimiento detectado`” y “`No se detecta movimiento`” en caso contrario.

Aquí se da por finalizada la explicación del código diseñado.

#### 4- RESULTADO FINAL

Realizando todos los pasos como se han indicado, construimos un sensor de movimiento con aviso a Telegram capacitado para que, mediante el uso de comandos definidos, podamos activar la alarma en caso de sospecha de movimiento. A continuación, se muestra una captura del chat de Telegram verificando el correcto enlace entre nuestro microcontrolador ESP32 y el bot, además de una foto real del montaje.



#### 5- IMPLEMENTACIÓN, FUTURAS MEJORAS Y CONCLUSIÓN

Hoy en día la digitalización es sumamente importante y necesaria, lo que significa que es un punto a favor de la protección de nuestros hogares. Es por ello que realizar una alarma es sencillo, además de útil para una vivienda. Aunque no solo podría ser algo personal, también se podría utilizar para la automatización de entorno como invernaderos o plantaciones solares, oficinas, domotización de un edificio o en parkings para detectar si una plaza se encuentra ocupada, entre otras muchas opciones.

Futuros avances sobre el proyecto pueden ir ligados a la seguridad, pero con un cambio de temática, ya sean detectores de humo, de humedad, cámara, etc.

Pueden surgir numerosos avances respecto al código que lo haría más completo. Un módulo SIM conectado a nuestro controlador sería una buena idea para que, en caso de no tener una conexión a internet estable por cualquier circunstancia, poder al menos realizar una llamada al usuario por su seguridad.

En última instancia, este proyecto ha sido una excelente oportunidad para explorar la integración de sensores y redes inalámbricas en sistemas de seguridad. Hemos logrado un sistema de monitoreo que se adapta a nuestras necesidades cotidianas, brindándonos tranquilidad y control a distancia de manera sencilla y accesible.

## 6- BIBLIOGRAFÍA

- Uso de la función handleNewMessages ([fuente](#))
- Uso del comando continue ([fuente](#))
- Forma utilizada de aplicar retardo con millis() y tiempo definido ([fuente1](#)) ([fuente2](#))
- Creación del bot de Telegram ([fuente](#))
- Descarga de la librería de Telegram necesaria para el funcionamiento ([fuente](#))
- Agregar preferencias del ESP32 en Arduino IDE ([fuente](#))
- Uso de la librería Json ([fuente](#))
- Guía de instalación para que el ordenador reconozca el puerto del ESP32 ([fuente](#))