```
    /* NODE SENSOR                                                        *
     * Xarxa de sensors d'audio per a la deteccio d'accident urbans *
     * Dani Marti Verge                                                   *
     * v.1.05                                                             *
     **********************************************************************/


  ////////////////////////////////////////////////////////////////////
 //              LLIBRERIES DEL NODE SENSOR 10.1                   //
////////////////////////////////////////////////////////////////////

#include <RF24Network.h>
#include <RF24.h>
#include <SPI.h>

  ////////////////////////////////////////////////////////////////////
 //                            PARAMETRES I VARIABLES                //
////////////////////////////////////////////////////////////////////

//Pins SPI i declaració del NRF24
RF24 radio(7, 8);
RF24Network network(radio);

// Adreça del node actual en octal
const uint16_t aquest_node = 01;

// Adreça node a enviar en octal
const uint16_t altre_node = 00;

// Variables del payload a enviar
struct dades_t
{
  float Acc;
  float BatS;
  float SolS;
  float Id;
};

// Paràmetres i variables del ADC del micròfon
const uint32_t FS = 40000;
const uint32_t FS_INTERVAL = F_CPU / FS;
const uint16_t MIN_ADC_CYCLES = 15;
uint8_t const ADC_REF_AVCC = (1 << REFS0);
int BufPtr = 0;
boolean ErrorCond = false;

// Declaració funcions adquisició
void adcInit(uint32_t ticks, uint8_t pin, uint8_t ref);
void adcStart();
uint16_t acquire();

// Configuració de sortida lineal de la transformada FHT
#define LIN_OUT 1

// Numero de punts de la transformada FHT
#define FHT_N 256

// lliberia de la transformada FHT
#include <FHT.h>
```

```
// Declaració entrades analògiques microfon i placa solar
const uint8_t microfon = 0;
int solar = A1;    // Solar

// Variables anivellació amplitud
long mx1;
long mx2;
long equiv1;

// Variables càlcul similitud
long dist1;
long dist2;
long dist1total;
long dist2total;
long rel1;
long rel2;
long rel1acum;
long rel2acum;

// Variables compliment condicions similitud
int n1 = 0;
int n2 = 0;
int t2 = 0;
byte t3 = 0;
bool frenada = false;
bool inicifrenada = false;

// Constants compliment condicions similitud
const int limit1 = 22;
const int limit2 = 44;
const int nlimit1 = 5;
const int nlimit2 = 4;
const int tlimit1 = 50;
const int tlimit2 = 170;
const int tlimit3 = 40;

// Declaració payload
dades_t dades;

// Comptadors
unsigned long t1 = 0;
int i;

  ////////////////////////////////////////////////////////////////
 //              VECTORS DE FREQUENCIES DELS PATRONS             //
////////////////////////////////////////////////////////////////

// Patro 1, fase de frenada
long patro1[] =
{/*0Hz*/155,/*156Hz*/77,/*312Hz*/326,/*468Hz*/214,/*625Hz*/4271,/*781H
z*/2146,/*937Hz*/1152,/*1093Hz*/150,/*1250Hz*/400,/*1406Hz*/1071,/*156
2Hz*/331,/*1718Hz*/30,/*1875Hz*/0,/*2031Hz*/20,/*2187Hz*/30,/*2343Hz*/
77,/*2500Hz*/0,/*2656Hz*/0,/*2812Hz*/40,/*2968Hz*/0,/*3125Hz*/0,/*3281
Hz*/0,/*3437Hz*/0,/*3593Hz*/0,/*3750Hz*/0,/*3906Hz*/0,/*4062Hz*/0,/*42
19Hz*/20, /*4375z*/ 0, /*4531Hz*/0, /*4688Hz*/ 2, /*4844Hz*/ 0,
/*5000Hz*/ 4, /*5156Hz*/0, /*5313Hz*/0, /*5469Hz*/ 4, /*5625Hz*/0,
/*5781Hz*/ 0, /*5938Hz*/ 0, /*6094Hz*/ 0 , /*6250Hz*/ 0, /*6406Hz*/0,
/*6563Hz*/ 0, /*6719Hz*/ 0, /*6875Hz*/ 8, /*7031Hz*/ 0, /*7188Hz*/ 0,
/*7344Hz*/ 0, /*7500Hz*/ 0, /*7566Hz*/0, /*7813Hz*/0};
```

```
// Patro 2, fase de xoc
long patro2[] =
{/*0Hz*/669,/*156Hz*/846,/*312Hz*/2014,/*468Hz*/1256,/*625Hz*/164,/*78
1Hz*/217,/*937Hz*/54,/*1093Hz*/684,/*1250Hz*/78,/*1406Hz*/67,/*1562Hz*
/89,/*1718Hz*/360,/*1875Hz*/90,/*2031Hz*/75,/*2187Hz*/40,/*2343Hz*/101
,/*2500Hz*/269,/*2656Hz*/96,/*2812Hz*/0,/*2968Hz*/0,/*3125Hz*/0,/*3281
Hz*/0,/*3437Hz*/30,/*3593Hz*/0,/*3750Hz*/58,/*3906Hz*/0,/*4062Hz*/30,/
*4219Hz*/0, /*4375z*/ 0, /*4531Hz*/0, /*4688Hz*/ 81, /*4844Hz*/ 74,
/*5000Hz*/ 0, /*5156Hz*/30, /*5313Hz*/30, /*5469Hz*/53, /*5625Hz*/30,
/*5781Hz*/30, /*5938Hz*/ 0, /*6094Hz*/ 0 , /*6250Hz*/ 0, /*6406Hz*/21,
/*6563Hz*/ 0, /*6719Hz*/ 0, /*6875Hz*/ 0, /*7031Hz*/ 11, /*7188Hz*/ 0,
/*7344Hz*/ 0, /*7500Hz*/ 0, /*7566Hz*/0, /*7813Hz*/0};

// Mitjana dels 2 valors maxims del patro 1 i del patro 2
const long pmx1 = 3142;

  ////////////////////////////////////////////////////////////////////
 //            CONFIGURACIÓ INICIAL DEL NODE SENSOR                //
////////////////////////////////////////////////////////////////////

void setup()
{
  // Configuració de l'ADC del microfon
  adcInit(FS_INTERVAL, microfon, ADC_REF_AVCC);

  // Inicialització NRF24
   SPI.begin();
   radio.begin();
   network.begin(90, aquest_node);

  // Habilitacio acusament de recepció NRF24
  radio.setAutoAck(1);

  // Temps d'espera per reenviar paquet (multiple de 250us) i numero
  d'intents
  radio.setRetries(8, 11);

  // Velocitat transmissió radio del NRF24
  radio.setDataRate(RF24_250KBPS);

  // Identificació del node sensor
  dades.Id = 10.1;
}

  ////////////////////////////////////////////////////////////////////
 //                  BUCLE PRINCIPAL DEL NODE SENSOR               //
////////////////////////////////////////////////////////////////////

void loop()
{

  // Reset variables
  mx1 = 0;
  mx2 = 0;

  // Comptador
  t1 = t1 + 1;

  // Crida a la funcio que realitza adquisicio
  adquisiciofht();
```

```
//Buscar els 2 màxims del vector capturat
for (i = 0; i < 50; i++) {
  dist1 = (patro1[i] - fht_lin_out[i]);
  if (fht_lin_out[i] > mx1) {
      mx1 = fht_lin_out[i];
  } else if (fht_lin_out[i] > mx2) {
      mx2 = fht_lin_out[i];
  }
}

//Mitjana dels 2 maxims
mx1 = (mx1 + mx2) / 2;

// Anivellació del vector capturat
if (mx1 > 150) {
  equiv1 = pmx1 / mx1;
  for (i = 0; i < 50; i++) {
    if (equiv1 >= 1) {
        fht_lin_out[i] = fht_lin_out[i] * equiv1;
    } else if (equiv1 < 1) {
        fht_lin_out[i] = fht_lin_out[i] * equiv1;
    }
  }
}

// Relació de distàncies entre el vector anivellat i els patrons
rel1acum = 0.0;
rel2acum = 0.0;
for (i = 0; i < 50; i++) {
  dist1 = (patro1[i] - fht_lin_out[i]);
  dist2 = (patro2[i] - fht_lin_out[i]);
  if (dist1 < 0) {
      dist1 = dist1 * (-1);
  }
  if (dist2 < 0) {
      dist2 = dist2 * (-1);
  }
  dist1total = (patro1[i] + fht_lin_out[i]);
  dist2total = (patro2[i] + fht_lin_out[i]);
  rel1 = round((100 * dist1) / dist1total);
  rel2 = round((100 * dist2) / dist2total);
  if (rel1 < 0) {
      rel1 = rel1 * (-1);
  }
  if (rel2 < 0) {
      rel2 = rel2 * (-1);
  }
  rel1 = 100 - rel1;
  rel2 = 100 - rel2;
  rel1acum = rel1 + rel1acum;
  rel2acum = rel2 + rel2acum;
}
rel1acum = round(rel1acum / (i - 1));
rel2acum = round(rel2acum / (i - 1));

// Coincidencia del patro 1 i inici de la frenada
if (rel1acum > limit1) {
    n1 = n1 + 1;
    inicifrenada = true;
}
```

```
// Si s'ha hi ha coincidencia del patro 1, temporitzador t3
if (inicifrenada){
 t3 = t3 + 1;
}

//Condició 1 supeditada a t3
if (t3 > tlimit3 and frenada == false){
  n1 = 0;
  n2 = 0;
  t2 = 0;
  t3 = 0;
  inicifrenada = false;
}

// Compliment condició 1
if (n1 > nlimit1) {
  n1 = 0;
  n2 = 0;
  t2 = 0;
  t3 = 0;
  inicifrenada = false;
  frenada = true;
}

// Si s'ha hi ha compliment condicio 1, temporitzador t2
if (frenada)
{
 t2 = t2 + 1;
}

// Coincidencia del patro 2
if (rel2acum > limit2) {
    n2 = n2 + 1;
}

// Condició 2 supeditada a t2
if (t2 > tlimit2){
  n1 = 0;
  n2 = 0;
  t2 = 0;
  t3 = 0;
  frenada = false;
}

// Compliment condició 2
if (n2 > nlimit2 and t2<tlimit2 and t2>tlimit1 and frenada) {
  n1 = 0;
  n2 = 0;
  t2 = 0;
  t3 = 0;
  frenada = false;
  dades.Acc = 20.2;
}

// Si repetició de bucle és igual a 100.000 (30 minuts) o hi ha un
accident
if (t1 > 99999 or dades.Acc > 10.0) {

  // Si no hi ha accident
  if (dades.Acc < 10.0) {
```

```
      //Crida a la funció llegirVcc() de lectura del voltatge intern
      dades.BatS = 1126400 / (llegirVcc() * 10);
      dades.BatS = round(dades.BatS);
      dades.BatS = dades.BatS / 100;

      // Lectura pin analogic del voltatge de la placa solar
      dades.SolS  = analogRead(solar);
      dades.SolS = dades.SolS * 4.0 / 1023;
      dades.Acc = 0.0;
      t1 = 0;

    // Si hi ha accident
    } else if (t1 < 100000) {
        dades.BatS = 0.0;
        dades.SolS = 0.0;
    }

    // Despertar NRF24
    radio.powerUp();
    radio.begin();
    network.begin(90, aquest_node);
    network.update();
    SPI.begin();
    RF24NetworkHeader header(altre_node);

    // Enviament paquet de dades
    bool ok = network.write(header, &dades, sizeof(dades));
  }

  // Temporitzador t1 (repeticions de bucle)
  t1 = t1 + 1;

  // Adormir el NRF24
  SPI.end();
  radio.powerDown();
}

  ////////////////////////////////////////////////////////////////
 //                      FUNCIONS DEL NODE SENSOR                //
////////////////////////////////////////////////////////////////

//Funció d'adquisició i de transformada de la FHT
uint16_t adquisiciofht()
{
  // Inici de l'ADC del microfon
  adcStart();
  BufPtr = 0;

  // Esperar que el buffer de l'ADC tinguui 256 dades
  while (BufPtr < FHT_N) {
      millis();
  }

  // Enfinestrat ordenament dades de l'adc per a la FHT
  fht_window();
  fht_reorder();

  // Processar la FHT
  fht_run();

  // Output de la FHT, vector de 256 punts
```

```
  fht_mag_lin();
}

// Lectura interna del voltatge de l'alimentació
long llegirVcc() {
  long result;
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1); delay(2);
  ADCSRA |= _BV(ADSC); // Convert
  while (bit_is_set(ADCSRA, ADSC));
  result = ADCL;
  result |= ADCH << 8;
  result = 1126400L / result;
}

// Configuració i inicialització de l'ADC del microfon
void adcInit(uint32_t ticks, uint8_t pin, uint8_t ref)
{
  if (ref & ~((1 << REFS0) | (1 << REFS1))) {
      ErrorCond = true;
      return;
  }
  ADMUX = ref | (pin & 7);
#if RECORD_EIGHT_BITS
    ADMUX |= (1 << ADLAR);
#endif
  ADCSRB = (1 << ADTS2) | (1 << ADTS0);
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
  if (pin < 8) {
    ADCSRB &= ~(1 << MUX5);
    DIDR0 |= 1 << pin;
  } else {
    ADCSRB |= (1 << MUX5);
    DIDR2 |= 1 << (7 & pin);
  }
#else
    if (pin < 6) DIDR0 |= 1 << pin;
#endif
  #if ADPS0 != 0 || ADPS1 != 1 || ADPS2 != 2
  #error unexpected ADC prescaler bits
  #endif
  uint8_t adps;
  for (adps = 7; adps > 0; adps--) {
      if (ticks >= (MIN_ADC_CYCLES << adps)) break;
  }
  if (adps < 3)
  {
      ErrorCond = true;
      return;
  }
  ADCSRA = adps;
  ticks >>= adps;
  ticks <<= adps;
  TCCR1A = 0;
  uint8_t tshift;
  if (ticks < 0X10000) {
      TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS10);
      tshift = 0;
  } else if (ticks < 0X10000 * 8) {
      TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11);
      tshift = 3;
  } else if (ticks < 0X10000 * 64) {
```

```
      TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS11) | (1 <<
CS10);
      tshift = 6;
  } else if (ticks < 0X10000 * 256) {
      TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS12);
      tshift = 8;
  } else if (ticks < 0X10000 * 1024) {
      TCCR1B = (1 << WGM13) | (1 << WGM12) | (1 << CS12) | (1 <<
CS10);
      tshift = 10;
  } else {
      ErrorCond = true;
      return;
  }
  ticks >>= tshift;
  ICR1 = ticks - 1;
  OCR1B = 0;
  ticks <<= tshift;
}

// Inici de l'ADC del microfon
void adcStart() {
  ADCSRA |= (1 << ADATE)  | (1 << ADEN) | (1 << ADIE) | (1 << ADSC) ;

  //Habilitació interrupcions del timer1
  TIMSK1 = (1 << OCIE1B);
  TCNT1 = 0;
}

// Interrupció de l'ADC del microfon
ISR(ADC_vect) {
#if RECORD_EIGHT_BITS
    uint8_t d = ADCH;
#else
    uint8_t low = ADCL;
    uint8_t high = ADCH;
    uint16_t d = (high << 8) | low;
#endif
int k = d - 0x0200;
k <<= 6;

  // Només guardar en el vector capturat si el buffer té menys de 256
  dades
  if (BufPtr < FHT_N)
  {
      // Vector capturat
      fht_input[BufPtr] = k;
  }
    BufPtr++;
}
ISR(TIMER1_COMPB_vect) {}
```

```
    /* COORDINADOR                                                          *
     * Xarxa de sensors d'audio per a la deteccio d'accident urbans *
     * Dani Marti Verge                                                     *
     * v.1.03                                                               *
     ************************************************************/


  ////////////////////////////////////////////////////////////////////
 //                  LLIBRERIES DEL COORDINADOR                    //
////////////////////////////////////////////////////////////////////

#include <RF24Network.h>
#include <RF24.h>
#include <SPI.h>

  ////////////////////////////////////////////////////////////////////
 //                            PARAMETRES I VARIABLES                //
////////////////////////////////////////////////////////////////////

// Pins SPI declaració del NRF24
RF24 radio(7, 8);
RF24Network network(radio);

// Adreça del node actual en octal
const uint16_t aquest_node = 00;

// Adreça node a enviar en octal
const uint16_t altre_node = 01;

// Variables del payload a rebre
struct dades_t {
  float Acc;
  float BatS;
  float SolS;
  float Id;
};

// Variables a adquirir en el coordinador
float BatC;
float SolC;

// Matrius de caràcters per a la concatenació
char charsols[10];
char charbats[10];
char characc[10];
char charbatc[10];
char charsolc[10];
char charid[10];

// Entrada analògica de la placa solar
int sensorsolar = A1;

// Declaració tupla per guardar el payload a rebre
dades_t dades;

// Comptadors
int t4;
```

```
  ////////////////////////////////////////////////////////////////////
 //       CONFIGURACIÓ INICIAL DEL COORDINADOR                     //
////////////////////////////////////////////////////////////////////
void setup(void) {

  // Inicialització port série
  Serial.begin(115200);

  // Pin digital per a despertar el ESP8266
  pinMode(3, OUTPUT);

  // Inicialització NRF24
  SPI.begin();
  radio.begin();
  network.begin(90, aquest_node);

  // Velocitat de transmissió radio del NRF24
  radio.setDataRate(RF24_250KBPS);

}

  ////////////////////////////////////////////////////////////////////
 //           BUCLE PRINCIPAL COORDINADOR                          //
////////////////////////////////////////////////////////////////////

void loop(void) {

  // Reset variables
  dades.Id=0.0;

  // Comptador
  t4 = t4 + 1;

  // Si s'ha arribat a la repetició 5
  if (t4 > 4) {

    // Despertar NRF24
    radio.powerUp();
    radio.begin();
    network.begin(90, altre_node);
    network.update();
    SPI.begin();

         // Escoltar si arriba algun paquet
    while ( network.available() ) {
       RF24NetworkHeader header;
       bool ok = network.read(header, &dades, sizeof(dades));

       // Si dades.Id>1.0 ha arribat un nou paquet
       if (dades.Id > 1.0) {

          // Despertar el ESP8266 activant el pin connectat a CH_PD
          digitalWrite(3, HIGH);
          delayMicroseconds(100);

          // Inici port serie per enviar les dades al ESP8266
          Serial.begin(115200);

          // El paquet no es d'accident
          if (dades.Acc < 10.0) {
```

```
    //Crida la funció llegirVcc() llegir voltatge intern
    BatC = 1126400 / (llegirVcc() * 10);
    BatC = round(BatC);
    BatC = BatC / 100;

    //Lectura pin analogic del voltatge de la placa solar
    SolC  = analogRead(sensorsolar);
    SolC = SolC * 4.0 / 1023;

// El paquet es d'accident
} else if (dades.Acc > 10.0) {

    //Assignem 0.0 a les variables SolC i BatC
    BatC = 0.0;
    SolC = 0.0;
}

// Espais i lletres a concatenar
String stringbats = "";
String stringsols = "";
String stringbatc = "";
String stringsolc = "";
String stringacc = "";
String stringid = "";
String bats = "B";
String sols = "S";
String batc = "C";
String solc = "T";
String acc = "A";
String idd = "I";

// Concatenació variable dades.BatS i el prefix B
dtostrf(dades.BatS, 5, 2, charbats);
for (int i = 0; i < sizeof(charbats); i++)
{
    stringbats += charbats[i];
}
bats += stringbats;
bats.remove(5, 10);

// Impressió del string bats pel port serie
Serial.println(bats);

// Concatenació i impressio dades.SolS i el prefix S
dtostrf(dades.SolS, 5, 2, charsols);
for (int i = 0; i < sizeof(charsols); i++)
{
    stringsols += charsols[i];
}
sols += stringsols;
sols.remove(5, 10);
Serial.println(sols);

// Concatenació i impressió BatC i el prefix C
dtostrf(BatC, 5, 2, charbatc);
for (int i = 0; i < sizeof(charbatc); i++)
{
    stringbatc += charbatc[i];
}
batc += stringbatc;
batc.remove(5, 10);
```

```
            Serial.println(batc);

            // Concatenació i impressió SolC i el prefix T
            dtostrf(SolC, 5, 2, charsolc);
            for (int i = 0; i < sizeof(charsolc); i++)
            {
                stringsolc += charsolc[i];
            }
            solc += stringsolc;
            solc.remove(5, 10);
            Serial.println(solc);

            // Concatenació i impressió dades.Acc i el prefix A
            dtostrf(dades.Acc, 5, 2, characc);
            for (int i = 0; i < sizeof(characc); i++)
            {
                stringacc += characc[i];
            }
            acc += stringacc;
            acc.remove(5, 10);
            Serial.println(acc);

            // Concatenació dades.Id i el prefix I
            dtostrf(dades.Id, 5, 2, charid);
            for (int i = 0; i < sizeof(charid); i++)
            {
                stringid += charid[i];
            }
            idd += stringid;
            idd.remove(5, 10);
            Serial.println(idd);

            // Desalimentar el pin en mode power down
            digitalWrite(3, LOW);
            delayMicroseconds(100);
     }
    }

    // Adormir el NRF24 i apagar el port serie
    SPI.end();
    radio.powerDown();
    Serial.end();

    // Resetejar comptador t4
    t4=0;
  }
}

  ////////////////////////////////////////////////////////////////////
 //              FUNCIONS DEL COORDINADOR                          //
////////////////////////////////////////////////////////////////////

// Lectura interna del voltatge d'alimentació
long llegirVcc() {
  long result; // Read 1.1V reference against AVcc
  ADMUX = _BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1); delay(2); //
Wait for Vref to settle
  ADCSRA |= _BV(ADSC); // Convert
  while (bit_is_set(ADCSRA, ADSC)); result = ADCL; result |= ADCH <<
8; result = 1126400L / result; // Back-calculate AVcc in mV return
result;}
```

```
    /* ESP8266                                                          *
     * Xarxa de sensors d'audio per a la deteccio d'accident urbans *
     * Dani Marti Verge                                                 *
     * v.1.03                                                           *
     ********************************************************************/

  ////////////////////////////////////////////////////////////////////
 //                  LLIBRERIES DE L'ESP8266                       //
////////////////////////////////////////////////////////////////////

#include <ESP8266WiFi.h>

  ////////////////////////////////////////////////////////////////////
 //              PARÀMETRES DEL WIFI I DEL SERVIDOR                //
////////////////////////////////////////////////////////////////////

// Servidor ThingSpeak
String apiKey = "74YPULIPVKTJTQAU";
const char* server = "api.thingspeak.com";

// SSID WiFi
const char* ssid = "Orange";
const char* password = "A5E1D37D3C";

// Declaració pin GPIO 0 connectat a CH_PD
int pinON = 0;

// Inicialització llibreria
WiFiClient client;

  ////////////////////////////////////////////////////////////////////
 //                    CONFIGURACIONS INICIALS                     //
////////////////////////////////////////////////////////////////////

void setup() {

  // Cada vegada que es desperta, es reinicia, i el pinON activa
  CH_PD
  pinMode(pinON, OUTPUT);
  digitalWrite(pinON, HIGH);
  delayMicroseconds(100);

  // Inici comunicació serie amb l'Arduino
  Serial.begin(115200);

  // Connexió a la xarxa
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
      delayMicroseconds(100);
  }
}
  ////////////////////////////////////////////////////////////////////
 //                         BUCLE PRINCIPAL                        //
////////////////////////////////////////////////////////////////////

void loop() {

  // Connexió al servidor
  String bufferArray[ 4 ];
  if (client.connect(server, 80)) {
    String postStr = apiKey;
```

```
// Emmagatzament dades del port serie
if ( Serial.available() > 1) {
  for (int i = 0; i < 3; i++) {
      bufferArray[ i ] = Serial.readStringUntil('\n');
  }

  // Descodificació primer caràcter de la primera dada
  String BatS = bufferArray[0];
  if (BatS.charAt(0) == 'B') {
      BatS.remove(0, 1);

      // Guardar al camp que relaciona amb al servidor
      postStr += "&field1=";
      postStr += String(BatS);
  }

  // Descodificació i guardar al camp
  String SolS = bufferArray[1];
  if (SolS.charAt(0) == 'S') {
      SolS.remove(0, 1);
      postStr += "&field2=";
      postStr += String(SolS);
  }
  String BatC = bufferArray[2];
  if (BatC .charAt(0) == 'C') {
      BatC .remove(0, 1);
      postStr += "&field3=";
      postStr += String(BatC );
  }
  String SolC = bufferArray[2];
  if (SolC.charAt(0) == 'T') {
      SolC.remove(0, 1);
      postStr += "&field4=";
      postStr += String(SolC);
  }
  String Acc = bufferArray[2];
  if (Acc.charAt(0) == 'A') {
      Acc.remove(0, 1);
      postStr += "&field5=";
      postStr += String(Acc);
  }
  String Id = bufferArray[2];
  if (Id.charAt(0) == 'I') {
      Id.remove(0, 1);
      postStr += "&field6=";
      postStr += String(Id);
  }
}


// Enviament de les dades al servidor
postStr += "\r\n\r\n";
client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
client.print("X-THINGSPEAKAPIKEY: " + apiKey + "\n");
client.print("Content-Type: application/x-www-form-urlencoded\n");
client.print("Content-Length: ");
client.print(postStr.length());
client.print("\n\n");
```

```
    client.print(postStr);

  }

  // Finalitzazció conexxió al servidor
  client.stop();

  // Deshabilitació pin Power On
  digitalWrite(pinON, LOW);
  delayMicroseconds(100);
}
```