ANEXO 1

CODIGO ESP32 PARA ARDUINO IDE

```
// Habilitacion de sensores para medición de caracteristicas
//    - Clima
//       1. MAX6675
//       2. DHT11
//       3. BMP280
//       4. MQ135
//       5. WH-SP-WS01
//    - Agua
//       6. Nivel genérico
//       7. DS18B20
//       8. TSW-20M
//       9. SEN0244
const bool habilitar_sensor[10] = { NULL, true, true, true, true, true, true, true, true, true };

// Habilitacion de acuadores
//       1. ELECTROVALVULA
//       2. MOTORES
const bool habilitar_actuador[3] = { NULL, true, false };

// Habilitacion de comunicacion
//       1. SERIAL
//       2. WIFI
const bool habilitar_comunicacion[3] = { NULL, false, true };

// Habilitacion de multitarea
//       1. Toma y envio de datos, actuador electrovalvula
//       2. Sensor y actuador de cortina

// Declaracion de librerias adicionales
#include <SPI.h>

// Declaración SENSOR ID 1 (MAX6675)(CLIMA)
#include <max6675.h>
#define MAX6675_SCK 5
#define MAX6675_SC 18
#define MAX6675_DO 19
MAX6675 Sensor1(MAX6675_SCK, MAX6675_SC, MAX6675_DO);

// Declaración SENSOR ID 2 (DHT11)(CLIMA)
#include <DHT.h>
#define DHT11_DATA 23
#define DHTYPE DHT11
DHT Sensor2(DHT11_DATA, DHTYPE);

// Declaración SENSOR ID 3 (BMP280)(CLIMA)
#include <Adafruit_BMP280.h>
Adafruit_BMP280 Sensor3;
// BMP280 I2C   SCL =  22, SDA = 21, SDO = 3.3V

// Declaración SENSOR ID 4 (MQ135)(CLIMA)
#include <MQ135.h>
#define MQ135_DO 34
MQ135 Sensor4(MQ135_DO, 92, 60);
```

```
// Declaración SENSOR ID 5 (WH SP WS01)(CLIMA)
#define Sensor5 32
unsigned long timeold = 0;
volatile unsigned int pulsos = 0;
float ppv = 2.4;

// Declaración SENSOR ID 6 (NIVEL GENÉRICO)(AGUA)
#define Sensor6 33

// Declaración SENSOR ID 7 (DS18B20)(AGUA)
#include <OneWire.h>
#include <DallasTemperature.h>
#define DS18B20 15
OneWire ourWire(DS18B20);
DallasTemperature Sensor7(&ourWire);

// Declaración SENSOR ID 8 (TSW-20M)(AGUA)
#define Sensor8 35

// Declaración SENSOR ID 9 (SEN0244)(AGUA) ------
#define Sensor9 26

// Declaración de variables para toma de datos
//   - Clima
//      1.  Temperatura exterior (1  MAX6675)
//      2.  Humedad (2 - DHT11)
//      3.  Temperatura interior 1 (2  DHT11)
//      4.  Temperatura interior 2 (3  BMP280)
//      5.  Presión atmosférica (3  BMP280)
//      6.  Altitud (3  BMP280)
//      7.  Particulas de CO2 (4  MQ135)
//      8.  Velocidad del viento (5  WH-SP-WS01)
//   - Agua
//      9.  Nivel del Agua (6  Nivel Genérico)
//      10. Temperatura del Agua (7  DS18B20)
//      11. Turbidez del Agua (8  TSW-20M)
//      12. Conductividad del Agua(9  SEN0244)
float error[13] = { NULL, -1.01, 4.8, -0.58, -0.69, 0.0, 200.0, 0.0, 0.0, 8.35178902, 0.72, 0.0, -4.3 };
float dato[13] = { NULL, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
unsigned int const MuestrasPorDato = 10;
unsigned const DelayMuestrasPorDato = 500;
unsigned long const tiempo_entre_mediciones = 5 * 60000;  // Aprox 5 min sistema

// Declaración de variables para ACTUADOR ID 2 (ELECTROVALVULA)
#define EL 25

// Declaración de variables para ACTUADOR ID 1 (PUENTE H L298)
#define IN1 14
#define IN2 27
#define IN3 12
#define IN4 13
```

```
// Declaración SENSOR ID 10 (TCRT5000)(CORTINA)
#define VoltageCortina 4
#define SensorCortina 2
const int referencia_motores1 = 580;
const int referencia_motores2 = 580;
volatile unsigned int pulsos_motores = 0;

// Declaración de variables y constantes para actuadores
unsigned long TiempoElectrovalvula = 10000;
unsigned long TiempoReferencia = 0;
unsigned const NivelLectura = 135;
unsigned const NivelApertura = 160;
bool ElectrovalvulaActivacion = false;

// Declaracion de modulo de COMUNICACION
#define PowerWifi 17
#define SearchSignalModem 16
#include <WiFi.h>
const char* ssid = "AN-USTA TUNJA";  //AN-USTA TUNJA
const char* pass = "261020CP";       // 261020CP
const IPAddress remote_ip(8, 8, 8, 8);

// Declaracion de nube para datos con MATLAB
#include <ThingSpeak.h>
unsigned long ChannelAmbiente = 2421782;
const char* KeyChannelAmbiente = "XFIS11W0C95JB3XN";
unsigned long ChannelAgua = 2411580;
const char* KeyChannelAgua = "IMNRC8VB5EIF3XPZ";
unsigned long ChannelSettings = 2424123;
const char* KeyChannelSettingsWrite = "NXSAUKG43EERHAL5";
const char* KeyChannelSettingsRead = "1FTERFIDDWGI9O2B";
WiFiClient client;

// Declaración de RTC para tiempo del ESP32
#include <ESP32Time.h>
ESP32Time rtc;
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = -18000;
const int daylightOffset_sec = 3600;

// Variables globales del sistema
#include <Preferences.h>
Preferences preferences;
int configuracion = 0;
unsigned int HoraESP = 0;
unsigned int MinutosESP = 0;
unsigned int Hora1 = 0;
unsigned int Hora2 = 0;
char texto[70];
float valor = 0;
float formula = 0;

// Configuración inicial
```

```
void setup() {
  //Inicialización puerto serial
  if (habilitar_comunicacion[1] == true) {
    Serial.begin(9600);
    delay(1000);
    Serial.println("Comunicacion 1: ON");
  }
  delay(500);

  // Inicialización de ACTUADOR 1 ELECTROVALVULA
  if (habilitar_actuador[1] == true) {
    pinMode(EL, OUTPUT);
    digitalWrite(EL, true);
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Actuador 1: OK");
    }
  } else {
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Actuador 1: NO OK");
    }
  }
  delay(500);

  // Inicialización de ACTUADOR ID 2 MOTORES
  if (habilitar_actuador[2] == true) {
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(VoltageCortina, OUTPUT);
    pinMode(SensorCortina, INPUT_PULLUP);
    digitalWrite(IN1, false);
    digitalWrite(IN2, false);
    digitalWrite(IN3, false);
    digitalWrite(IN4, false);
    digitalWrite(VoltageCortina, true);
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Actuador 2: OK");
    }
  } else {
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Actuador 2: NO OK");
    }
    digitalWrite(VoltageCortina, true);
  }
  delay(500);

  // Inicialización SENSOR 1 MAX6675 (CLIMA)
  if (habilitar_sensor[1] == true) {
    // No aplica
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 1: ON");
    }
```

```
  } else {
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 1: OFF");
    }
  }
  delay(500);

  // Inicialización SENSOR 2 DHT11 (CLIMA)
  if (habilitar_sensor[2] == true) {
    Sensor2.begin();
    delay(500);
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 2: ON");
    }
  } else {
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 2: OFF");
    }
  }
  delay(500);

  // Inicialización SENSOR 3 BMP280 (CLIMA)
  if (habilitar_sensor[3] == true) {
    Sensor3.begin();
    delay(500);
    Sensor3.setSampling(Adafruit_BMP280::MODE_NORMAL, Adafruit_BMP280::SAMPLING_X2,
Adafruit_BMP280::SAMPLING_X16, Adafruit_BMP280::FILTER_X16, Adafruit_BMP280::STANDBY_MS_500);
/* Tiempo Standby. */
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 3: ON");
    }
  } else {
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 3: OFF");
    }
  }
  delay(500);

  // Inicialización SENSOR 4 MQ135 (CLIMA)
  if (habilitar_sensor[4] == true) {
    // No aplica
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 4: ON");
    }
  } else {
    if (habilitar_comunicacion[1] == true) {
      Serial.println("Sensor 4: OFF");
    }
  }
  delay(500);

  // Inicialización SENSOR 5 WH-SP-WS01 (CLIMA)
  if (habilitar_sensor[5] == true) {
```

```
  pinMode(Sensor5, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(Sensor5), contador_pulsos, RISING);
  timeold = millis();
  delay(500);
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 5: ON");
  }
} else {
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 5: OFF");
  }
}
delay(500);

// Inicialización SENSOR 6 Nivel Genérico (AGUA)
if (habilitar_sensor[6] == true) {
  pinMode(Sensor6, INPUT);
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 6: ON");
  }
} else {
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 6: OFF");
  }
}
delay(500);

// Inicialización SENSOR 7 DS18B20 (AGUA)
if (habilitar_sensor[7] == true) {
  Sensor7.begin();
  delay(500);
  pinMode(DS18B20, INPUT_PULLUP);
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 7: ON");
  }
} else {
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 7: OFF");
  }
}
delay(500);

// Inicialización SENSOR 8 TSW-20M (AGUA)
if (habilitar_sensor[8] == true) {
  pinMode(Sensor8, INPUT);
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 8: ON");
  }
} else {
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 8: OFF");
  }
}
```

```
delay(500);

// Inicialización SENSOR 9 SEN0244 (AGUA)
if (habilitar_sensor[9] == true) {
  pinMode(Sensor9, INPUT_PULLUP);
  delay(500);
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 9: ON");
  }
} else {
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Sensor 9: OFF");
  }
}
delay(500);

//Inicialización Wifi
if (habilitar_comunicacion[2] == true) {
  pinMode(PowerWifi, OUTPUT);
  pinMode(SearchSignalModem, OUTPUT);
  digitalWrite(PowerWifi, true);
  digitalWrite(SearchSignalModem, true);
  conectar_wifi();
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Comunicacion 2: ON");
  }
} else {
  if (habilitar_comunicacion[1] == true) {
    Serial.println("Comunicacion 2: OFF");
  }
}
delay(1000);

// Configuracion de conexión WIFI (Hora)
if (habilitar_comunicacion[2] == true) {
  int x = 0;
  while (x != 200) {
    ThingSpeak.begin(client);
    sprintf(texto, "[ESP32] Actualizando zona horaria");
    ThingSpeak.setStatus(texto);
    x = ThingSpeak.writeFields(ChannelSettings, KeyChannelSettingsWrite);
    delay(5000);
    if (x != 200) {
      signal_modem();
    } else {
      configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
      struct tm timeinfo;
      if (getLocalTime(&timeinfo)) {
        rtc.setTimeStruct(timeinfo);
      }
      delay(500);
    }
  }
```

```
  if (habilitar_comunicacion[1] == true) {
   Serial.println(texto);
  }

  x = 0;
  while (x != 200) {
   ThingSpeak.begin(client);
   HoraESP = rtc.getHour(true);
   MinutosESP = rtc.getMinute();
   sprintf(texto, "[ESP32] Inicializado %U hrs %U min", HoraESP, MinutosESP);
   ThingSpeak.setStatus(texto);
   delay(2000);
   x = ThingSpeak.writeFields(ChannelSettings, KeyChannelSettingsWrite);
   delay(5000);
   if (x != 200) {
    signal_modem();
   }
  }
  if (habilitar_comunicacion[1] == true) {
   Serial.println(texto);
  }
 }
 desconectar_wifi();

 configuracion = 1;
 delay(2000);
}

void loop() {
 bool checking = true;

 // Toma de DATO 1 Temperatura exterior (SENSOR 1 MAX6675)
 if (configuracion == 1 && habilitar_sensor[1] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
   checking = true;
   while (checking) {
    valor = Sensor1.readCelsius();
    formula = valor;
    checking = rectificacion(formula);
   }
   dato[configuracion] = dato[configuracion] + formula;
   delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
   Serial.println("DATO 1: OK");
  }
 }

 // Toma de DATO 2 Humedad (SENSOR 2 DHT11)
 else if (configuracion == 2 && habilitar_sensor[2] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
   checking = true;
```

```
    while (checking) {
      valor = Sensor2.readHumidity();
      formula = valor;
      checking = rectificacion(formula);
    }
    dato[configuracion] = dato[configuracion] + formula;
    delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
    Serial.println("DATO 2: OK");
  }
}

// Toma de DATO 3 Temperatura interior 1 (SENSOR 2 DHT11)
else if (configuracion == 3 && habilitar_sensor[2] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
    checking = true;
    while (checking) {
      valor = Sensor2.readTemperature();
      formula = valor;
      checking = rectificacion(formula);
    }
    dato[configuracion] = dato[configuracion] + formula;
    delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
    Serial.println("DATO 3: OK");
  }
}

// Toma de DATO 4 Temperatura interior 2 (SENSOR 3 BMP280)
else if (configuracion == 4 && habilitar_sensor[3] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
    checking = true;
    while (checking) {
      valor = Sensor3.readTemperature();
      formula = valor;
      checking = rectificacion(formula);
    }
    dato[configuracion] = dato[configuracion] + formula;
    delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
    Serial.println("DATO 4: OK");
  }
}

// Toma de DATO 5 Presión atmosférica (SENSOR 3 BMP280)
else if (configuracion == 5 && habilitar_sensor[3] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
```

```
    checking = true;
    while (checking) {
      valor = Sensor3.readPressure();
      formula = valor;
      checking = rectificacion(formula);
    }
    dato[configuracion] = dato[configuracion] + formula;
    delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
    Serial.println("DATO 5: OK");
  }
}

// Toma de DATO 6 Altitud (SENSOR 3 BMP280)
else if (configuracion == 6 && habilitar_sensor[3] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
    checking = true;
    while (checking) {
      valor = Sensor3.readAltitude();
      formula = valor;
      checking = rectificacion(formula);
    }
    dato[configuracion] = dato[configuracion] + formula;
    delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
    Serial.println("DATO 6: OK");
  }
}

// Toma de DATO 7 Particulas de CO2 (SENSOR 4 MQ135)
else if (configuracion == 7 && habilitar_sensor[4] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
    checking = true;
    while (checking) {
      valor = Sensor4.getCorrectedPPM(dato[3], dato[2]);
      formula = valor;
      checking = rectificacion(formula);
    }
    dato[configuracion] = dato[configuracion] + formula;
    delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
    Serial.println("DATO 7: OK");
  }
}

// Toma de DATO 8 Velocidad del viento (SENSOR 5 WS-SP-WS01)
else if (configuracion == 8 && habilitar_sensor[5] == true) {
```

```
  int tiempo_trascurrido = millis() - timeold;
  if (tiempo_trascurrido >= 1000) {
   detachInterrupt(digitalPinToInterrupt(Sensor5));
   valor = pulsos;
   formula = (float)valor / ((float)tiempo_trascurrido / 1000.0) * ppv / 3.6;
   attachInterrupt(digitalPinToInterrupt(Sensor5), contador_pulsos, RISING);
   pulsos = 0;
   timeold = millis();
  }
  delay(DelayMuestrasPorDato);
  dato[configuracion] = formula + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
   Serial.println("DATO 8: OK");
  }
 }

 // Toma de DATO 9 Nivel del Agua (SENSOR 6 Nivel Genérico)
 else if (configuracion == 9 && habilitar_sensor[6] == true) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
   checking = true;
   while (checking) {
    valor = analogRead(Sensor6);
    formula = 4095.0 - valor;
    checking = rectificacion(formula);
   }
   dato[configuracion] = dato[configuracion] + formula;
   delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato);
  dato[configuracion] = ((0.00007307 * dato[configuracion] * dato[configuracion]) - (0.03147317 *
dato[configuracion]) + (47.5891487)) + error[configuracion];
  if (dato[configuracion] < 30) {
   dato[configuracion] = 30;
  }
  if (habilitar_comunicacion[1] == true) {
   Serial.println("DATO 9: OK");
  }
 }

 // Toma de DATO 10 Temperatura del Agua (SENSOR 7 DS18B20)
 else if (configuracion == 10 && habilitar_sensor[7] == true && dato[9] >= NivelLectura) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
   checking = true;
   while (checking) {
    Sensor7.requestTemperatures();
    valor = Sensor7.getTempCByIndex(0);
    formula = valor;
    checking = rectificacion(formula);
   }
   dato[configuracion] = dato[configuracion] + formula;
   delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
```

```
  if (habilitar_comunicacion[1] == true) {
   Serial.println("DATO 10: OK");
  }
 }

 // Toma de DATO 11 Turbidez del Agua (SENSOR 8 TSW-20M)
 else if (configuracion == 11 && habilitar_sensor[8] == true && dato[9] >= NivelLectura) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
   checking = true;
   while (checking) {
    valor = analogRead(Sensor8);
    formula = valor;
    checking = rectificacion(formula);
   }
   dato[configuracion] = dato[configuracion] + formula;
   delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = dato[configuracion] / MuestrasPorDato;
  dato[configuracion] = (0.00001863 * dato[configuracion] * dato[configuracion]) - (0.04529214 *
dato[configuracion]) + 27.32840392 + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
   Serial.println("DATO 11: OK");
  }
 }

 // Toma de DATO 12 Conductividad del Agua (SENSOR 9 SEN0244)
 else if (configuracion == 12 && habilitar_sensor[9] == true && dato[9] >= NivelLectura) {
  for (int i = 1; i <= MuestrasPorDato; i++) {
   checking = true;
   while (checking) {
    valor = analogRead(Sensor9);
    formula = valor * 500.0 / 4095.0;
    checking = rectificacion(formula);
   }
   dato[configuracion] = dato[configuracion] + formula;
   delay(DelayMuestrasPorDato);
  }
  dato[configuracion] = (dato[configuracion] / MuestrasPorDato) + error[configuracion];
  if (habilitar_comunicacion[1] == true) {
   Serial.println("DATO 12: OK");
  }
 }

 // Envio de datos Thingspeak por Serial
 else if (configuracion == 13 && habilitar_comunicacion[1] == true) {
  Serial.print("Temperatura exterior (°C) :");
  Serial.println(dato[1]);

  Serial.print("Humedad (%) :");
  Serial.println(dato[2]);

  Serial.print("Temperatura interior 1 (°C) :");
  Serial.println(dato[3]);
```

```
  Serial.print("Temperatura interior 2 (°C) :");
  Serial.println(dato[4]);

  Serial.print("Presión atmosférica (Pa) :");
  Serial.println(dato[5]);

  Serial.print("Altitud (msnm) :");
  Serial.println(dato[6]);

  Serial.print("Particulas de CO2 (ppm) :");
  Serial.println(dato[7]);

  Serial.print("Velocidad del viento (m/s) :");
  Serial.println(dato[8]);

  Serial.print("Nivel del agua (mL) :");
  Serial.println(dato[9]);

  Serial.print("Temperatura del agua (°C) :");
  if (dato[9] < NivelLectura) {
    Serial.println("NA");
  } else {
    Serial.println(dato[10]);
  }

  Serial.print("Turbidez del agua (NTU) :");
  if (dato[9] < NivelLectura) {
    Serial.println("NA");
  } else {
    Serial.println(dato[11]);
  }

  Serial.print("Conductividad del agua (uS/m) :");
  if (dato[9] < NivelLectura) {
    Serial.println("NA");
  } else {
    Serial.println(dato[12]);
  }
  delay(100);
}

// Activación electrovalvula
else if (configuracion == 14 && dato[9] >= NivelApertura && habilitar_actuador[1] == true) {
  digitalWrite(EL, false);
  TiempoReferencia = millis();
  ElectrovalvulaActivacion = true;
}

// Desactivación electrovalvula
else if (configuracion == 15 && habilitar_actuador[1] == true && ElectrovalvulaActivacion == true) {
  while (millis() - TiempoReferencia < TiempoElectrovalvula) {
    delay(200);
```

```
  }
  digitalWrite(EL, true);
  ElectrovalvulaActivacion = false;
}

// Encendido de WIFI
if (configuracion == 16 && habilitar_comunicacion[2] == true) {
  conectar_wifi();
}

// Envio de datos Thingspeak
else if (configuracion == 17 && habilitar_comunicacion[2] == true) {
  // Envío de datos Ambiente
  ThingSpeak.begin(client);
  int x = 0;
  while (x != 200) {
    for (int i = 1; i <= 8; i++) {
      ThingSpeak.setField(i, dato[i]);
      delay(10);
    }
    HoraESP = rtc.getHour(true);
    MinutosESP = rtc.getMinute();
    sprintf(texto, "[ESP32] Datos ambiente enviados %U hrs %U min", HoraESP, MinutosESP);
    ThingSpeak.setStatus(texto);
    delay(100);
    x = ThingSpeak.writeFields(ChannelAmbiente, KeyChannelAmbiente);
    delay(5000);
    if (x != 200) {
      signal_modem();
    }
  }
  ThingSpeak.begin(client);
  x = 0;
  while (x != 200) {
    ThingSpeak.setStatus(texto);
    delay(100);
    x = ThingSpeak.writeFields(ChannelSettings, KeyChannelSettingsWrite);
    delay(5000);
    if (x != 200) {
      signal_modem();
    }
    if (habilitar_comunicacion[1] == true && x == 200) {
      Serial.println(texto);
    }
  }

  // Envío de datos Agua
  ThingSpeak.begin(client);
  x = 0;
  while (x != 200) {
    ThingSpeak.setField(1, dato[9]);
    if (dato[9] >= NivelLectura) {
      for (int i = 10; i <= 12; i++) {
```

```
    ThingSpeak.setField(i - 8, dato[i]);
    delay(10);
   }
  }
  HoraESP = rtc.getHour(true);
  MinutosESP = rtc.getMinute();
  sprintf(texto, "[ESP32] Datos agua enviados %U hrs %U min", HoraESP, MinutosESP);
  ThingSpeak.setStatus(texto);
  delay(100);
  x = ThingSpeak.writeFields(ChannelAgua, KeyChannelAgua);
  delay(10000);
  if (x != 200) {
   signal_modem();
  }
}
ThingSpeak.begin(client);
x = 0;
while (x != 200) {
  ThingSpeak.setStatus(texto);
  x = ThingSpeak.writeFields(ChannelSettings, KeyChannelSettingsWrite);
  delay(5000);
  if (x != 200) {
   signal_modem();
  }
  if (habilitar_comunicacion[1] == true && x == 200) {
   Serial.println(texto);
  }
}

// Toma de datos Horas Cortina
preferences.begin("my-app", false);
Hora1 = preferences.getUInt("Hora1", 6);
Hora2 = preferences.getUInt("Hora2", 18);
float var1 = 0;
float var2 = 0;
unsigned int H1 = 0;
unsigned int H2 = 0;
delay(100);
ThingSpeak.begin(client);
x = 0;
while (x != 200) {
  delay(1000);
  var1 = ThingSpeak.readFloatField(ChannelSettings, 1, KeyChannelSettingsRead);
  var2 = ThingSpeak.readFloatField(ChannelSettings, 2, KeyChannelSettingsRead);
  delay(5000);
  H1 = (int)var1;
  H2 = (int)var2;
  delay(100);
  if ((H1 >= 0 && H1 < 12) && (H2 >= 12 && H2 < 24)) {
   x = 200;
  } else {
   signal_modem();
  }
```

```
   }
   HoraESP = rtc.getHour(true);
   MinutosESP = rtc.getMinute();
   delay(100);
   if ((H1 != Hora1 || H2 != Hora2)) {
    preferences.putUInt("Hora1", H1);
    preferences.putUInt("Hora2", H2);
    delay(1000);
    sprintf(texto, "[ESP32] Horas actualizadas (Hora 1 -> %U:00, Hora 2 -> %U:00) %U hrs %U min", H1, H2,
HoraESP, MinutosESP);
   } else {
    sprintf(texto, "[ESP32] Horas no actualizadas (Hora 1 -> %U:00, Hora 2 -> %U:00) %U hrs %U min", Hora1,
Hora2, HoraESP, MinutosESP);
   }
   ThingSpeak.begin(client);
   x = 0;
   while (x != 200) {
    ThingSpeak.setStatus(texto);
    x = ThingSpeak.writeFields(ChannelSettings, KeyChannelSettingsWrite);
    delay(5000);
    if (x != 200) {
     signal_modem();
     delay(10000);
    }
    if (habilitar_comunicacion[1] == true && x == 200) {
     Serial.println(texto);
    }
   }
  }

  // Funcion de cortina
  else if (configuracion == 18 && habilitar_actuador[2] == true) {
   HoraESP = rtc.getHour(true);
   delay(100);
   preferences.begin("my-app", false);
   unsigned int EstadoCortina = preferences.getUInt("EC", 2);  // 0 -> ARRIBA  1 -> BAJANDO  2 -> ABAJO  3 ->
SUBIENDO
   pulsos_motores = preferences.getUInt("PM", 0);
   Hora1 = preferences.getUInt("Hora1", 6);
   Hora2 = preferences.getUInt("Hora2", 18);
   delay(1000);

   if (EstadoCortina == 0 || EstadoCortina == 2) {
    pulsos_motores = 0;
    preferences.putUInt("PM", pulsos_motores);
   }

   if (habilitar_comunicacion[1] == true) {
    Serial.print("Estado de la Cortina: ");
    Serial.println(EstadoCortina);
    Serial.print("Horas actuales: : ");
    Serial.println(HoraESP);
    Serial.print("Minutos actuales: ");
```

```
  Serial.println(MinutosESP);
  Serial.print("Hora 1: ");
  Serial.println(Hora1);
  Serial.print("Hora 2: ");
  Serial.println(Hora2);
 }

 if (HoraESP >= 0 && HoraESP < 12) {
  if (HoraESP >= Hora1 && EstadoCortina != 0) {
   digitalWrite(VoltageCortina, false);
   StopCortina();
   delay(2000);
   if (habilitar_comunicacion[1] == true) {
    Serial.println("Cortina: SUBIENDO");
   }
   attachInterrupt(digitalPinToInterrupt(SensorCortina), funcion_pulso_motores, RISING);
   EstadoCortina = 3;
   preferences.putUInt("EC", EstadoCortina);  // SUBIENDO
   delay(100);
   while (EstadoCortina != 0) {
    for (int i = 1; i <= 250; i++) {
     SubirCortina();
     if (pulsos_motores >= referencia_motores1) {
      StopCortina();
      EstadoCortina = 0;
      preferences.putUInt("EC", EstadoCortina);
      detachInterrupt(digitalPinToInterrupt(SensorCortina));
      pulsos_motores = 0;
      i = 250;
      delay(100);
     }
     Serial.println(pulsos_motores);
     preferences.putUInt("PM", pulsos_motores);
     delay(2);
    }
    StopCortina();
    delay(500);
   }

   if (habilitar_comunicacion[1] == true) {
    Serial.println("Cortina: ARRIBA");
   }
   delay(2000);
   digitalWrite(VoltageCortina, true);
  } else if (HoraESP < Hora1 && EstadoCortina != 2) {
   digitalWrite(VoltageCortina, false);
   StopCortina();
   delay(2000);
   if (habilitar_comunicacion[1] == true) {
    Serial.println("Cortina: BAJANDO");
   }
   attachInterrupt(digitalPinToInterrupt(SensorCortina), funcion_pulso_motores, RISING);
   EstadoCortina = 1;
```

```
    preferences.putUInt("EC", EstadoCortina);  // BAJANDO
    delay(100);
    while (EstadoCortina != 2) {
     for (int i = 1; i <= 250; i++) {
       BajarCortina();
       if (pulsos_motores >= referencia_motores2) {
         StopCortina();
         EstadoCortina = 2;
         preferences.putUInt("EC", EstadoCortina);
         detachInterrupt(digitalPinToInterrupt(SensorCortina));
         pulsos_motores = 0;
         i = 250;
         delay(100);
       }
       Serial.println(pulsos_motores);
       preferences.putUInt("PM", pulsos_motores);
       delay(2);
     }
     StopCortina();
     delay(500);
    }

    if (habilitar_comunicacion[1] == true) {
     Serial.println("Cortina: ABAJO");
    }
    delay(2000);
    digitalWrite(VoltageCortina, true);
   }
  } else if (HoraESP >= 12 && HoraESP < 24) {
   if (HoraESP >= Hora2 && EstadoCortina != 2) {
    digitalWrite(VoltageCortina, false);
    StopCortina();
    delay(2000);
    if (habilitar_comunicacion[1] == true) {
     Serial.println("Cortina: BAJANDO");
    }
    attachInterrupt(digitalPinToInterrupt(SensorCortina), funcion_pulso_motores, RISING);
    EstadoCortina = 1;
    preferences.putUInt("EC", EstadoCortina);  // BAJANDO
    delay(100);
    while (EstadoCortina != 2) {
     for (int i = 1; i <= 250; i++) {
       BajarCortina();
       if (pulsos_motores >= referencia_motores2) {
         StopCortina();
         EstadoCortina = 2;
         preferences.putUInt("EC", EstadoCortina);
         detachInterrupt(digitalPinToInterrupt(SensorCortina));
         pulsos_motores = 0;
         i = 250;
         delay(100);
       }
       Serial.println(pulsos_motores);
```

```
        preferences.putUInt("PM", pulsos_motores);
        delay(2);
       }
       StopCortina();
       delay(500);
      }

      if (habilitar_comunicacion[1] == true) {
       Serial.println("Cortina: ABAJO");
      }
      delay(2000);
     } else if (HoraESP < Hora2 && EstadoCortina != 0) {
      digitalWrite(VoltageCortina, false);
      StopCortina();
      delay(2000);
      if (habilitar_comunicacion[1] == true) {
       Serial.println("Cortina: SUBIENDO");
      }
      attachInterrupt(digitalPinToInterrupt(SensorCortina), funcion_pulso_motores, RISING);
      EstadoCortina = 3;
      preferences.putUInt("EC", EstadoCortina);  // SUBIENDO
      delay(100);
      while (EstadoCortina != 0) {
       for (int i = 1; i <= 250; i++) {
        SubirCortina();
        if (pulsos_motores >= referencia_motores1) {
         StopCortina();
         preferences.putUInt("EC", 0);
         detachInterrupt(digitalPinToInterrupt(SensorCortina));
         pulsos_motores = 0;
         EstadoCortina = 0;
         i = 250;
         delay(100);
        }
        Serial.println(pulsos_motores);
        preferences.putUInt("PM", pulsos_motores);
        delay(2);
       }
       StopCortina();
       delay(500);
      }

      if (habilitar_comunicacion[1] == true) {
       Serial.println("Cortina: ARRIBA");
      }
      delay(2000);
     }
    }
    digitalWrite(VoltageCortina, true);
    delay(1000);
   }

   else if (configuracion == 19 && habilitar_comunicacion[2] == true) {
```

```
   int x = 0;
   preferences.begin("my-app", false);
   unsigned int EstadoCortina = preferences.getUInt("EC", 2);  // 0 -> ARRIBA  1 -> BAJANDO  2 -> ABAJO  3 ->
SUBIENDO
   char aux[10];
   if(EstadoCortina == 0){
    sprintf(aux, "Arriba");
   }
   else if(EstadoCortina == 2){
    sprintf(aux, "Abajo");
   }
   while (x != 200) {
    ThingSpeak.setField(3, (float)EstadoCortina);
    HoraESP = rtc.getHour(true);
    MinutosESP = rtc.getMinute();
    sprintf(texto, "[ESP32] Estado cortina: %S %U hrs %U min", aux, HoraESP, MinutosESP);
    ThingSpeak.setStatus(texto);
    x = ThingSpeak.writeFields(ChannelSettings, KeyChannelSettingsWrite);
    delay(5000);
    if (x != 200) {
     signal_modem();
     delay(10000);
    }
    if (habilitar_comunicacion[1] == true && x == 200) {
     Serial.println(texto);
    }
   }
   desconectar_wifi();
  }

  // Inicio de ciclo nuevo
  else if (configuracion == 20) {
  delay(tiempo_entre_mediciones);
   for (int i = 1; i <= 12; i++) {
    dato[i] = 0;
   }
   configuracion = 0;
   delay(1000);
  }
  configuracion++;
  delay(100);
}

// Funcion contador de pulsos
void contador_pulsos() {
  pulsos++;
}

void funcion_pulso_motores() {
  pulsos_motores++;
}

//Funcion de rectificacion de datos
```

```
bool rectificacion(float valor) {
  if (isnan(valor)) {
    return true;
  } else {
    return false;
  }
  delay(5);
}

// Funcion para conectar wifi
void conectar_wifi() {
  while (WiFi.status() != WL_CONNECTED) {

    WiFi.begin(ssid, pass);
    WiFi.mode(WIFI_STA);
    delay(2000);

    if (WiFi.status() != WL_CONNECTED) {
      digitalWrite(SearchSignalModem, false);
      delay(500);
      digitalWrite(SearchSignalModem, true);
      delay(2000);

      digitalWrite(PowerWifi, false);
      delay(5000);
      digitalWrite(PowerWifi, true);
      delay(20000);
    }
  }
  if (WiFi.status() == WL_CONNECTED) {
    signal_modem();
  }
}

// Funcion para desconectar wifi
void desconectar_wifi() {
  while (WiFi.status() == WL_CONNECTED) {
    WiFi.disconnect(true);
    WiFi.mode(WIFI_OFF);
    delay(2000);
  }
}

// Función encontrar señal modem
void signal_modem() {
  while (WiFi.status() != WL_CONNECTED) {
    WiFi.begin(ssid, pass);
    WiFi.mode(WIFI_STA);
    delay(2000);

    if (WiFi.status() != WL_CONNECTED) {
      digitalWrite(SearchSignalModem, false);
      delay(500);
```

```
    digitalWrite(SearchSignalModem, true);
    delay(2000);

    digitalWrite(PowerWifi, false);
    delay(5000);
    digitalWrite(PowerWifi, true);
    delay(20000);
   }
  }

  if (WiFi.status() == WL_CONNECTED) {
   digitalWrite(SearchSignalModem, false);
   delay(200);
   digitalWrite(SearchSignalModem, true);
   delay(500);

   digitalWrite(SearchSignalModem, false);
   delay(800);
   digitalWrite(SearchSignalModem, true);
   delay(1000);

   digitalWrite(SearchSignalModem, false);
   delay(800);
   digitalWrite(SearchSignalModem, true);
   delay(20000);
  }
}

// Funcion actualizar horas
void actualizar_hora() {
}

void BajarCortina() {
  digitalWrite(IN1, true);
  digitalWrite(IN2, false);
  digitalWrite(IN3, true);
  digitalWrite(IN4, false);
}

void SubirCortina() {
  digitalWrite(IN1, false);
  digitalWrite(IN2, true);
  digitalWrite(IN3, false);
  digitalWrite(IN4, true);
}

void StopCortina() {
  digitalWrite(IN1, false);
  digitalWrite(IN2, false);
  digitalWrite(IN3, false);
  digitalWrite(IN4, false);
}
```