Nodos de ejemplo: Detector de Presencia

En este nodo creamos un detector de presencia que compone un sistema de alarma con aviso a teléfonos móviles.

Podemos encontrar el código en github. https://github.com/turbolargoo/Free-Connect/tree/master/Ejemplos%20proporcionados/detector_presencia

Se presuponen conocimientos básicos sobre programación de Arduino.

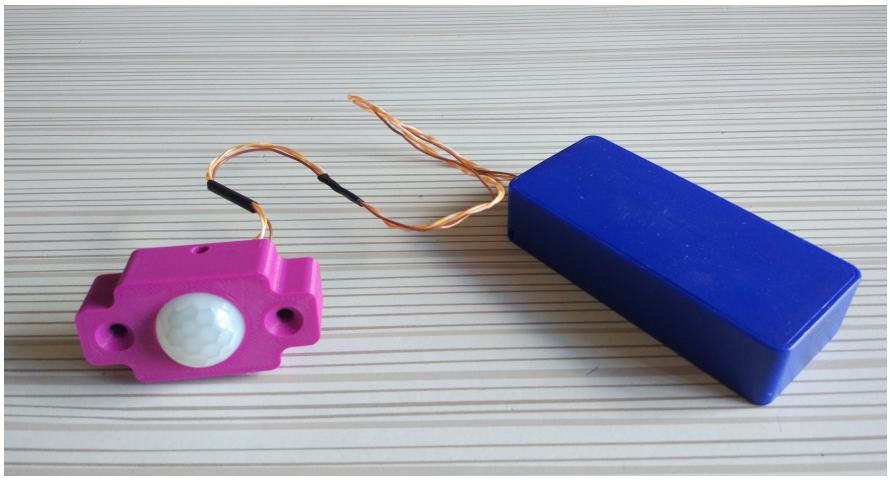

Figura 1: Nodo Detector de presencia

Características

- Activación y desactivación desde la interfaz.
- Aviso directo al móvil mediante el cliente de mensajería Telegram.

- Personalización de parámetros.
- Detección de falsos positivos.

Hardware empleado

- Placa esp32.
- Sensor PIR.
- Optoacoplador.
- Sirena

Circuito eléctrico

A continuación describimos el circuito diseñado:

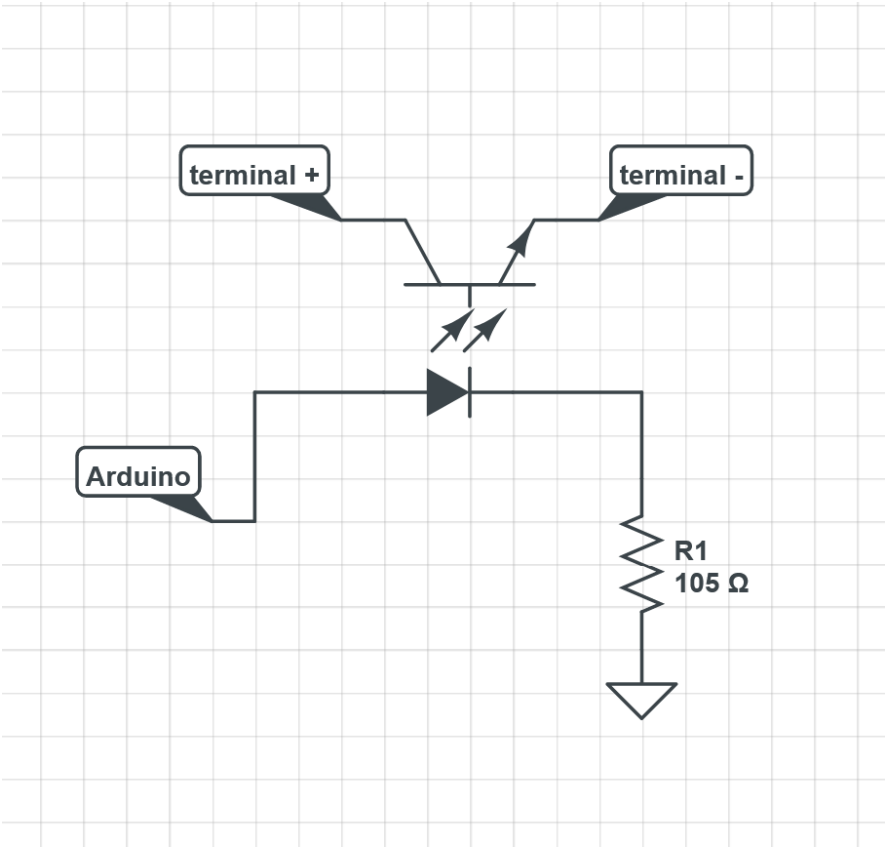

Figura 2: Circuito eléctrico para el nodo Detector de presencia

Utilizamos un optoacoplador Sharp PC817 y una resistencia para controlar el encendido de la sirena tal y como aprendimos en el ejemplo del Termostato

El sensor PIR se conecta directamente a la placa ESP32 ya que su señal es de 3.3V

Para la conexión del optoacoplador con el circuito que queremos controlar es necesario fijarse en la polaridad de la sirena que suele venir indicada con el color de los cables.

Es muy importante asegurarnos de que el optoacoplador aguantará la tensión e intensidad del circuito que vamos controlar. Si este no fuera el caso, debemos buscar otro optoacoplador más potente o un transistor. Para medir la intensidad, conectamos el polímetro (con la escala más grande de amperios) entre contactos.

Consideraciones sobre el sensor PIR

El sensor PIR utilizado es un DSN-FIR800.

Con el potenciómetro de ajuste de tiempo al mínimo obtenemos un pulso de 2 segundos cada vez que detecta un objeto.

En la imágen podemos ver como cada vez que paso la mano por delante del sensor se registra un pulso de 3.3V que dura exáctamente 2 segundos (una división con esa configuración de tiempo).

Esta información la utilizaremos para ajustar los tiempos de reset en el código.

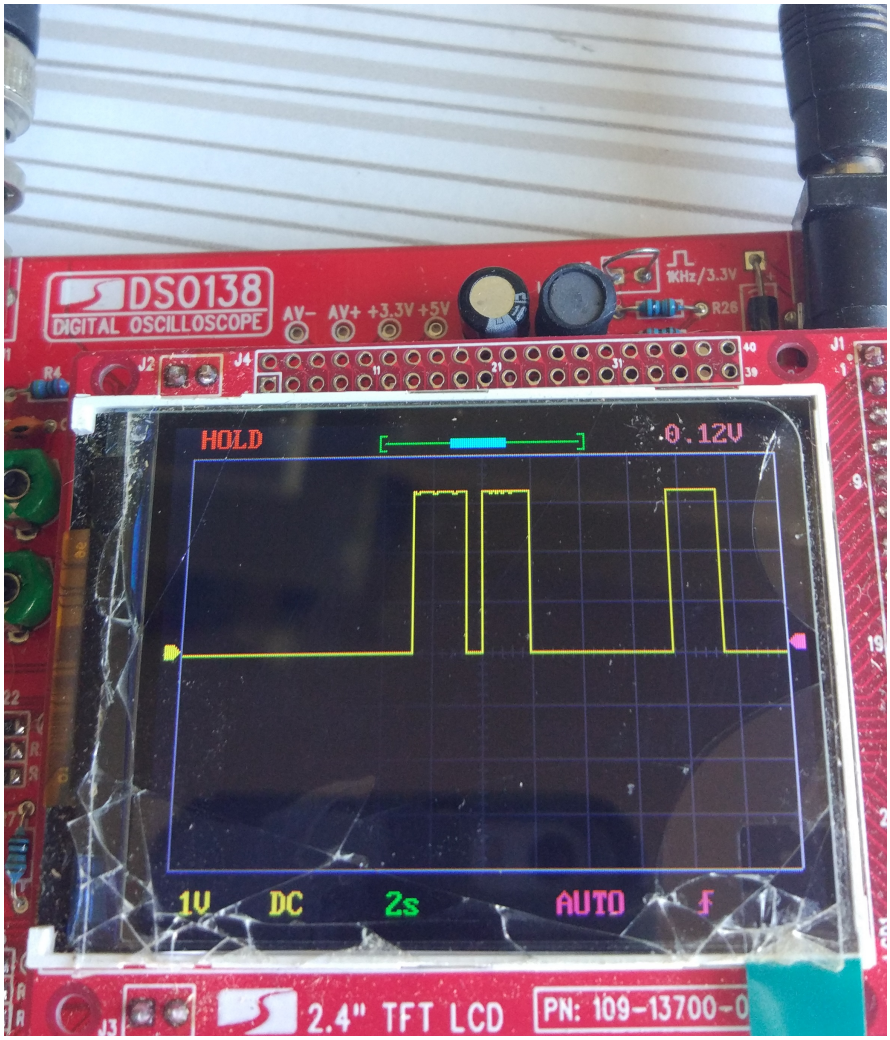

Figura 3: Señal enviada por el sensor PIR vista desde un osciloscopio

Programación

Partimos del código base para este desarrollo. A continuación se muestra el código añadido.

Archivo de configuración

Añadimos una serie de líneas:

1. **Configuración de pines:** Ubicación de la sirena y sensor. Adicionalmente incluimos el LED integrado en la placa.

   const int pir=26; const int led=2; const int siren=4;

2. **Control de detecciones**

   - **numDetections** Es una constante. Indica el número de veces consecutivas que debe activarse el sensor para no considerarse un falso positivo.
   - **detectionCount** Es la variable donde se van contando las detecciones.

3. **Control de tiempos**

   - **t** Variable para almacenar instante de tiempo cuando se realiza la primera detección.
   - **tReset** Tiempo que define el intervalo tras el que una detección se considera falsa y resetea el contador.
   - **tMessages** Tiempo que espera entre envíos de mensaje de alerta.
   - **tLastMessage** Variable que almacena el instante cuado se mandó el último mensaje.
   - **tTriggered** Variable que almacena cuándo comenzó a sonar la alarma.
   - **tRinging** Constante que determina cuánto tiempo sonará la alarma.

4. **Ajustes Webthings**

   Configuramos la capacidad de alarma, dos propiedades y un evento.

```
const char deviceName[] ="alarm";
WebThingAdapter *adapter;
const char *capacidades[] = {"Alarm", nullptr};
ThingDevice Sensor("alarma", "detector presencia", capacidades);
ThingProperty Detectado("detectado", "persona detectada", BOOLEAN,
 "AlarmProperty");
ThingProperty Activado(«activado», "alarma ready", BOOLEAN,
"BooleanProperty");
ThingEvent DeteccionEvento("persona","detectada",BOOLEAN,
 "AlarmEvent");
```

   Detectado indica si la alarma ha saltado y está sonando.

   Activado controla si la alarma está armada o desarmada.

   El evento se lanzará cuando la alarma salte para desencadenar acciones en la interfaz.

5. **Configuración de telegram** Introducimos los parámetros necesarios: `#define bOTtoken "token"` uTLGBot Bot(bOTtoken); const int debugLevelBot = 0; const int numChats=2; //we can specify to send the messages to more than one account //we specify here the chat ids we want the messages sent to `const char chatID[]={"chat"};`

   `const char chatID2[]={"chat"};`

   `const char *chatIDs[numChats]={chatID,chatID2};`

Setup

Añadimos la inicialización de los pines utilizados.

Añadimos la interrupción para el sensor. Escogemos la opción RISING"para detectar el flanco de subida (instante de inicio del puslo enviado por el sensor).

Loop

En este tipo de proyectos es muy importante realizar el control de tiempos guardando el resultado de la función millis() y comparándolo con los valores predefinidos en la configuración en lugar de utilizar delays. Esto es basar las condiciones de ejecución en el control de tiempo del microprocesador en vez de pararlo con delays.

En primer lugar encotramos un bloque condicional para diferenciar si la alarma está armada o no.

- **Si está armada:** Encontramos cuatro condiciones:
  1. Ha habido alguna detección pero se supera el tiempo de reset, ponemos el número de detecciones a 0
  2. La alarma no está sonando y alcanzamos el número de detecciones fijado, hacemos sonar la alarma.
  3. La alarma está sonando y se supera el tiempo de envío de mensaje, enviamos un nuevo mensaje.
  4. La alarma está sonando y se supera el tiempo máximo para estar sonando, apagamos la alarma.
- **Si no está armada:** Llamamos a la función para parar de sonar. untriggerAlarm();

```
if(active){ //if alarm is armed
  if(detectionCount>0&&millis()>t+tReset){//some detections were made but reset time
    detectionCount=0;
  }
  if(!triggered&&numDetections==detectionCount){//detections threshold reached
    triggerAlarm();
  }
  if(triggered&&millis()>tLastMessage+tMessages){//it's time to send another message
    sendMessagege();
  }
  if(triggered&&millis()>tTriggered+tRinging){//alarm has rung for long enough
   untriggerAlarm();
  }
}else{
  untriggerAlarm();
}
```

Figura 4: Código en la sección loop del Nodo Detector de presencia que controla los estados del sistema

En el último bloque ejecutamos tres comandos:

Actualizar las datos contra la interfaz.

Comprobar si debemos armar o desarmar la alarma.

Llama a la función de actualización inalámbrica.

imagenloop2detect.pngCódigo en la sección loop del Nodo Detector de presencia que hace las llamadas a funciones de actualización de datos

Funciones

En este archivo se encuentran las funciones llamadas desde el loop. Comentaremos los aspectos mas relevantes sobre las funciones:

- **alerta** Esta es la función que ejecuta la interrupción que desencadena el sensor con cada pulso que manda. Suma uno al contador de detecciones y guarda el tiempo de la primera detección.

```
void alerta(){
  if(detectionCount==0){
    t=millis();
  }
  if(!triggered){
    detectionCount++;
  }
}
```

Figura 5: Función alerta del nodo Detector de presencia

- **triggerAlarm** Hace saltar la alarma. Enciende la sirena y led de aviso, guarda el instante de tiempo de activación, cambia el estado de las variables en la interfaz y envía el evento.

```
void triggerAlarm(){
  triggered=true;
  tTriggered=millis();
  digitalWrite(led,HIGH);
  digitalWrite(siren,HIGH);
  ThingPropertyValue value;
  value.boolean = true;
  Detected.setValue(value);
  ThingEventObject *ev = new ThingEventObject("DetectionEvent", BOOLEAN, value);
  Sensor.queueEventObject(ev);
}
```

Figura 6: Función triggerAlarm del nodo Detector de presencia

- **untriggerAlarm** Revierte los cambios hechos por triggerAlarm.

```
void untriggerAlarm(){
    triggered=false;
    detectionCount=0;
    digitalWrite(led,LOW);
    digitalWrite(siren,LOW);
    if(Detected.getValue().boolean==true){
     ThingPropertyValue value;
     value.boolean = false;
     Detected.setValue(value);
    }
}
```

Figura 7: Función untriggerAlarm del nodo Detector de presencia

- **sendMessage** Envía el mensaje de alarma a todos los chats.

```
void sendMessagege(){
  tLastMessage=millis();
  for (int i=0; i<numChats; i++) {
       Bot.sendMessage(chatIDs[i], "Intrusion detected");
  }
}
```

Figura 8: Función sendMsg del nodo Detector de presencia

- **checkStatus** Refresca los datos de la interfaz y devuelve el estado de la variable que define si la alarma se encuentra armada o no.

```
boolean checkStatus(){
   adapter->update();
   return Active.getValue().boolean;
}
```

Figura 9: Función checkStatus del nodo Detector de presencia

Coste de este nodo

Descripción	Unidades	Coste unitario €	Coste total €
Placa esp32	1	9	9
Placa perfboard	1	2	2
Optoacoplador	1	0,25	0,25
Resistencias	1	0,1	0,1
Sensor PIR	1	0,9	0,9
Sirena	1	15	15
Impresión 3D	1	2	2

Tabla 1: Coste del nodo Detector de presencia

Gastos totales de material: 29,25€