Escuela Universitaria
Politécnica - La Almunia
Centro adscrito
Universidad Zaragoza

ESCUELA UNIVERSITARIA POLITÉCNICA
DE LA ALMUNIA DE DOÑA GODINA (ZARAGOZA)

ANEXOS

Diseño de sistema de climatización con control adaptativo

Air Conditioning System Design with adaptive Control

424.22.39

Autor: Ignacio Enríquez Romero

Director: Javier Esteban Escaño

Fecha: 20 de Septiembre de 2022

INDICE DE CONTENIDO

1. PLANOS **1**

1.1. PLANOS ELÉCTRICOS 1

1.2. ESQUEMAS ELECTRÓNICOS 2

2. DIAGRAMAS **3**

2.1. DIAGRAMAS UML 3

2.1.1. Diagramas de actividad 3

2.1.2. Diagramas de casos de uso 4

2.2. DIAGRAMAS DE BLOQUES 5

3. CÓDIGO **6**

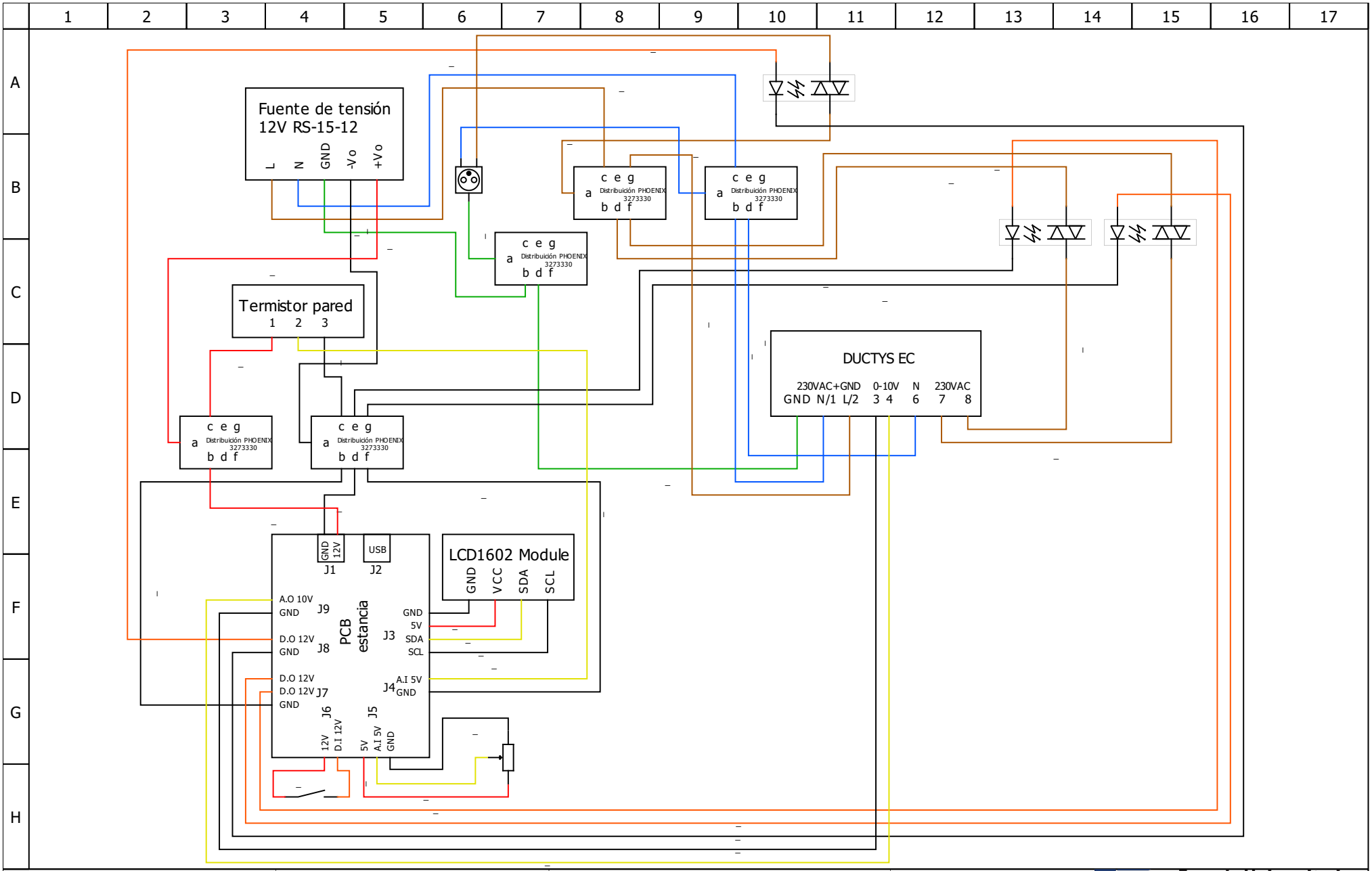
3.1. CONTROL CENTRAL 6

3.2. CONTROL PERIFÉRICO 8

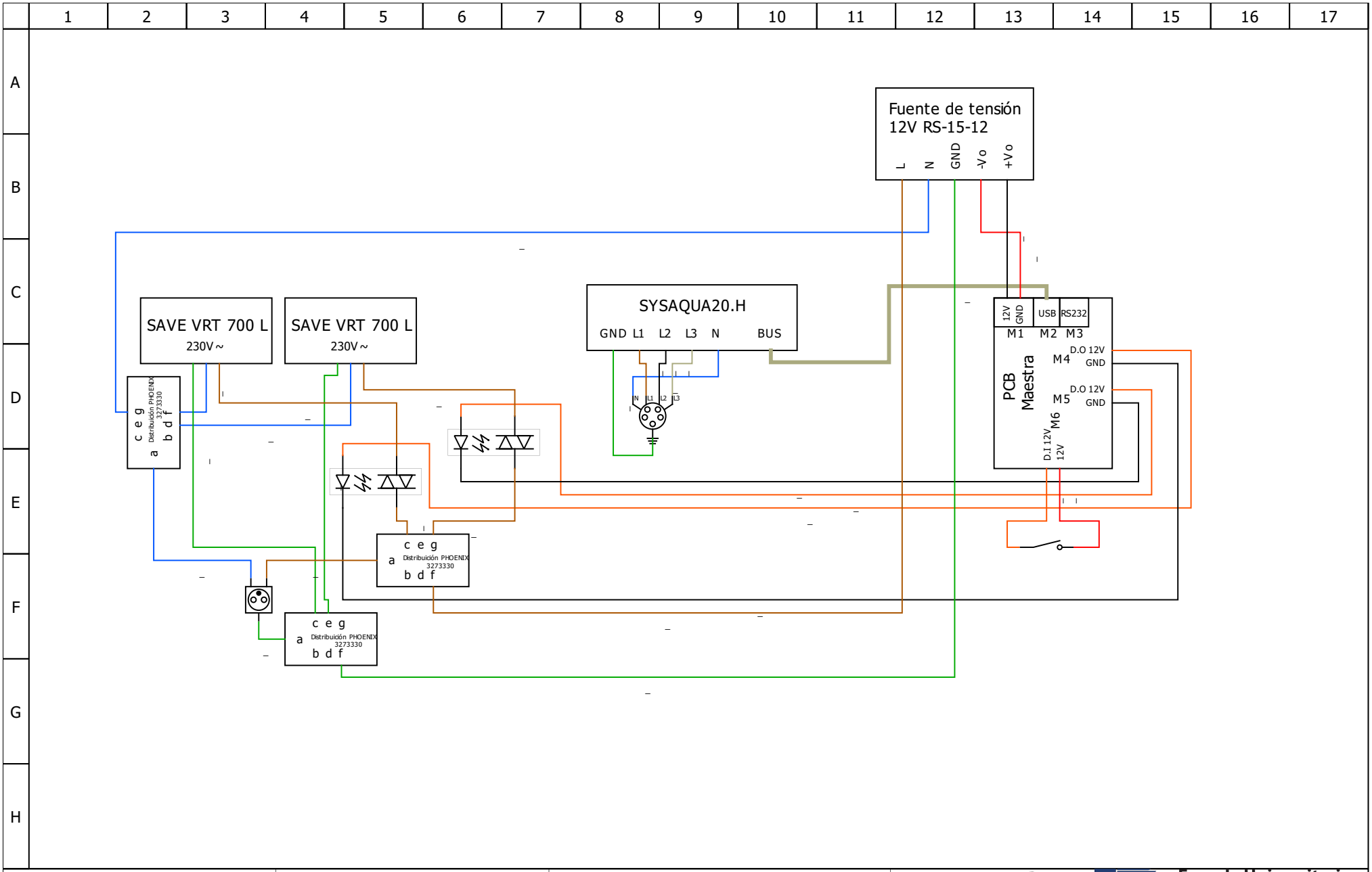
3.3. CÓDIGO DE LA SIMULACIÓN 35

1. PLANOS

1.1. PLANOS ELÉCTRICOS

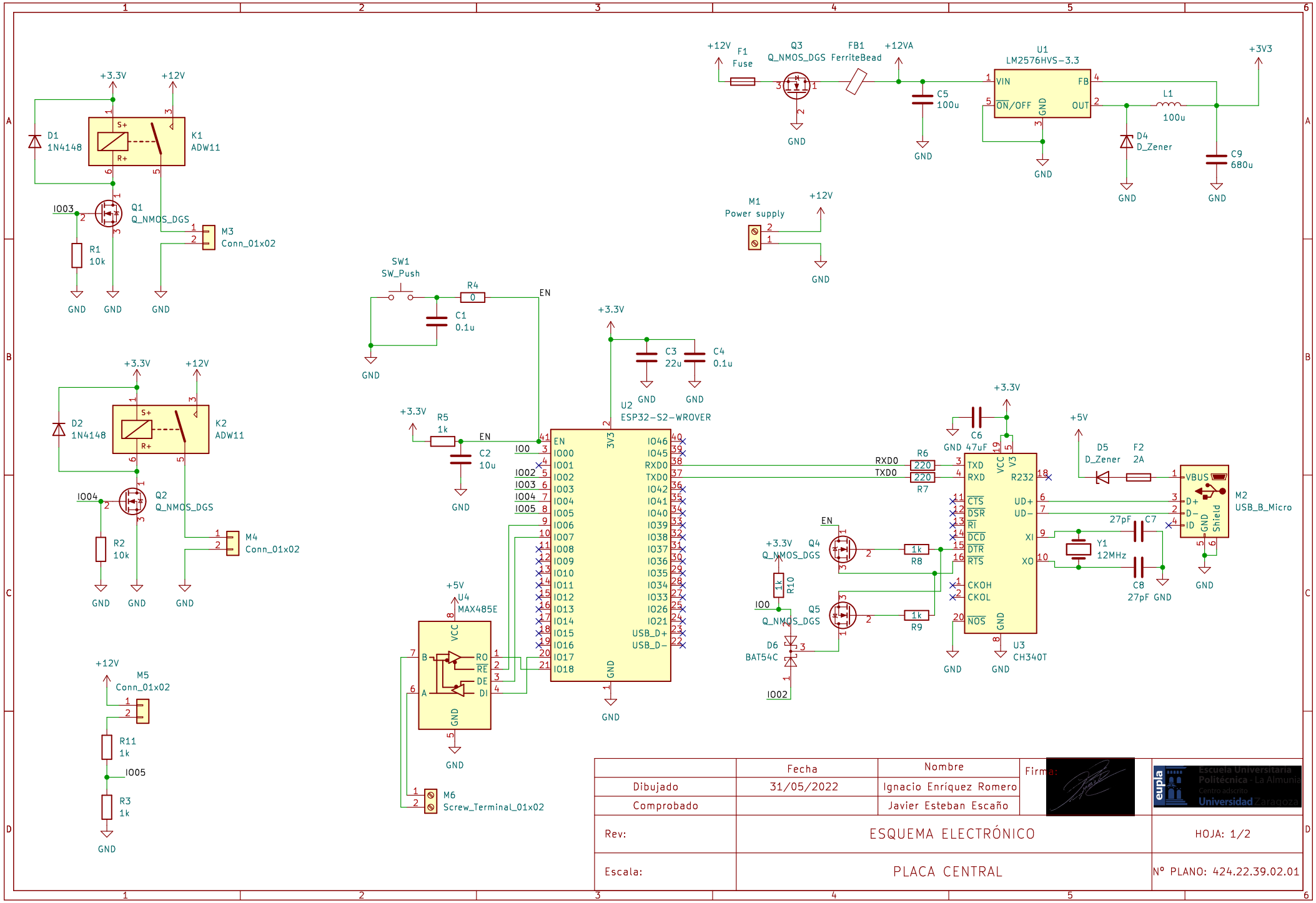


	Fecha	Nombre	
Dibujado	31/05/2022	Ignacio Enríquez Romero	Escuela Universitaria Politécnica - La Almunia, Centro adscrito Universidad Zaragoza
Comprobado		Javier Esteban Escaño	
Rev:	ESQUEMA ELÉCTRICO		Nº PLANO: 424.22.39.01.1
Escala:	Circuito de estancia		HOJA: 1/2



	Fecha	Nombre		
Dibujado	31/05/2022	Ignacio Enríquez Romero		Escuela Universitaria Politécnica - La Almunia Centro adscrito Universidad Zaragoza
Comprobado		Javier Esteban Escaño		
Rev:	ESQUEMA ELÉCTRICO			Nº PLANO: 424.22.39.01.2
Escala:	Circuito principal			HOJA: 2/2

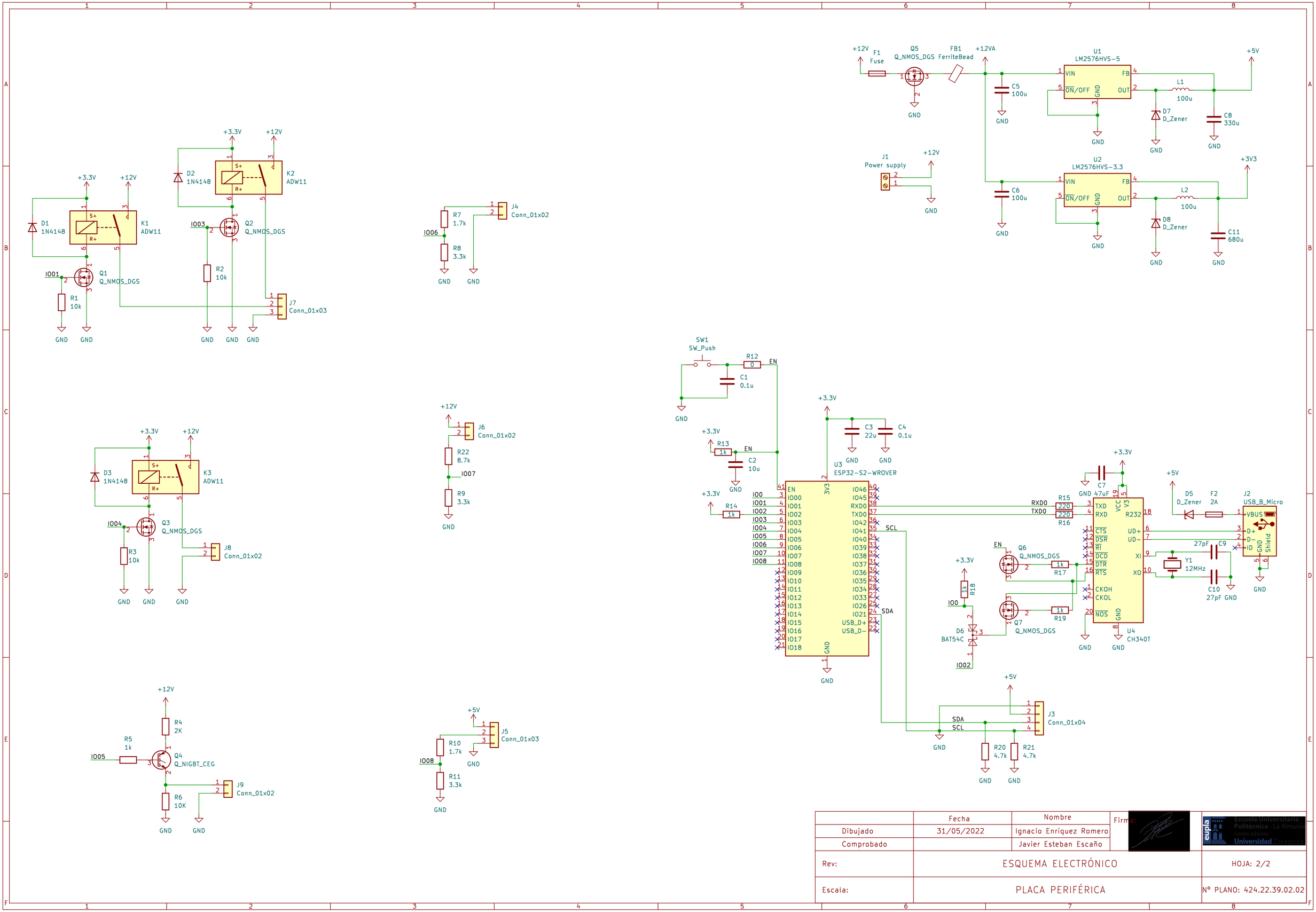
1.2. ESQUEMAS ELECTRÓNICOS



	Fecha	Nombre	Firma:
Dibujado	31/05/2022	Ignacio Enríquez Romero	
Comprobado		Javier Esteban Escaño	

Escuela Universitaria Politécnica - La Almunia
Centro adscrito Universidad Zaragoza

Rev:	ESQUEMA ELECTRÓNICO	HOJA: 1/2
Escala:	PLACA CENTRAL	Nº PLANO: 424.22.39.02.01



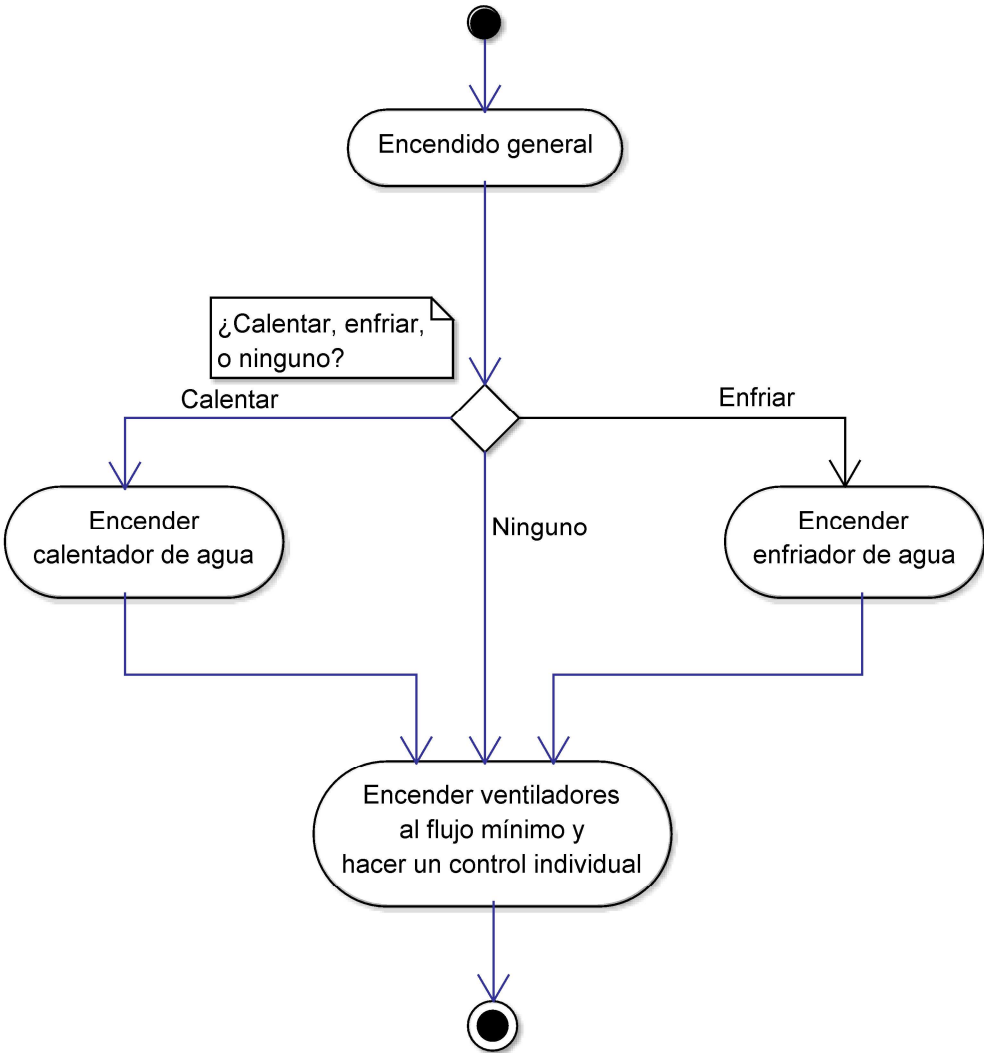
	Fecha	Nombre	Firma:
Dibujado	31/05/2022	Ignacio Enríquez Romero	
Comprobado		Javier Esteban Escaño	
Rev:	ESQUEMA ELECTRÓNICO		HOJA: 2/2
Escala:	PLACA PERIFÉRICA		Nº PLANO: 424.22.39.02.02

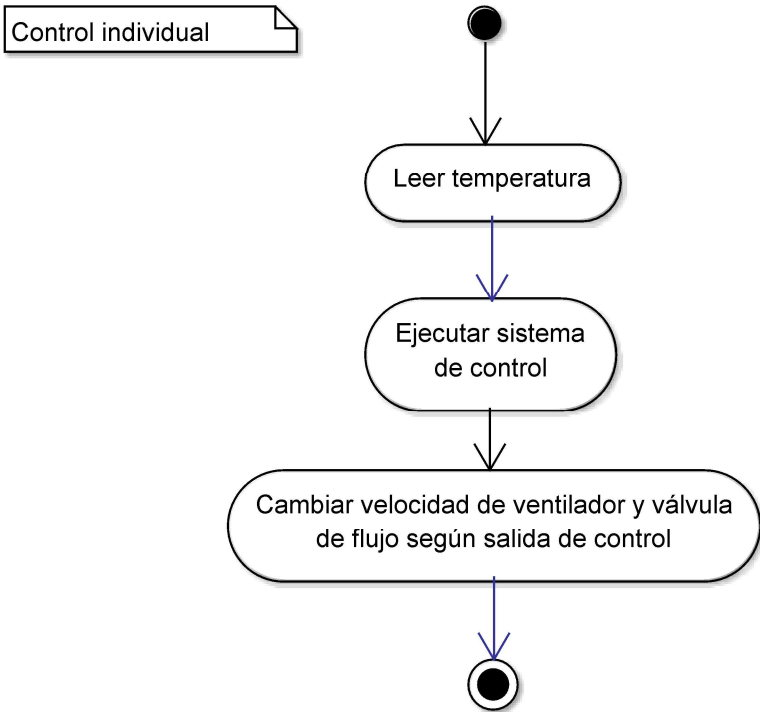
Escuela Universitaria Politécnica - La Almunia
Centro adscrito Universidad Zaragoza

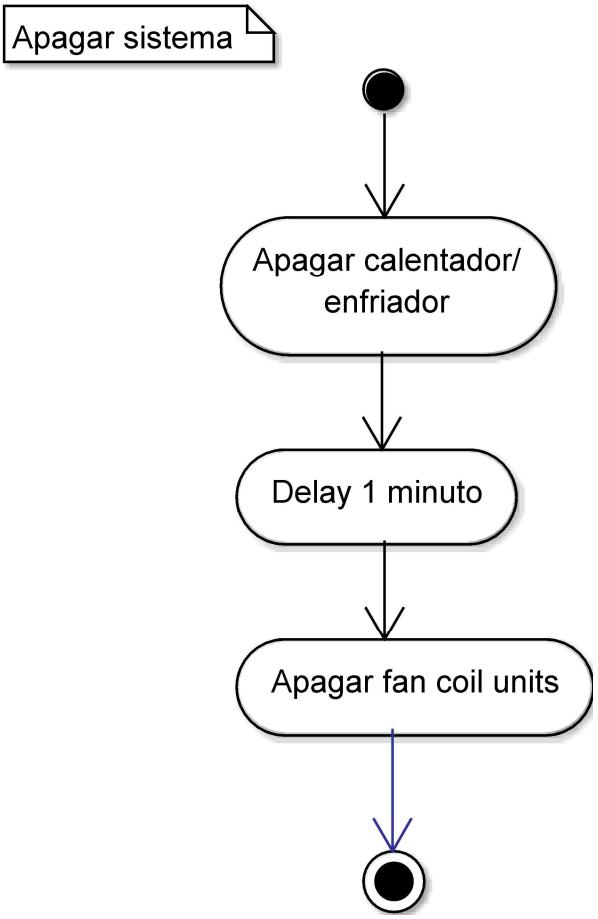
2. DIAGRAMAS

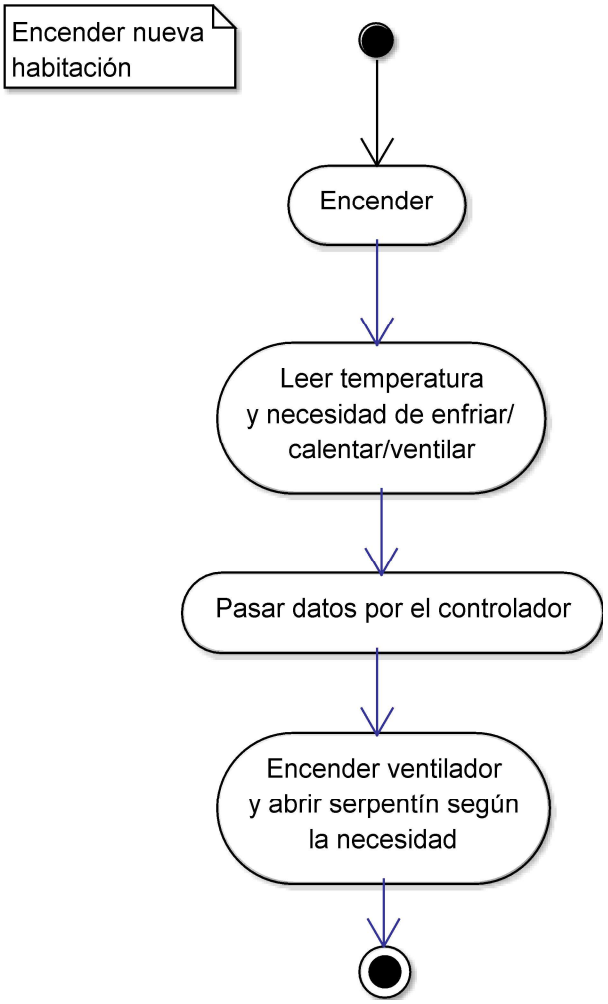
2.1. DIAGRAMAS UML

2.1.1. Diagramas de actividad

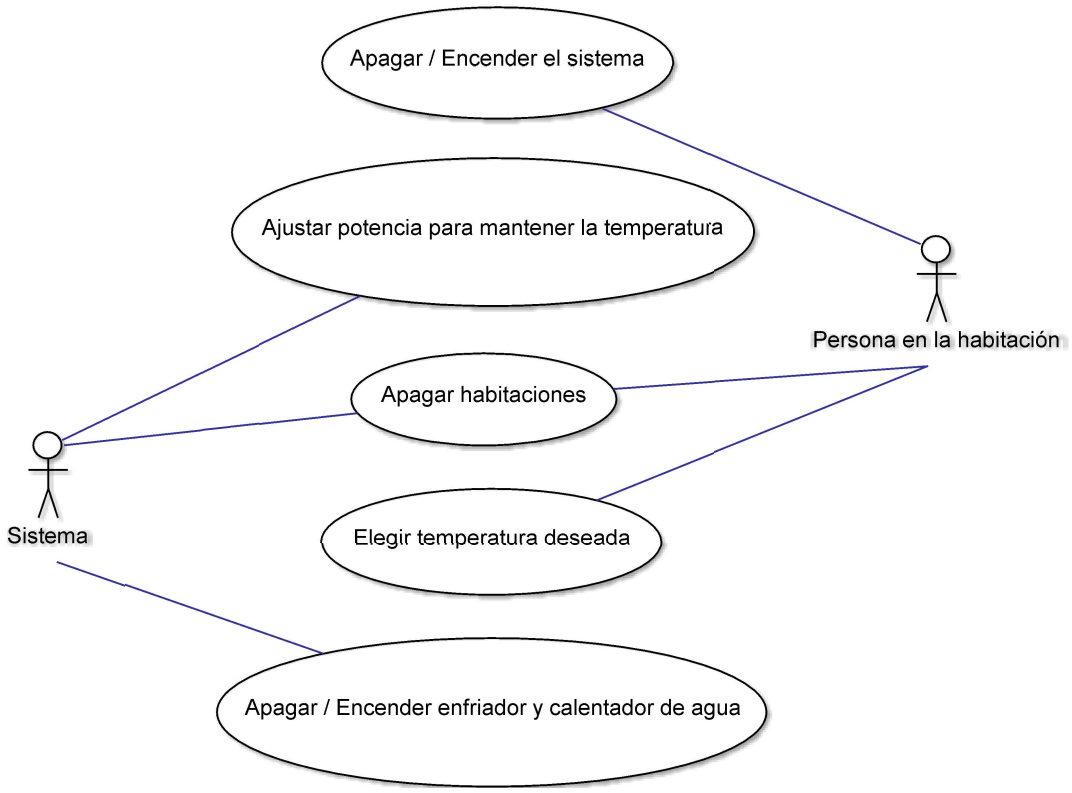

Encendido general
¿Calentar, enfriar,
o ninguno?
Calentar
Enfriar
Ninguno
Encender
calentador de agua
Encender
enfriador de agua
Encender ventiladores
al flujo mínimo y
hacer un control individual


Control individual
Leer temperatura
Ejecutar sistema de control
Cambiar velocidad de ventilador y válvula de flujo según salida de control

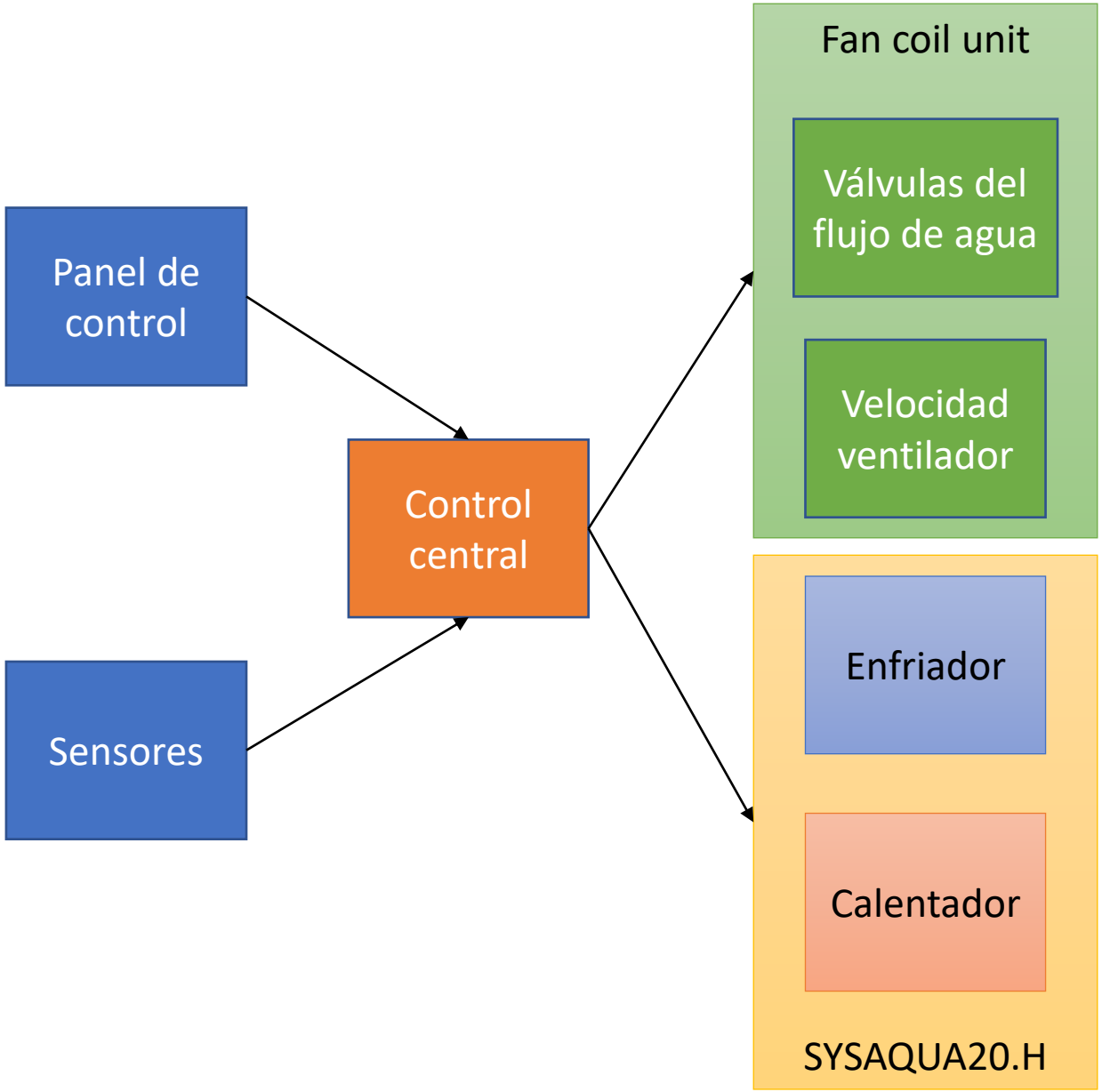

Apagar sistema
Apagar calentador/ enfriador
Delay 1 minuto
Apagar fan coil units

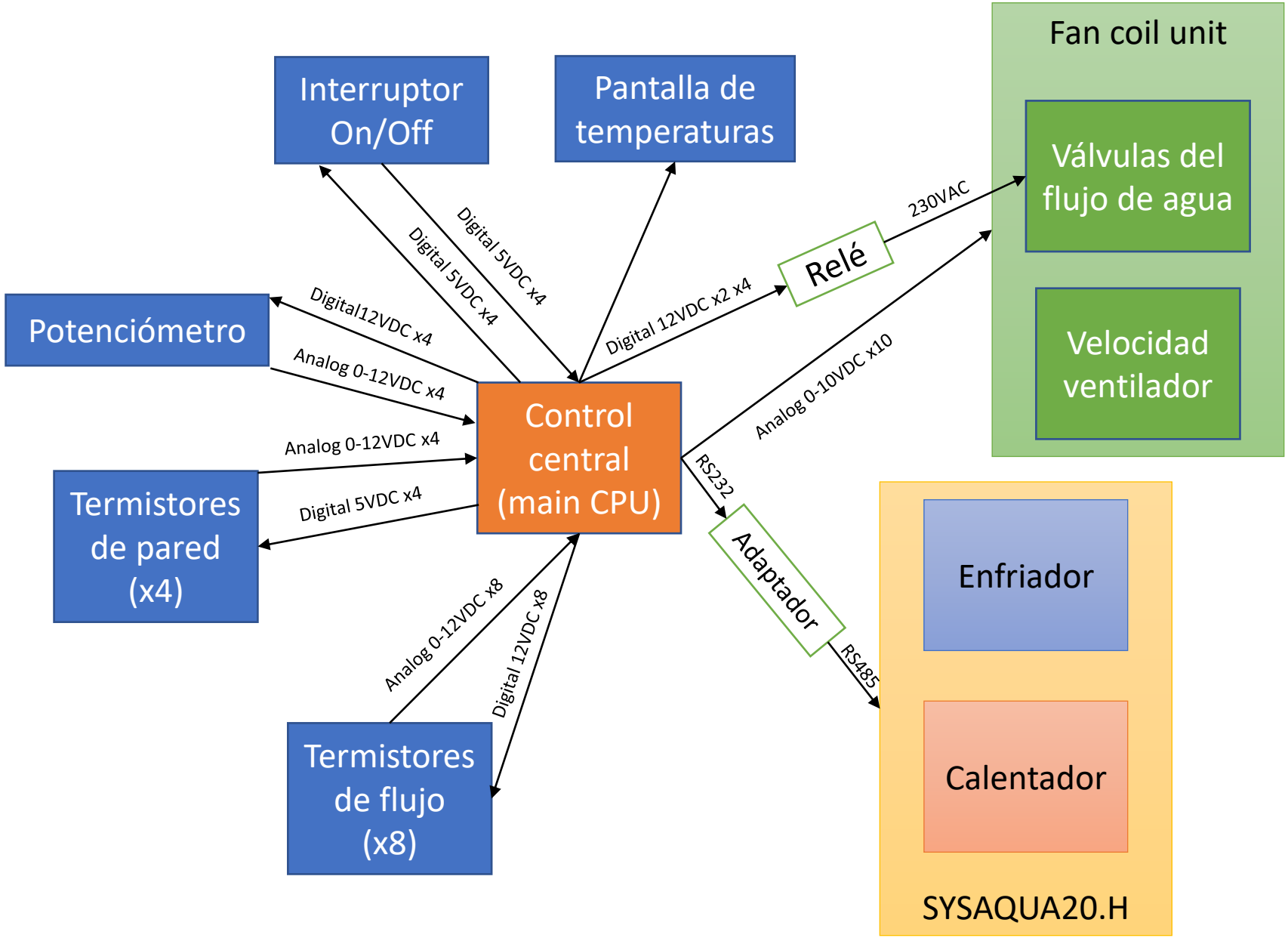

Encender nueva habitación
Encender
Leer temperatura y necesidad de enfriar/ calentar/ventilar
Pasar datos por el controlador
Encender ventilador y abrir serpentín según la necesidad

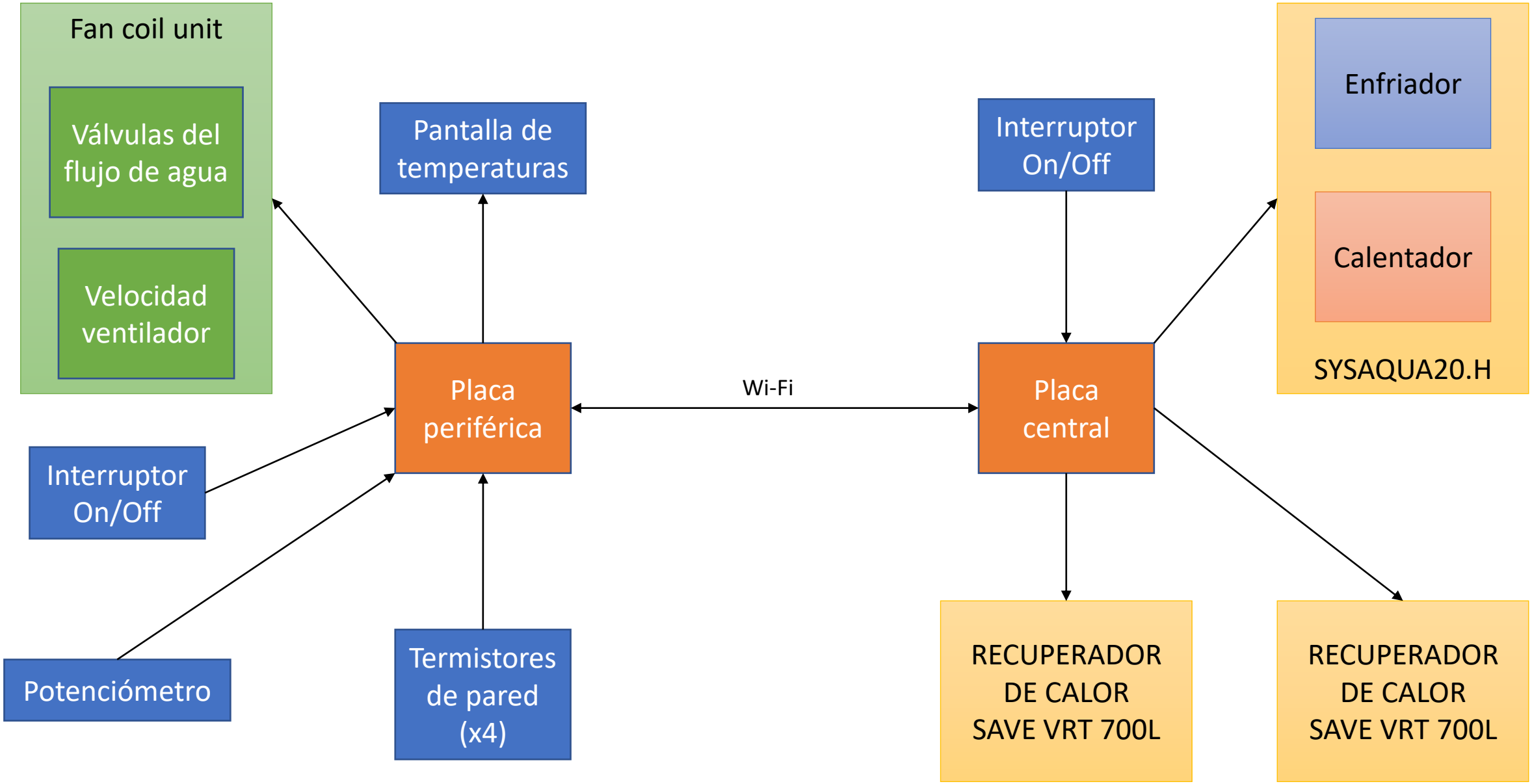
2.1.2. Diagramas de casos de uso


Apagar / Encender el sistema
Ajustar potencia para mantener la temperatura
Persona en la habitación
Apagar habitaciones
Sistema
Elegir temperatura deseada
Apagar / Encender enfriador y calentador de agua

2.2. DIAGRAMAS DE BLOQUES


Panel de control
Sensores
Control central
Fan coil unit
Válvulas del flujo de agua
Velocidad ventilador
Enfriador
Calentador
SYSAQUA20.H


Fan coil unit
Válvulas del flujo de agua
Velocidad ventilador
Interruptor On/Off
Pantalla de temperaturas
Digital 5VDC x4
Digital 5VDC x4
Digital 12VDC x2 x4
Relé
230VAC
Potenciómetro
Digital12VDC x4
Analog 0-12VDC x4
Analog 0-10VDC x10
Control central (main CPU)
Analog 0-12VDC x4
Digital 5VDC x4
Termistores de pared (x4)
RS232
Adaptador
RS485
Enfriador
Calentador
SYSAQUA20.H
Analog 0-12VDC x8
Digital 12VDC x8
Termistores de flujo (x8)


Fan coil unit
Válvulas del flujo de agua
Velocidad ventilador
Pantalla de temperaturas
Interruptor On/Off
Placa periférica
Wi-Fi
Placa central
Enfriador
Calentador
SYSAQUA20.H
Interruptor On/Off
Potenciómetro
Termistores de pared (x4)
RECUPERADOR DE CALOR SAVE VRT 700L
RECUPERADOR DE CALOR SAVE VRT 700L

3. CÓDIGO

3.1. CONTROL CENTRAL

```
#include <Arduino.h>
#include <esp_now.h>
#include <WiFi.h>

uint8_t slaveAddress[] = {0xFF, 0xFF, 0xFF, 0xFF};

typedef struct struct_message {
  byte id;
  byte power;
  byte heating;
};
struct_message message;

struct_message board1;
struct_message board2;
struct_message board3;
struct_message board4;

struct_message board[3] = {board1, board2, board3, board4};

byte total_power;
byte power_heating;
byte power_cooling;
bool heating;

byte save1_pin;
byte save2_pin;

void setup() {
  save1_pin = 3;
  save2_pin = 4;

  pinMode(save1_pin, OUTPUT);
  pinMode(save2_pin, OUTPUT);


  Serial.begin(115200);

  WiFi.mode(WIFI_STA);

  // Init ESP-NOW
  if (esp_now_init() != ESP_OK) {
    Serial.println("There was an error initializing ESP-NOW");
    return;
  }

  esp_now_register_recv_cb(OnRecv);

}
```

```
void loop() {
  power_heating = 0;
  for (byte i = 0; i <= 3 ; i++) {
    if (board[i].heating == 1) {
      power_heating += board[i].power;
    }
    else if (board[i].heating == 0) {
      power_cooling += board[i].power;
    }
  }

  if (power_heating >= power_cooling){
    heating = true;
    Serial.write("Heating", 6);
    total_power = power_heating;
  }
  else{
    heating = false;
    Serial.write("Cooling", 6);
    total_power = power_cooling;
  }

  if (total_power > 0 && total_power < 127) {
    digitalWrite(save1_pin, HIGH);
    digitalWrite(save2_pin, LOW);
  }
  else if (total_power > 127) {
    digitalWrite(save1_pin, HIGH);
    digitalWrite(save2_pin, HIGH);
  }
  else if (total_power = 0) {
    digitalWrite(save1_pin, LOW);
    digitalWrite(save2_pin, LOW);
  }


}


void OnRecv(const uint8_t * mac_addr, const uint8_t *incomingData, int len) {
  char macStr[18];
  Serial.print("Packet received from: ");
  snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
           mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4],
mac_addr[5]);

  memcpy(&message, incomingData, sizeof(message));

  boardsStruct[message.id - 1].power = message.power;
  boardsStruct[message.id - 1].heating = message.heating;
}
```

3.2. CONTROL PERIFÉRICO

```
#include <esp_now.h>
#include <WiFi.h>
#include <Wire.h>
#include <Preferences.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define I2C_SDA 21
#define I2C_SCL 41
#define OLED_RESET 4
Adafruit_SSD1306 display(OLED_RESET);

uint8_t broadcastAddress[] = {0xFF, 0xFF, 0xFF, 0xFF};


typedef struct struct_message {
  byte id;
  byte power;
  byte heating;
} struct_message;
struct_message message;

esp_now_peer_info_t peerInfo;


byte PID_lower_limit = 0, PID_upper_limit = 255;
```

```
unsigned long current_millis;
unsigned long previous_millis_big, previous_millis_small;
int T = 15000, small_T = 1000;

float PID_p = 0.0, PID_i = 0.0, PID_d = 0.0;
float PID_kp = 1.0, PID_ki = 1.0, PID_kd = 1.0;
float process_output[60], slope[60], setpoint[2], error[60], temperature[7];
int control_signal[60];
float noise, u_min, u_max, max_uh, max_ul, min_uh, min_ul, sl, sh, tune_band,
lower_tune_limit, upper_tune_limit, upper_threshold,
      uh, ul, yh, yl, phi, theta, tsl, tsh, tyl, tyh, tys, sigma[2], average_sigma,
lambda_sn, lambda_ss, eh, el;
byte slope_sign_changes, nex_uh, nex_ul, nex_eh, nex_el, feature_samples;
bool output_rising, after_setpoint_change, reached_threshold, slope_positive,
disturbance, reached_setpoint,
      increase_kp, increase_ki, increase_kd, decrease_kp, decrease_ki,
decrease_kd;


float outside_temperature, lowest_setpoint = 15.0;
const float heating_limit_temperature = 19.0, cooling_limit_temperature = 27.0,
hysteresis = 0.1;
bool switching = false;

enum State {
  HEATING,
  COOLING,
  NONE,
  OFF
};
```

```
State state;

const byte heating_pin = 1;
const byte cooling_pin = 3;
const byte fan_pin = 5;
const byte sensor_pin = 6;
const byte button_pin = 7;
const byte potentiometer_pin = 8;



void setup() {
  preferences.begin("my-app", false);

  // Start Wire library for I2C
  Wire.begin();
  // initialize OLED with I2C addr 0x3C
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

  Serial.begin(115200);

  WiFi.mode(WIFI_STA);

  if (esp_now_init() != ESP_OK) {
    Serial.println("Error initializing ESP-NOW");
    return;
  }

  esp_now_register_send_cb(OnDataSent);
```

```
  memcpy(peerInfo.peer_addr, broadcastAddress, 4);
  peerInfo.channel = 0;
  peerInfo.encrypt = false;

  if (esp_now_add_peer(&peerInfo) != ESP_OK) {
    Serial.println("Failed to add peer");
    return;
  }

  myData.id = 1;

  for (byte i = 0; i <= 6; i++) {
    temperature[i] = (analogRead(sensor_pin)/1023.75) * 0.0915751 - 18;
  }
  process_output[0] = temperature[3];

  outside_temperature = process_output[0];

  ledcSetup(0, 5000, 1);
  ledcAttachPin(fan_pin, 0);

  state = OFF;
}

void loop() {

  current_millis = millis();
```

```
    if (digitalRead(button_pin) == HIGH) {
      switching = true;
    }

    if (state == NONE) {
      if (outside_temperature >= cooling_limit_temperature + hysteresis) {
        state = COOLING;
        startCooling();
      }
      else if (outside_temperature <= heating_limit_temperature - hysteresis) {
        state = HEATING;
        startHeating();
      }
    }
    else if (state == HEATING && outside_temperature >=
heating_limit_temperature + hysteresis) {
      state = NONE;
      finishHeating();
    }
    else if (state == COOLING && outside_temperature <=
cooling_limit_temperature - hysteresis) {
      state = NONE;
      finishCooling();
    }
    else if (state == OFF && switching && digitalRead(button_pin) == LOW) {
      switching = false;
      state = NONE;
    }
    else if (state == HEATING && switching && digitalRead(button_pin) == LOW) {
```

```
    switching = false;
    state = OFF;
    motorController.TurnRight(0);
    finishHeating();
  }
  else if (state == COOLING && switching && digitalRead(button_pin) == LOW)
{
    switching = false;
    state = OFF;
    motorController.TurnRight(0);
    finishCooling();
  }

  if (current_millis - previous_millis_small >= small_T) {
    if (state == HEATING) {
    setpoint[0] = round(2 * (analogRead(potentiometer_pin) / 1023.75 +
(heating_limit_temperature - 4.0))) / 2;
    }
    else if (state == COOLING) {
      setpoint[0] = round(2 * (analogRead(potentiometer_pin) / 1023.75 +
cooling_limit_temperature)) / 2;
    }
    showOnDisplay();
    previous_millis_small = current_millis;
  }

  if (current_millis - previous_millis_big >= T && (state == HEATING || state ==
COOLING)) {

    PIDcontrol();
```

```
      plant();
      Block1();
      if (!Block2()) {
        _block2.number = 0.0;
        if (Block3()) {
          _block3.number = 1.0;
          Block4();
          Block5();
        }
        else
          _block3.number = 0.0;
      }
      else {
        _block2.number = 1.0;
      }

      previous_millis_big = current_millis;
    }

  }


  void plant() {
    motorController.Enable();
    if (state == HEATING) {
      digitalWrite(heating_pin, HIGH);
      digitalWrite(cooling_pin, LOW);
      ledcWrite(0, control_signal[0]);
      message.heating = 1;
```

```
  }
  else if (state == COOLING) {
    digitalWrite(heating_pin, LOW);
    digitalWrite(cooling_pin, HIGH);
    ledcWrite(0, control_signal[0]);
    message.heating = 0;
  }

  message.power = (control_signal[0] / 990) * 360;
  esp_err_t result = esp_now_send(broadcastAddress, (uint8_t *) &message,
sizeof(message));
}



void showOnDisplay() {

 display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(2);

  display.setCursor(0, 0);
  display.print("Current: ");
  display.print(inside_temperature);
  display.print(" ºC");



  display.setCursor(0, 15);
  display.print("Desired: ");
```

```
  display.print(desired_temperature);
  display.print(" ºC");

  display.display();

}


void PIDcontrol() {

  for (byte i = 50; i >= 1; i--) {
    error[i] = error[i - 1];
  }


  if (state == HEATING)
    error[0] = setpoint[0] - process_output[0];
  else if (state == COOLING)
    error[0] =  process_output[0] - setpoint[0];


  PID_p = PID_kp * error[0];                              //  P //

  if (PID_p > PID_upper_limit)
    PID_p = PID_upper_limit;
  else if (PID_p < PID_lower_limit)
    PID_p = PID_lower_limit;
```

```
//PID_i = PID_ki * (T / 1000) * (error[0] + error[1]);
PID_i = PID_i + (PID_ki * error[0]);                    //  I //

PID_d = PID_kd * ((error[0] - error[1]) / (T / 1000));          //  D //

float lower_limit_i, upper_limit_i, upper_limit_d, lower_limit_d;

if (PID_upper_limit > PID_p) {
  upper_limit_i = PID_upper_limit - PID_p;
}
else
  upper_limit_i = 0;

if (PID_upper_limit > (PID_p + PID_i)) {
  upper_limit_d = PID_upper_limit - PID_p - PID_i;
}
else
  upper_limit_d = 0;

if (PID_lower_limit < (PID_p + PID_i)) {
  lower_limit_d = PID_lower_limit - PID_p - PID_i;
}
else
  upper_limit_d = 0;


if (PID_lower_limit < PID_p) {
```

```
    lower_limit_i = PID_lower_limit - PID_p;
  }
  else
    lower_limit_i = 0;

  if (PID_i > upper_limit_i) {
    PID_i = upper_limit_i;
  }
  else if (PID_i < lower_limit_i) {
    PID_i = lower_limit_i;
  }


  int output = PID_p + PID_i + PID_d;

  if (output > PID_upper_limit) {
    output = PID_upper_limit;
  }
  else if (output < PID_lower_limit) {
    output = PID_lower_limit;
  }

  for (byte i = 50; i >= 1; i--) {
    control_signal[i] = control_signal[i - 1];
  }
  control_signal[0] = output;
}
```

```
void Block1() {
  temperature[6] = temperature[5];
  temperature[5] = temperature[4];
  temperature[4] = temperature[3];
  temperature[3] = (analogRead(sensor_pin)/1023.75) * 0.0915751 - 18;

  float temperature_difference;
  float temperature_delta;

  temperature_difference = ((temperature[5] - temperature[6]) +
(temperature[4] - temperature[5]) + (temperature[3] - temperature[4])) / 3;
  temperature_delta = ((temperature[5] - temperature[6] -
temperature_difference) +
                    (temperature[4] - temperature[5] - temperature_difference) +
                    (temperature[3] - temperature[4] - temperature_difference) /
3);

  temperature[2] = temperature[3] + temperature_difference +
temperature_delta;
  temperature[1] = temperature[2] + temperature_difference +
temperature_delta;
  temperature[0] = temperature[1] + temperature_difference +
temperature_delta;

  for (byte i = 50; i >= 1; i--) {
    process_output[i] = process_output[i - 1];
  }
```

```
    process_output[0] = (-6 * temperature[5] + 24 * temperature[4] + 34 *
temperature[3] + 24 * temperature[2] - 6 * temperature[1]) / 70;


    for (byte i = 50; i >= 1; i--) {
      slope[i] = slope[i - 1];
    }
    if (state == HEATING)
      slope[0] = process_output[0] - process_output[1];
    else if (state == COOLING)
      slope[0] = process_output[1] - process_output[0];
    //slope = (-3 * temperature[6] - 2 * temperature[5] - temperature[4] +
temperature[2] + 2 * temperature[1] + 3 * temperature[0]) / (28 * (T / 1000));


    noise = noise + 0.001 * (abs(process_output[0] - temperature[3]) - noise);
  }





  bool Block2() {
    bool setpoint_change;


    if (setpoint[1] == setpoint[0])
      setpoint_change = false;
    else {
      setpoint_change = true;
      feature_samples = 0;
      after_setpoint_change = true;
    }
```

```
  setpoint[1] = setpoint[0];

  return setpoint_change;
}

bool Block3() {
  bool finished;
  feature_samples++;

  if ((feature_samples >= 50) && (after_setpoint_change)) {

    uh = control_signal[0];
    ul = control_signal[0];
    yh = process_output[0];
    yl = process_output[0];
    sh = slope[0];
    sl = slope[0];
    eh = error[0];
    el = error[0];
    tyh = 0;
    tyl = 0;
    tys = 0;
    tsh = 0;
    tsl = 0;
    slope_sign_changes = 0;
```

```
if (slope[0] >= 0)
  slope_positive = true;
else
  slope_positive = false;

for (byte i = 0; i <= feature_samples; i++) {
  if (control_signal[i] > uh) {
    uh = control_signal[i];
  }
  else if (control_signal[i] < ul) {
    ul = control_signal[i];
  }
}

for (byte i = 0; i <= feature_samples; i++) {
  if (error[i] > eh) {
    eh = error[i];
  }
  else if (error[i] < el) {
    el = error[i];
  }
}

for (byte i = 0; i <= feature_samples; i++) {
  if (state == HEATING) {
    if (process_output[i] > yh) {
      yh = process_output[i];
      tyh = i * (T / 1000);
```

```
      }
      else if (process_output[i] < yl) {
        yl = process_output[i];
      }

      if (process_output[i] >= setpoint[0] && !reached_setpoint) {
        reached_setpoint = true;
        tys = i * (T / 1000);
      }
    }
    else if (state == COOLING) {
      if (process_output[i] < yh) {
        yh = process_output[i];
        tyh = i * (T / 1000);
      }
      else if (process_output[i] > yl) {
        yl = process_output[i];
      }

      if (process_output[i] <= setpoint[0] && !reached_setpoint) {
        reached_setpoint = true;
        tys = i * (T / 1000);
      }
    }

  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (state == HEATING) {
```

```
    if (slope[i] > sh) {
      sh = slope[i];
      tsh = i * (T / 1000);
    }
    else if (slope[i] < sl) {
      sl = slope[i];
      tsl = i * (T / 1000);
    }
  }
  else if (state == COOLING) {
    if (slope[i] < sh) {
      sh = slope[i];
      tsh = i * (T / 1000);
    }
    else if (slope[i] > sl) {
      sl = slope[i];
      tsl = i * (T / 1000);
    }
  }

  if (slope_positive && (slope[i] < 0)) {
    slope_sign_changes++;
    slope_positive = false;
  }
  else if (!slope_positive && (slope[i] > 0)) {
    slope_sign_changes++;
    slope_positive = true;
  }
```

```
    }

    finished = true;
    after_setpoint_change = false;
  }
  else if ((feature_samples >= 30) && (!after_setpoint_change)) {

    uh = control_signal[0];
    ul = control_signal[0];
    yh = process_output[0];
    yl = process_output[0];
    sh = slope[0];
    sl = slope[0];
    eh = error[0];
    el = error[0];
    tyh = 0;
    tyl = 0;
    tsh = 0;
    tsl = 0;
    slope_sign_changes = 0;

    if (slope[0] >= 0)
      slope_positive = true;
    else
      slope_positive = false;

    for (byte i = 0; i <= feature_samples; i++) {
      if (control_signal[i] > uh) {
        uh = control_signal[i];
```

```
    }
    else if (control_signal[i] < ul) {
      ul = control_signal[i];
    }
  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (error[i] > eh) {
      eh = error[i];
    }
    else if (error[i] < el) {
      el = error[i];
    }
  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (state == HEATING) {
      if (process_output[i] > yh) {
        yh = process_output[i];
        tyh = i * (T / 1000);
      }
      else if (process_output[i] < yl) {
        yl = process_output[i];
      }
    }
    else if (state == COOLING) {
      if (process_output[i] < yh) {
        yh = process_output[i];
        tyh = i * (T / 1000);
```

```
      }
      else if (process_output[i] > yl) {
        yl = process_output[i];
      }
    }
  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (state == HEATING) {
      if (slope[i] > sh) {
        sh = slope[i];
        tsh = i * (T / 1000);
      }
      else if (slope[i] < sl) {
        sl = slope[i];
        tsl = i * (T / 1000);
      }
    }
    else if (state == COOLING) {
      if (slope[i] < sh) {
        sh = slope[i];
        tsh = i * (T / 1000);
      }
      else if (slope[i] > sl) {
        sl = slope[i];
        tsl = i * (T / 1000);
      }
    }
```

```
        if (slope_positive && (slope[i] < 0)) {
          slope_sign_changes++;
          slope_positive = false;
        }
        else if (!slope_positive && (slope[i] > 0)) {
          slope_sign_changes++;
          slope_positive = true;
        }

      }

      finished = true;
      after_setpoint_change = false;
    }
    else
      finished = false;

    sigma[1] = sigma[0];
    sigma[0] = maxFloat(yh, setpoint[0]) - minFloat(yl, setpoint[0]);
    average_sigma = (sigma[0] + sigma[1]) / 2;
    tune_band = sigma[0] / noise;

    if (tsl > tsh)
      phi = minFloat(1.0, abs(sl / sh));
    else
      phi = minFloat(1.0, abs(sh / sl));

    lambda_sn = minFloat(maxFloat(0.0, (3.1 * sigma[0]) / tune_band - 2.97),
1.0);
```

```
  lambda_ss = minFloat(maxFloat(0.0, sigma[0] / (2.19 * average_sigma)),
1.0);
  theta = (yh - yl) / ((T / 1000) * maxFloat(abs(sl), abs(sh)));



  return finished;
}



void Block4() {
  increase_kp = false;
  increase_ki = false;
  increase_kd = false;
  decrease_kp = false;
  decrease_ki = false;
  decrease_kd = false;

  if (feature_samples > 45) {
    if ((el < -1.5) && (slope_sign_changes <= 3)) {
      increase_kp = true;
      decrease_ki = true;
      increase_kd = true;
    }
    else if ((el < -1.5) && (slope_sign_changes >= 4) && (slope_sign_changes
<= 5)) {
      decrease_kp = true;
      increase_kd = true;
```

```
        }
        else if ((el < -1.5) && (slope_sign_changes >= 6)) {
          decrease_kd = true;
        }
        else if (!reached_setpoint) {
          increase_kp = true;
          increase_ki = true;
          decrease_kd = true;
        }
      }
      else if (feature_samples < 45) {
        if ((eh > 0.3) && (slope_sign_changes <= 3)) {
          increase_kp = true;
        }
        else if ((eh > 0.3) && (slope_sign_changes >= 4) && (slope_sign_changes
<= 5)) {
          increase_kd = true;
          decrease_kp = true;
        }
        else if ((eh > 0.3) && (slope_sign_changes >= 6)) {
          increase_ki = true;
          increase_kp = true;
          decrease_kd = true;
        }
        else if ((el < -0.3) && (slope_sign_changes <= 3)) {
          decrease_ki = true;
          increase_kd = true;
        }
```

```
      else if ((el < -0.3) && (slope_sign_changes >= 4) && (slope_sign_changes
<= 5)) {
        decrease_kp = true;
        increase_kd = true;
      }
      else if ((el < -0.3) && (slope_sign_changes >= 6)) {
        decrease_kp = true;
        decrease_kd = true;
      }
    }


    if (increase_kp && decrease_kp) {
      increase_kp = false;
      decrease_kp = false;
    }

    if (increase_ki && decrease_ki) {
      increase_ki = false;
      decrease_ki = false;
    }

    if (increase_kd && decrease_kd) {
      increase_kd = false;
      decrease_kd = false;
    }

  }
```

```
void Block5() {
  _block5.number = 0;

  if (increase_kp && PID_kp <= 100) {
    //float new_PID_kp = PID_kp * (1 + lambda_sn * lambda_ss * (0.29 - 2.49 *
phi + 4.32 * powf(phi, 2) - 4.45 * powf(phi, 3) + 1.82 * powf(phi, 4)));
    //PID_kp = new_PID_kp;
    PID_kp = PID_kp * 1.6;
  }
  else if (decrease_kp) {
    //PID_kp = PID_kp * (1 - ((min_uh - max_ul) / (2 * (u_max - u_min))));
    PID_kp = PID_kp * 0.7;
  }

  if (increase_ki  && PID_ki <= 10) {
    //float Ti_opt = T * minFloat(30, maxFloat(2, -3.4 + 1.3 * theta));
    //float new_PID_ki = PID_ki + lambda_sn * lambda_ss * (Ti_opt - PID_ki);
    //PID_ki = new_PID_ki;
    PID_ki = PID_ki * 1.5;
  }
  else if (decrease_ki) {
    //PID_ki = PID_ki * (1 - ((min_uh - max_ul) / (2 * (u_max - u_min))));
    PID_ki = PID_ki * 0.8;
  }

  if (increase_kd && PID_kd <= 1000) {
```

```
    PID_kd = PID_kd * 1.9;
  }
  else if (decrease_kd) {
    PID_kd = PID_kd * 0.6;
  }




  feature_samples = 0;
  reached_setpoint = false;



  if (state == HEATING)
    finishHeating();
  else if (state == COOLING)
    finishCooling();
}



void finishHeating() {
  preferences.putFloat("PID_kp_heat", PID_kp);
  preferences.putFloat("PID_ki_heat", PID_ki);
  preferences.putFloat("PID_kd_heat", PID_kd);
}

void startHeating() {
  PID_kp = preferences.getFloat("PID_kp_heat", 1.0);
  PID_ki = preferences.getFloat("PID_ki_heat", 1.0);
  PID_kd = preferences.getFloat("PID_kd_heat", 1.0);
```

```
  T = 15000;
  small_T = 1000;
  PID_upper_limit = 255;
}

void finishCooling() {
  preferences.putFloat("PID_kp_cool", PID_kp);
  preferences.putFloat("PID_ki_cool", PID_ki);
  preferences.putFloat("PID_kd_cool", PID_kd);
}

void startCooling() {
  PID_kp = preferences.getFloat("PID_kp_cool", 1.0);
  PID_ki = preferences.getFloat("PID_ki_cool", 1.0);
  PID_kd = preferences.getFloat("PID_kd_cool", 1.0);
  T = 45000;
  small_T = 3000;
  PID_upper_limit = 255;
}
```

3.3. CÓDIGO DE LA SIMULACIÓN

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include "Adafruit_SHT31.h"
#include "BTS7960.h"
#include <Preferences.h>

#define I2C_SDA 21
#define I2C_SCL 22
Adafruit_SHT31 sht31 = Adafruit_SHT31();
Preferences preferences;

const byte potentiometer_pin = 4, power_button = 2, peltier_EN = 14,
peltier_L_PWM = 32, peltier_R_PWM = 33;
byte PID_lower_limit = 0, PID_upper_limit = 255;

unsigned long current_millis;
unsigned long previous_millis_big, previous_millis_small;
int T = 15000, small_T = 1000;

float PID_p = 0.0, PID_i = 0.0, PID_d = 0.0;
float PID_kp = 1.0, PID_ki = 1.0, PID_kd = 1.0;
float process_output[60], slope[60], setpoint[2], error[60], temperature[7];
int control_signal[60];
float noise, u_min, u_max, max_uh, max_ul, min_uh, min_ul, sl, sh, tune_band,
lower_tune_limit, upper_tune_limit, upper_threshold,
```

```
        uh, ul, yh, yl, phi, theta, tsl, tsh, tyl, tyh, tys, sigma[2], average_sigma,
lambda_sn, lambda_ss, eh, el;
    byte slope_sign_changes, nex_uh, nex_ul, nex_eh, nex_el, feature_samples;
    bool output_rising, after_setpoint_change, reached_threshold, slope_positive,
disturbance, reached_setpoint,
        increase_kp, increase_ki, increase_kd, decrease_kp, decrease_ki,
decrease_kd;



    float outside_temperature = 30.0, lowest_setpoint = 20.0;
    const float heating_limit_temperature = 21.0, cooling_limit_temperature = 25.0,
hysteresis = 0.5;
    bool switching = false;



    BTS7960 motorController(peltier_EN, peltier_L_PWM, peltier_R_PWM);


    enum State {
      HEATING,
      COOLING,
      NONE,
      OFF
    };
    State state;



    typedef union {
      float number;
      byte bytes[4];
    } value;
```

```
value _inside_temperature;
value _desired_temperature;
value _PID_p;
value _PID_i;
value _PID_d;
value _PID_kp;
value _PID_ki;
value _PID_kd;
value _phi;
value _sl;
value _sh;
value _lambda_sn;
value _lambda_ss;
value _sigma_0;
value _yl;
value _yh;
value _tune_band;
value _temperature_3;
value _slope;
value _noise;
value _u_max;
value _u_min;
value _nex_e;
value _nex_u;
value _error_0;
value _error_6;
value _error_12;
value _control_signal_0;
```

```
value _control_signal_6;
value _control_signal_12;
value _feature_samples;
value _block2;
value _block3;
value _block4;
value _block5;
value _eh;
value _el;
value _slope_sign_changes;


void setup() {
  preferences.begin("my-app", false);
  Wire.begin();   //Start Wire library for I2C

  Serial.begin(9600);

  sht31.begin(0x44);

  for (byte i = 0; i <= 6; i++) {
    temperature[i] = sht31.readTemperature();
  }
  process_output[0] = temperature[3];

  state = OFF;
```

```
}



void loop() {


  current_millis = millis();


  if (digitalRead(power_button) == HIGH) {
    switching = true;
  }


  if (state == NONE) {
    if (outside_temperature >= cooling_limit_temperature + hysteresis) {
      state = COOLING;
      startCooling();
    }
    else if (outside_temperature <= heating_limit_temperature - hysteresis) {
      state = HEATING;
      startHeating();
    }
  }
  else if (state == HEATING && outside_temperature >=
heating_limit_temperature + hysteresis) {
    state = NONE;
    finishHeating();
  }
  else if (state == COOLING && outside_temperature <=
cooling_limit_temperature - hysteresis) {
    state = NONE;
```

```
    finishCooling();
  }
  else if (state == OFF && switching && digitalRead(power_button) == LOW) {
    switching = false;
    state = NONE;
  }
  else if (state == HEATING && switching && digitalRead(power_button) ==
LOW) {
    switching = false;
    state = OFF;
    motorController.TurnRight(0);
    finishHeating();
  }
  else if (state == COOLING && switching && digitalRead(power_button) ==
LOW) {
    switching = false;
    state = OFF;
    motorController.TurnRight(0);
    finishCooling();
  }

  if (current_millis - previous_millis_small >= small_T) {
    setpoint[0] = round(2 * (analogRead(potentiometer_pin) / 1023.75 +
lowest_setpoint)) / 2;
    showOnDisplay();
    previous_millis_small = current_millis;
  }

  if (current_millis - previous_millis_big >= T && (state == HEATING || state ==
COOLING)) {
```

```
    PIDcontrol();
    plant();
    Block1();
    if (!Block2()) {
      _block2.number = 0.0;
      if (Block3()) {
        _block3.number = 1.0;
        Block4();
        Block5();
      }
      else
        _block3.number = 0.0;
    }
    else {
      _block2.number = 1.0;
    }

    previous_millis_big = current_millis;
  }
}


void plant() {
  motorController.Enable();
  if (state == HEATING)
    motorController.TurnRight(control_signal[0]);
```

```
    else if (state == COOLING)
      motorController.TurnLeft(control_signal[0]);
  }




  void showOnDisplay() {


    _inside_temperature.number = process_output[0];
    _desired_temperature.number = setpoint[0];
    //_u_min.number = u_min;
    //_u_max.number = u_max;
    _PID_p.number = PID_p;
    _PID_i.number = PID_i;
    _PID_d.number = PID_d;
    //_temperature_3.number = temperature[3];
    _slope.number = slope[0];
    _noise.number = noise;
    _PID_kp.number = PID_kp;
    _PID_ki.number = PID_ki;
    _PID_kd.number = PID_kd;
    _phi.number = phi;
    //_sl.number = sl;
    //_sh.number = sh;
    //_lambda_sn.number = lambda_sn;
    //_lambda_ss.number = lambda_ss;
    //_yl.number = yl;
    //_yh.number = yh;
    //_sigma_0.number = sigma[0];
```

```
_tune_band.number = tune_band;
_nex_u.number = nex_uh + nex_ul;
_nex_e.number = nex_eh + nex_el;
_error_0.number = error[0];
//_error_6.number = error[6];
//_error_12.number = error[12];
_control_signal_0.number = control_signal[0];
//_control_signal_6.number = control_signal[6];
//_control_signal_12.number = control_signal[12];
_feature_samples.number = feature_samples;
_eh.number = eh;
_el.number = el;
_slope_sign_changes.number = slope_sign_changes;

Serial.write('T');
Serial.write(_inside_temperature.bytes, 4);
Serial.write(_desired_temperature.bytes, 4);
Serial.write(_control_signal_0.bytes, 4);
//Serial.write(_PID_p.bytes, 4);
//Serial.write(_PID_i.bytes, 4);
//Serial.write(_PID_d.bytes, 4);
Serial.write(_PID_kp.bytes, 4);
Serial.write(_PID_ki.bytes, 4);
Serial.write(_PID_kd.bytes, 4);
//Serial.write(_u_min.bytes, 4);
//Serial.write(_u_max.bytes, 4);
//Serial.write(_temperature_3.bytes, 4);
//Serial.write(_slope.bytes, 4);
//Serial.write(_noise.bytes, 4);
```

```
    //Serial.write(_sl.bytes, 4);
    //Serial.write(_sh.bytes, 4);
    //Serial.write(_phi.bytes, 4);
    //Serial.write(_lambda_sn.bytes, 4);
    //Serial.write(_lambda_ss.bytes, 4);
    //Serial.write(_yl.bytes, 4);
    //Serial.write(_yh.bytes, 4);
    //Serial.write(_sigma_0.bytes, 4);
    //Serial.write(_noise.bytes, 4);
    //Serial.write(_tune_band.bytes, 4);
    //Serial.write(_nex_e.bytes, 4);
    //Serial.write(_nex_u.bytes, 4);
    //Serial.write(_error_0.bytes, 4);
    //Serial.write(_error_6.bytes, 4);
    //Serial.write(_error_12.bytes, 4);
    //Serial.write(_control_signal_6.bytes, 4);
    //Serial.write(_control_signal_12.bytes, 4);
    Serial.write(_feature_samples.bytes, 4);
    //Serial.write(_block2.bytes, 4);
    //Serial.write(_block4.bytes, 4);
    //Serial.write(_block5.bytes, 4);
    //Serial.write(_eh.bytes, 4);
    //Serial.write(_el.bytes, 4);
    //Serial.write(_slope_sign_changes.bytes, 4);
    Serial.write('\n');

  }
```

```
void PIDcontrol() {

  for (byte i = 50; i >= 1; i--) {
    error[i] = error[i - 1];
  }


  if (state == HEATING)
    error[0] = setpoint[0] - process_output[0];
  else if (state == COOLING)
    error[0] =  process_output[0] - setpoint[0];


  PID_p = PID_kp * error[0];                                  //  P //

  if (PID_p > PID_upper_limit)
    PID_p = PID_upper_limit;
  else if (PID_p < PID_lower_limit)
    PID_p = PID_lower_limit;


  //PID_i = PID_ki * (T / 1000) * (error[0] + error[1]);
  PID_i = PID_i + (PID_ki * error[0]);                          //  I //

  PID_d = PID_kd * ((error[0] - error[1]) / (T / 1000));            //  D //

  float lower_limit_i, upper_limit_i, upper_limit_d, lower_limit_d;
```

```
if (PID_upper_limit > PID_p) {
  upper_limit_i = PID_upper_limit - PID_p;
}
else
  upper_limit_i = 0;

if (PID_upper_limit > (PID_p + PID_i)) {
  upper_limit_d = PID_upper_limit - PID_p - PID_i;
}
else
  upper_limit_d = 0;

if (PID_lower_limit < (PID_p + PID_i)) {
  lower_limit_d = PID_lower_limit - PID_p - PID_i;
}
else
  upper_limit_d = 0;


if (PID_lower_limit < PID_p) {
  lower_limit_i = PID_lower_limit - PID_p;
}
else
  lower_limit_i = 0;

if (PID_i > upper_limit_i) {
  PID_i = upper_limit_i;
}
```

```
  else if (PID_i < lower_limit_i) {
    PID_i = lower_limit_i;
  }



  int output = PID_p + PID_i + PID_d;


  if (output > PID_upper_limit) {
    output = PID_upper_limit;
  }
  else if (output < PID_lower_limit) {
    output = PID_lower_limit;
  }


  for (byte i = 50; i >= 1; i--) {
    control_signal[i] = control_signal[i - 1];
  }
  control_signal[0] = output;
}





void Block1() {
  temperature[6] = temperature[5];
  temperature[5] = temperature[4];
  temperature[4] = temperature[3];
  temperature[3] = sht31.readTemperature();
```

```
        float temperature_difference;
        float temperature_delta;

        temperature_difference = ((temperature[5] - temperature[6]) +
(temperature[4] - temperature[5]) + (temperature[3] - temperature[4])) / 3;
        temperature_delta = ((temperature[5] - temperature[6] -
temperature_difference) +
                        (temperature[4] - temperature[5] - temperature_difference) +
                        (temperature[3] - temperature[4] - temperature_difference) /
3);

        temperature[2] = temperature[3] + temperature_difference +
temperature_delta;
        temperature[1] = temperature[2] + temperature_difference +
temperature_delta;
        temperature[0] = temperature[1] + temperature_difference +
temperature_delta;

        for (byte i = 50; i >= 1; i--) {
          process_output[i] = process_output[i - 1];
        }
        process_output[0] = (-6 * temperature[5] + 24 * temperature[4] + 34 *
temperature[3] + 24 * temperature[2] - 6 * temperature[1]) / 70;

        for (byte i = 50; i >= 1; i--) {
          slope[i] = slope[i - 1];
        }
        if (state == HEATING)
          slope[0] = process_output[0] - process_output[1];
        else if (state == COOLING)
```

```
    slope[0] = process_output[1] - process_output[0];
  //slope = (-3 * temperature[6] - 2 * temperature[5] - temperature[4] +
temperature[2] + 2 * temperature[1] + 3 * temperature[0]) / (28 * (T / 1000));

  noise = noise + 0.001 * (abs(process_output[0] - temperature[3]) - noise);
}

bool Block2() {
  bool setpoint_change;

  if (setpoint[1] == setpoint[0])
    setpoint_change = false;
  else {
    setpoint_change = true;
    feature_samples = 0;
    after_setpoint_change = true;
  }

  setpoint[1] = setpoint[0];

  return setpoint_change;
}
```

```
bool Block3() {
  bool finished;
  feature_samples++;

  if ((feature_samples >= 50) && (after_setpoint_change)) {

    uh = control_signal[0];
    ul = control_signal[0];
    yh = process_output[0];
    yl = process_output[0];
    sh = slope[0];
    sl = slope[0];
    eh = error[0];
    el = error[0];
    tyh = 0;
    tyl = 0;
    tys = 0;
    tsh = 0;
    tsl = 0;
    slope_sign_changes = 0;

    if (slope[0] >= 0)
      slope_positive = true;
    else
      slope_positive = false;

    for (byte i = 0; i <= feature_samples; i++) {
      if (control_signal[i] > uh) {
        uh = control_signal[i];
```

```
    }
    else if (control_signal[i] < ul) {
      ul = control_signal[i];
    }
  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (error[i] > eh) {
      eh = error[i];
    }
    else if (error[i] < el) {
      el = error[i];
    }
  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (state == HEATING) {
      if (process_output[i] > yh) {
        yh = process_output[i];
        tyh = i * (T / 1000);
      }
      else if (process_output[i] < yl) {
        yl = process_output[i];
      }

      if (process_output[i] >= setpoint[0] && !reached_setpoint) {
        reached_setpoint = true;
        tys = i * (T / 1000);
      }
```

```
      }
      else if (state == COOLING) {
        if (process_output[i] < yh) {
          yh = process_output[i];
          tyh = i * (T / 1000);
        }
        else if (process_output[i] > yl) {
          yl = process_output[i];
        }

        if (process_output[i] <= setpoint[0] && !reached_setpoint) {
          reached_setpoint = true;
          tys = i * (T / 1000);
        }
      }

    }

    for (byte i = 0; i <= feature_samples; i++) {
      if (state == HEATING) {
        if (slope[i] > sh) {
          sh = slope[i];
          tsh = i * (T / 1000);
        }
        else if (slope[i] < sl) {
          sl = slope[i];
          tsl = i * (T / 1000);
        }
      }
```

```
      else if (state == COOLING) {
        if (slope[i] < sh) {
          sh = slope[i];
          tsh = i * (T / 1000);
        }
        else if (slope[i] > sl) {
          sl = slope[i];
          tsl = i * (T / 1000);
        }
      }

      if (slope_positive && (slope[i] < 0)) {
        slope_sign_changes++;
        slope_positive = false;
      }
      else if (!slope_positive && (slope[i] > 0)) {
        slope_sign_changes++;
        slope_positive = true;
      }

    }

    finished = true;
    after_setpoint_change = false;
  }
  else if ((feature_samples >= 30) && (!after_setpoint_change)) {

    uh = control_signal[0];
    ul = control_signal[0];
```

```
yh = process_output[0];
yl = process_output[0];
sh = slope[0];
sl = slope[0];
eh = error[0];
el = error[0];
tyh = 0;
tyl = 0;
tsh = 0;
tsl = 0;
slope_sign_changes = 0;

if (slope[0] >= 0)
  slope_positive = true;
else
  slope_positive = false;

for (byte i = 0; i <= feature_samples; i++) {
  if (control_signal[i] > uh) {
    uh = control_signal[i];
  }
  else if (control_signal[i] < ul) {
    ul = control_signal[i];
  }
}

for (byte i = 0; i <= feature_samples; i++) {
  if (error[i] > eh) {
    eh = error[i];
```

```
    }
    else if (error[i] < el) {
      el = error[i];
    }
  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (state == HEATING) {
      if (process_output[i] > yh) {
        yh = process_output[i];
        tyh = i * (T / 1000);
      }
      else if (process_output[i] < yl) {
        yl = process_output[i];
      }
    }
    else if (state == COOLING) {
      if (process_output[i] < yh) {
        yh = process_output[i];
        tyh = i * (T / 1000);
      }
      else if (process_output[i] > yl) {
        yl = process_output[i];
      }
    }
  }

  for (byte i = 0; i <= feature_samples; i++) {
    if (state == HEATING) {
```

```
    if (slope[i] > sh) {
      sh = slope[i];
      tsh = i * (T / 1000);
    }
    else if (slope[i] < sl) {
      sl = slope[i];
      tsl = i * (T / 1000);
    }
  }
  else if (state == COOLING) {
    if (slope[i] < sh) {
      sh = slope[i];
      tsh = i * (T / 1000);
    }
    else if (slope[i] > sl) {
      sl = slope[i];
      tsl = i * (T / 1000);
    }
  }

  if (slope_positive && (slope[i] < 0)) {
    slope_sign_changes++;
    slope_positive = false;
  }
  else if (!slope_positive && (slope[i] > 0)) {
    slope_sign_changes++;
    slope_positive = true;
  }
```

```
    }

    finished = true;
    after_setpoint_change = false;
  }
  else
    finished = false;

  sigma[1] = sigma[0];
  sigma[0] = maxFloat(yh, setpoint[0]) - minFloat(yl, setpoint[0]);
  average_sigma = (sigma[0] + sigma[1]) / 2;
  tune_band = sigma[0] / noise;

  if (tsl > tsh)
    phi = minFloat(1.0, abs(sl / sh));
  else
    phi = minFloat(1.0, abs(sh / sl));

  lambda_sn = minFloat(maxFloat(0.0, (3.1 * sigma[0]) / tune_band - 2.97),
1.0);
  lambda_ss = minFloat(maxFloat(0.0, sigma[0] / (2.19 * average_sigma)),
1.0);
  theta = (yh - yl) / ((T / 1000) * maxFloat(abs(sl), abs(sh)));

  return finished;
}
```

```
void Block4() {
  increase_kp = false;
  increase_ki = false;
  increase_kd = false;
  decrease_kp = false;
  decrease_ki = false;
  decrease_kd = false;

  if (feature_samples > 45) {
    if ((el < -1.5) && (slope_sign_changes <= 3)) {
      increase_kp = true;
      decrease_ki = true;
      increase_kd = true;
    }
    else if ((el < -1.5) && (slope_sign_changes >= 4) && (slope_sign_changes
<= 5)) {
      decrease_kp = true;
      increase_kd = true;
    }
    else if ((el < -1.5) && (slope_sign_changes >= 6)) {
      decrease_kd = true;
    }
    else if (!reached_setpoint) {
      increase_kp = true;
      increase_ki = true;
      decrease_kd = true;
    }
```

```
    }
    else if (feature_samples < 45) {
      if ((eh > 0.3) && (slope_sign_changes <= 3)) {
        increase_kp = true;
      }
      else if ((eh > 0.3) && (slope_sign_changes >= 4) && (slope_sign_changes
<= 5)) {
        increase_kd = true;
        decrease_kp = true;
      }
      else if ((eh > 0.3) && (slope_sign_changes >= 6)) {
        increase_ki = true;
        increase_kp = true;
        decrease_kd = true;
      }
      else if ((el < -0.3) && (slope_sign_changes <= 3)) {
        decrease_ki = true;
        increase_kd = true;
      }
      else if ((el < -0.3) && (slope_sign_changes >= 4) && (slope_sign_changes
<= 5)) {
        decrease_kp = true;
        increase_kd = true;
      }
      else if ((el < -0.3) && (slope_sign_changes >= 6)) {
        decrease_kp = true;
        decrease_kd = true;
      }
    }
```

```
    if (increase_kp && decrease_kp) {
      increase_kp = false;
      decrease_kp = false;
    }

    if (increase_ki && decrease_ki) {
      increase_ki = false;
      decrease_ki = false;
    }

    if (increase_kd && decrease_kd) {
      increase_kd = false;
      decrease_kd = false;
    }

  }



  void Block5() {
    _block5.number = 0;

    if (increase_kp && PID_kp <= 100) {
      //float new_PID_kp = PID_kp * (1 + lambda_sn * lambda_ss * (0.29 - 2.49 *
phi + 4.32 * powf(phi, 2) - 4.45 * powf(phi, 3) + 1.82 * powf(phi, 4)));
      //PID_kp = new_PID_kp;
```

```
  PID_kp = PID_kp * 1.6;
 }
 else if (decrease_kp) {
  //PID_kp = PID_kp * (1 - ((min_uh - max_ul) / (2 * (u_max - u_min))));
  PID_kp = PID_kp * 0.7;
 }

 if (increase_ki  && PID_ki <= 10) {
  //float Ti_opt = T * minFloat(30, maxFloat(2, -3.4 + 1.3 * theta));
  //float new_PID_ki = PID_ki + lambda_sn * lambda_ss * (Ti_opt - PID_ki);
  //PID_ki = new_PID_ki;
  PID_ki = PID_ki * 1.5;
 }
 else if (decrease_ki) {
  //PID_ki = PID_ki * (1 - ((min_uh - max_ul) / (2 * (u_max - u_min))));
  PID_ki = PID_ki * 0.8;
 }

 if (increase_kd && PID_kd <= 1000) {
  PID_kd = PID_kd * 1.9;
 }
 else if (decrease_kd) {
  PID_kd = PID_kd * 0.6;
 }



 feature_samples = 0;
 reached_setpoint = false;
```

```
  if (state == HEATING)
    finishHeating();
  else if (state == COOLING)
    finishCooling();

}



void finishHeating() {
  preferences.putFloat("PID_kp_heat", PID_kp);
  preferences.putFloat("PID_ki_heat", PID_ki);
  preferences.putFloat("PID_kd_heat", PID_kd);
}

void startHeating() {
  PID_kp = preferences.getFloat("PID_kp_heat", 1.0);
  PID_ki = preferences.getFloat("PID_ki_heat", 1.0);
  PID_kd = preferences.getFloat("PID_kd_heat", 1.0);
  T = 15000;
  small_T = 1000;
  PID_upper_limit = 255;
}

void finishCooling() {
  preferences.putFloat("PID_kp_cool", PID_kp);
```

```
  preferences.putFloat("PID_ki_cool", PID_ki);
  preferences.putFloat("PID_kd_cool", PID_kd);
}

void startCooling() {
  PID_kp = preferences.getFloat("PID_kp_cool", 1.0);
  PID_ki = preferences.getFloat("PID_ki_cool", 1.0);
  PID_kd = preferences.getFloat("PID_kd_cool", 1.0);
  T = 45000;
  small_T = 3000;
  PID_upper_limit = 255;
}
```

Relación de documentos

(_) Memoria	74	páginas
(X) Anexos	63	páginas

La Almunia, a 20 de Septiembre de 2022

Firmado por ENRIQUEZ ROMERO IGNACIO - 49042033F el dï¿½a 20/09/2022 con un certificado emitido por AC

Firmado: Ignacio Enríquez Romero