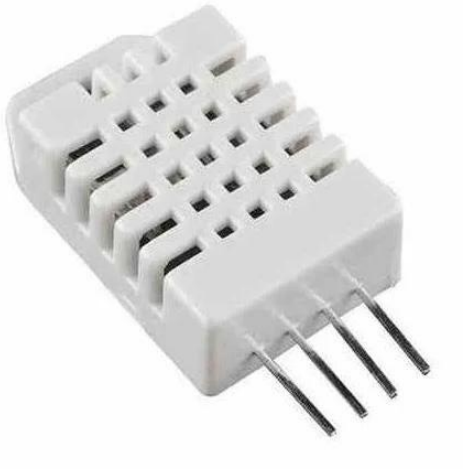
INSTRUTIVO ESP32 (Sensores, actuador, control, comunicación (BT+Wifi+EspNow))

Sensores usados:

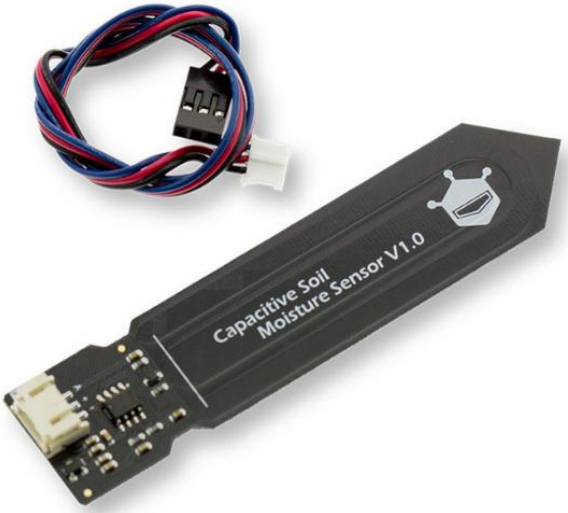
- **DHT22(Humedad y Temperatura)**



- **Sensor de agua (Vertical)**

- **Sensor de agua (Horizontal)**

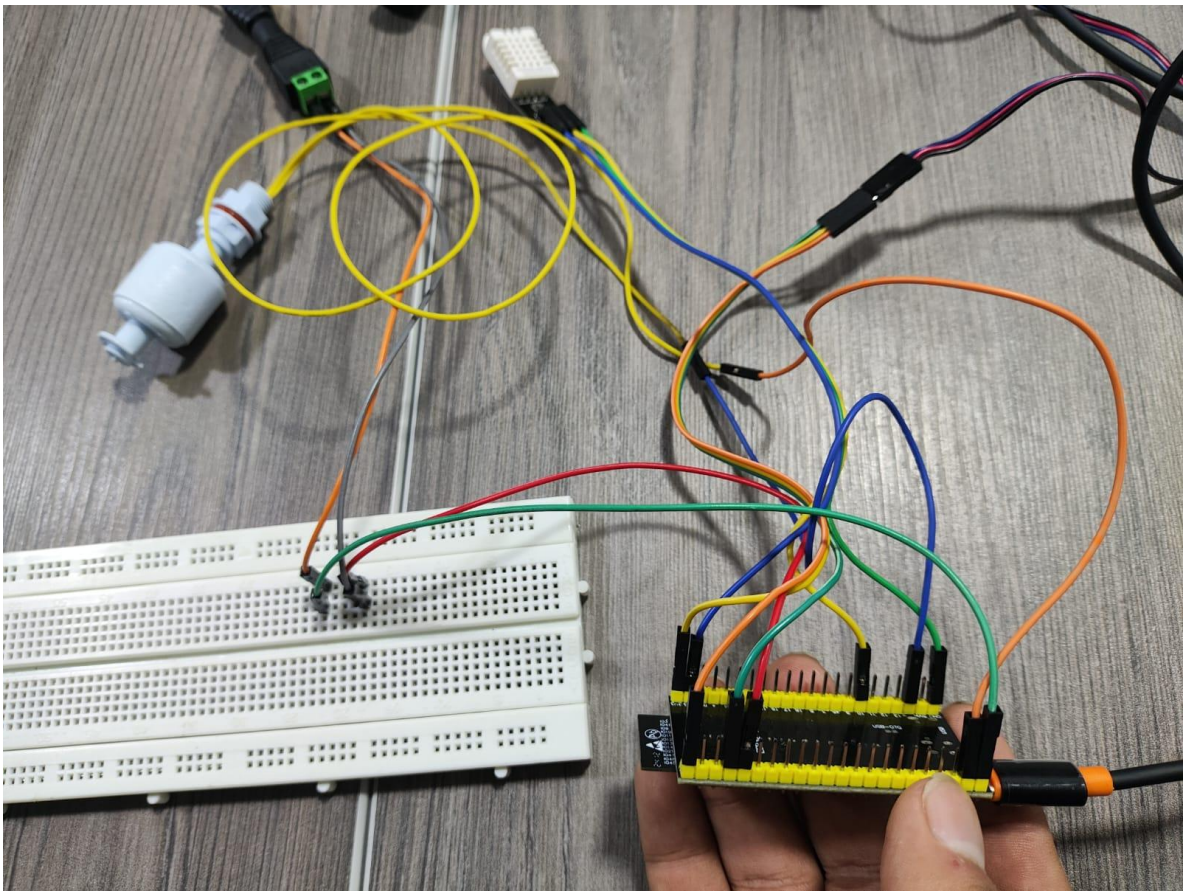
- **sensor de humedad del suelo**



- **ESP32CAM**

DOCUMENTACION SENSORES

El Resultado del cableado final con todos los sensores fue el siguiente:



Luego se quemo el sketch en el Esp32-s3 el principal de los sensores. El cual fue el siguiente:

1. Primero se importaron las librerías

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
```

   - WiFi.h: Para conexión WiFi (Para manejar la conexión mqtt).
   - Adafruit_Sensor.h, DHT.h, DHT_U.h: Permiten la lectura del sensor DHT22 (temperatura y humedad).
2. Se definen los pines

```
// Sensor de humedad del suelo
const int sueloPin = 1;  // GPIO1 (ADC1_CH0)

// Sensor DHT22
#define DHTPIN 10         // GPIO10
#define DHTTYPE DHT22
DHT_Unified dht(DHTPIN, DHTTYPE);

// Sensores de nivel líquido (horizontal/vertical)
#define SENSOR_H_MAX 2    // GPIO2
#define SENSOR_H_MIN 3    // GPIO3
#define BOMBA_HORIZONTAL 4 // GPIO4
#define BTN_H_START 5     // GPIO5
#define SENSOR_V 14       // GPIO14
#define BOMBA_VERTICAL 12 // GPIO12
#define BTN_V_START 13    // GPIO13
```

- sueloPin: Entrada analógica para sensor de humedad de suelo.
- DHTPIN: Pin digital donde está conectado el sensor DHT22.
- DHTTYPE: Tipo de sensor DHT.
- dht: Instancia del sensor DHT para su uso posterior.
- **Sensores de nivel líquido:**
  SENSOR_H_MAX: Detecta nivel máximo en el tanque horizontal.
  SENSOR_H_MIN: Detecta nivel mínimo.
  SENSOR_V: Detecta si el tanque vertical está lleno o vacío.
- **Bombas:**
  BOMBA_HORIZONTAL: Controla llenado del tanque horizontal.
  BOMBA_VERTICAL: Controla llenado del tanque vertical.
- **Botones manuales:**
  BTN_H_START: Enciende/apaga bomba horizontal manualmente.
  BTN_V_START: Enciende/apaga bomba vertical manualmente.

3. Configuración Inicial

```
void setup() {
  // Inicializar DHT22
  dht.begin();

  // Configurar pines de sensores de nivel
  pinMode(SENSOR_H_MAX, INPUT_PULLUP);
  pinMode(SENSOR_H_MIN, INPUT_PULLUP);
  pinMode(BOMBA_HORIZONTAL, OUTPUT);
  pinMode(BTN_H_START, INPUT_PULLUP);
  pinMode(SENSOR_V, INPUT_PULLUP);
  pinMode(BOMBA_VERTICAL, OUTPUT);
  pinMode(BTN_V_START, INPUT_PULLUP);

  // Inicializar bombas apagadas
  digitalWrite(BOMBA_HORIZONTAL, LOW);
  digitalWrite(BOMBA_VERTICAL, LOW);

  setup_wifi();
  client.setServer(MQTT_SERVER, MQTT_PORT);
  client.setCallback(callback);
}
```

- Se configuran pines como entrada (con pull-up) o salida.
- Las bombas inician apagadas.

4. Funciones de Lectura de Nivel de Agua

```
98  String leerHorizontal() {
99    if (digitalRead(SENSOR_H_MAX) == LOW) return "MAXIMO";
100   else if (digitalRead(SENSOR_H_MIN) == LOW) return "MINIMO";
101   else return "CRITICO";
102 }
103
104 String leerVertical() {
105   return (digitalRead(SENSOR_V) == LOW) ? "VACIO" : "LLENO";
106 }
```

   - Lee sensores SENSOR_H_MAX y SENSOR_H_MIN.
   - Retorna:
     - "MAXIMO" si el nivel llegó arriba.
     - "MINIMO" si el nivel está bajo.
     - "CRITICO" si no se detecta ninguno, probablemente vacío.
     - Retorna "VACIO" si el sensor vertical está activado (LOW), o "LLENO" si no lo está.

5. Lógica de Control de Bombas

```
void controlBombas() {
  // Control automático sistema horizontal
  String estadoH = leerHorizontal();
  if (estadoH == "CRITICO") digitalWrite(BOMBA_HORIZONTAL, HIGH);
  else if (estadoH == "MAXIMO") digitalWrite(BOMBA_HORIZONTAL, LOW);

  // Control automático sistema vertical
  String estadoV = leerVertical();
  if (estadoV == "VACIO") digitalWrite(BOMBA_VERTICAL, HIGH);
  else digitalWrite(BOMBA_VERTICAL, LOW);
}
```

   - Automáticamente enciende o apaga las bombas según los sensores:
   - Si el tanque horizontal está en estado "CRITICO" → bomba encendida.
   - Si está en "MAXIMO" → bomba apagada.
   - Si el vertical está "VACIO" → bomba encendida, si no → apagada.

6. Ciclo Principal loop()

```
138 void loop() {
139   if (!client.connected()) reconnect();
140   client.loop();
141
142   // --- Lectura de sensores ---
143   // 1. Humedad del suelo
144   int sueloValue = analogRead(sueloPin);
145   String sueloEstado = (sueloValue > 1488) ? "SECO" : "HUMEDO";
146   String payloadSuelo = "{\"valor\":" + String(sueloValue) + ",\"estado\":\"" + sueloEstado + "\"}";
147
148   // 2. DHT22 (temperatura y humedad ambiente)
149   sensors_event_t temp_event, hum_event;
150   dht.temperature().getEvent(&temp_event);
151   dht.humidity().getEvent(&hum_event);
152   float temperatura = temp_event.temperature;
153   float humedad = hum_event.relative_humidity;
154   String payloadAmbiente = "{\"temperatura\":" + String(temperatura, 1) + ",\"humedad\":" + String(humedad, 1) + "}";
155
156   // 3. Sensores de nivel líquido
157   String nivelH = leerHorizontal();
158   String nivelV = leerVertical();
159   String payloadNivel = "{\"horizontal\":\"" + nivelH + "\",\"vertical\":\"" + nivelV + "\"}";
160
161   // --- Log en Serial Monitor ---
162   Serial.println("📤 Enviados:");
163   Serial.println("- Tópico 'iot/suelo': " + payloadSuelo);
164   Serial.println("- Tópico 'iot/ambiente': " + payloadAmbiente);
165   Serial.println("- Tópico 'iot/nivel': " + payloadNivel);
166
167   // --- Control de bombas y botones (igual que antes) ---
168   controlBombas();
169   if (digitalRead(BTN_H_START) == LOW) {
170     digitalWrite(BOMBA_HORIZONTAL, !digitalRead(BOMBA_HORIZONTAL));
171     delay(300);
172   }
173   if (digitalRead(BTN_V_START) == LOW) {
174     digitalWrite(BOMBA_VERTICAL, !digitalRead(BOMBA_VERTICAL));
175     delay(300);
176   }
177
178   delay(5000); // Espera 5 segundos entre lecturas
179 }
```

Primero se hace la lectura del sensor de humedad del suelo

```
138 void loop() {
139   if (!client.connected()) reconnect();
140   client.loop();
141
142   // --- Lectura de sensores ---
143   // 1. Humedad del suelo
144   int sueloValue = analogRead(sueloPin);
145   String sueloEstado = (sueloValue > 1488) ? "SECO" : "HUMEDO";
146   String payloadSuelo = "{\"valor\":" + String(sueloValue) + ",\"estado\":\"" + sueloEstado + "\"}";
147
```

- Lectura del sensor de humedad del suelo (valor ADC 0-4095).
- Clasifica como "SECO" o "HUMEDO" según un umbral de 1488.

Luego Obtiene los valores actuales de temperatura y humedad ambiente desde el DHT22.

```
148   // 2. DHT22 (temperatura y humedad ambiente)
149   sensors_event_t temp_event, hum_event;
150   dht.temperature().getEvent(&temp_event);
151   dht.humidity().getEvent(&hum_event);
152   float temperatura = temp_event.temperature;
153   float humedad = hum_event.relative_humidity;
154   String payloadAmbiente = "{\"temperatura\":" + String(temperatura, 1) + ",\"humedad\":" + String(humedad, 1) + "}";
155
```

Lee los estados de los sensores de nivel de agua.

```
// 3. Sensores de nivel líquido
String nivelH = leerHorizontal();
String nivelV = leerVertical();
String payloadNivel = "{\"horizontal\":\"" + nivelH + "\",\"vertical\":\"" + nivelV + "\"}";
```

Aplica lógica de control automático de bombas.

```
  // --- Log en Serial Monitor ---
  Serial.println("📤 Enviados:");
  Serial.println("- Tópico 'iot/suelo': " + payloadSuelo);
  Serial.println("- Tópico 'iot/ambiente': " + payloadAmbiente);
  Serial.println("- Tópico 'iot/nivel': " + payloadNivel);

  // --- Control de bombas y botones ---
  controlBombas();
  if (digitalRead(BTN_H_START) == LOW) {
    digitalWrite(BOMBA_HORIZONTAL, !digitalRead(BOMBA_HORIZONTAL));
    delay(300);
  }
  if (digitalRead(BTN_V_START) == LOW) {
    digitalWrite(BOMBA_VERTICAL, !digitalRead(BOMBA_VERTICAL));
    delay(300);
  }

  delay(5000); // Espera 5 segundos entre lecturas
}
```

- Si se presiona el botón correspondiente, se cambia el estado de la bomba horizontal (ON/OFF).
- Similar lógica para el botón vertical.
- Espera 5 segundos entre cada ciclo de lectura y control.

DOCUMENTACION CAMARA

1. Primero se importó las librerías

```
#include <WiFi.h>
#include "esp_camera.h"
#include "mbedtls/base64.h"
```

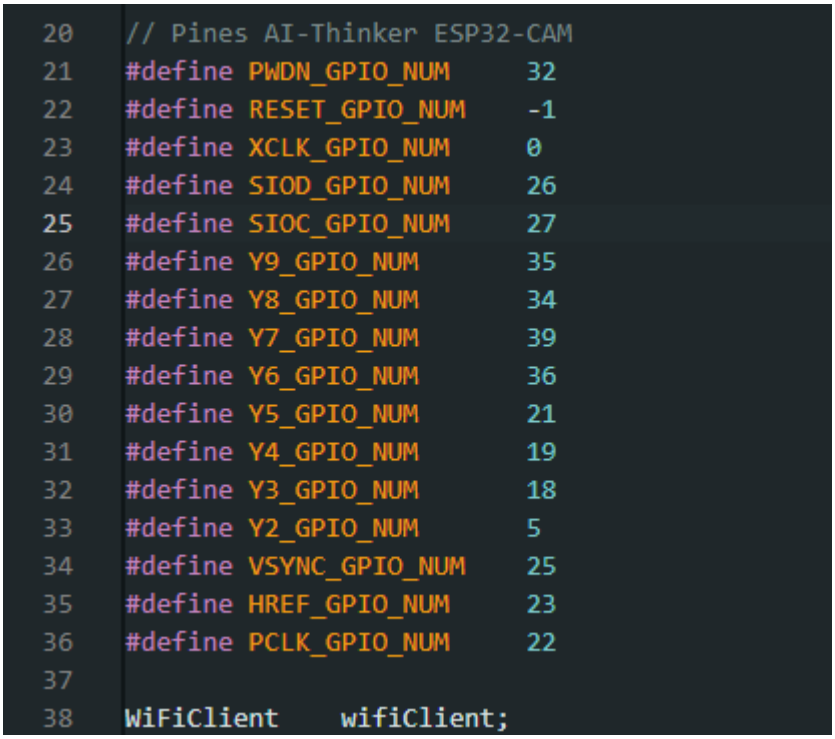
- WiFi.h: Permite conectar el ESP32 a una red Wi-Fi.
- esp_camera.h: Librería oficial de Espressif para manejar la cámara del ESP32-CAM.
- mbedtls/base64.h: Usada para codificar la imagen capturada en formato Base64 (útil para transferencia de datos).

2. Configuración Wi-Fi

```
const char* WIFI_SSID     = "USCO_CENTRAL";
const char* WIFI_PASS     = "";  // tu clave
```

- En setup(), se establece la conexión Wi-Fi mediante WiFi.begin() y un bucle de espera hasta que se logre la conexión.

3. Configuración de Pines de Cámara (AI-Thinker ESP32-CAM)



```
20  // Pines AI-Thinker ESP32-CAM
21  #define PWDN_GPIO_NUM     32
22  #define RESET_GPIO_NUM    -1
23  #define XCLK_GPIO_NUM     0
24  #define SIOD_GPIO_NUM     26
25  #define SIOC_GPIO_NUM     27
26  #define Y9_GPIO_NUM       35
27  #define Y8_GPIO_NUM       34
28  #define Y7_GPIO_NUM       39
29  #define Y6_GPIO_NUM       36
30  #define Y5_GPIO_NUM       21
31  #define Y4_GPIO_NUM       19
32  #define Y3_GPIO_NUM       18
33  #define Y2_GPIO_NUM       5
34  #define VSYNC_GPIO_NUM    25
35  #define HREF_GPIO_NUM     23
36  #define PCLK_GPIO_NUM     22
37
38  WiFiClient    wifiClient;
```

4. Inicialización de la Cámara

```
camera_config_t cfg;
cfg.ledc_channel = LEDC_CHANNEL_0;
cfg.ledc_timer   = LEDC_TIMER_0;
cfg.pin_d0       = Y2_GPIO_NUM;
cfg.pin_d1       = Y3_GPIO_NUM;
cfg.pin_d2       = Y4_GPIO_NUM;
cfg.pin_d3       = Y5_GPIO_NUM;
cfg.pin_d4       = Y6_GPIO_NUM;
cfg.pin_d5       = Y7_GPIO_NUM;
cfg.pin_d6       = Y8_GPIO_NUM;
cfg.pin_d7       = Y9_GPIO_NUM;
cfg.pin_xclk     = XCLK_GPIO_NUM;
cfg.pin_pclk     = PCLK_GPIO_NUM;
cfg.pin_vsync    = VSYNC_GPIO_NUM;
cfg.pin_href     = HREF_GPIO_NUM;
cfg.pin_sscb_sda = SIOD_GPIO_NUM;
cfg.pin_sscb_scl = SIOC_GPIO_NUM;
cfg.pin_pwdn     = PWDN_GPIO_NUM;
cfg.pin_reset    = RESET_GPIO_NUM;
cfg.xclk_freq_hz = 20000000;
cfg.pixel_format = PIXFORMAT_JPEG;
cfg.frame_size   = FRAMESIZE_QVGA;
cfg.jpeg_quality = 12;
cfg.fb_location  = CAMERA_FB_IN_PSRAM;
cfg.grab_mode    = CAMERA_GRAB_WHEN_EMPTY;
cfg.fb_count     = psramFound() ? 2 : 1;

if (esp_camera_init(&cfg) != ESP_OK) {
  Serial.println("❌ Error al iniciar cámara");
  while (true) delay(1000);
}
Serial.println("✓ Cámara inicializada");
```

Se crea una estructura camera_config_t que contiene:

- Pines específicos de conexión.
- Frecuencia del reloj de la cámara.
- Formato de imagen (PIXFORMAT_JPEG).
- Tamaño del frame (QVGA, 320x240 píxeles).
- Calidad JPEG y número de framebuffers (usa 2 si hay PSRAM).

La cámara se inicia con esp_camera_init(&cfg). Si falla, el sistema se detiene con un bucle infinito.

5. Gestión de Memoria en PSRAM

```
100     b64buf_len = 16384;
101     b64buf = (char*)ps_malloc(b64buf_len);
102     if (!b64buf) {
103       Serial.println("❌ No hay PSRAM para b64buf");
104       while (true) delay(1000);
105     }
```

   - Se reserva memoria en la PSRAM para almacenar la imagen codificada en Base64 antes de su envío.
   - Si no hay suficiente memoria, se imprime un error y se detiene el sistema.

6. Función de Captura y Codificación

```
142 void takeAndSendPhoto() {
143   Serial.println("▶ Capturando foto…");
144   camera_fb_t* fb = esp_camera_fb_get();
145   if (!fb) {
146     Serial.println("❌ Falló captura");
147     return;
148   }
149
150   // Codificar JPEG a Base64
151   size_t out_len = b64buf_len;
152   if (mbedtls_base64_encode((unsigned char*)b64buf, b64buf_len, &out_len, fb->buf, fb->len) != 0) {
153     Serial.println("❌ Error Base64");
154     esp_camera_fb_return(fb);
155     return;
156   }
157   esp_camera_fb_return(fb);
```

takeAndSendPhoto() realiza los siguientes pasos:

   - Captura de Imagen
   - Codificación en Base64
   - Liberación del Framebuffer