```
/* Control de respirador DQ3D NICA
 * Mayo 2020
 *
 * Arduino Nano
 * Motor Nema23
 * DQ3D.org

Ver 09
Sin MicroStep
min max PasosInsp =150 a 270
presionmin = -800P;  -8cmH2O
cambios calculo tentrepasos y pausita

Ver20
*********************************************************************************************
*Recordar cambios de pines  y recorte de mecate o pasos totales para equipos reconstruidos
*********************************************************************************************
*2 sensores
*  BMP280 0x76 distal   Sensor 2  BME1
*y BME280 0x77 proximal Sensor 1  BME2
*
*74HC595 para multiplicar pines de led, 1 por tipo de alarma
  595 pin 14 es Serial data  input(DS o SER),
  595 pin 12 es latch (Latch Clock(ST_CP o RCLK),
  595 pin 11 es clock (Shift clock(SH_CP o SRCLK)
  Enciende led x  si data es 2 a la potencia x . Si led x y led y, se suman las potencias
  o sino    bitWrite(data, desiredBit, desiredState) o bitSet(data, desiredBit)lo pone en 1;
*Opcion de respiraciones por minuto o reaccionar a intento respiratorio

*Pendientes
    Pulir TODOMAL, lucecitas
    YA CAMBIO DE PIN 11 y pin 9 porque interfiere con TONE
    YA Cambiar libreria para BMEEEEE280
    Medir humedad BME280, alarmas y variables
    YA sistema de chequeo de integridad de sensores sobre la marcha
    YA Reducir uso de serial cuando no hay computadora (version autonoma)
*/
//DEBUGGING*****************************************
//#define DEBUG

#ifdef DEBUG
 #define DEBUG_PRINT(x)     Serial.print (x)
 #define DEBUG_PRINTDEC(x)     Serial.print (x, DEC)
 #define DEBUG_PRINTln(x)  Serial.println (x)
#else
 #define DEBUG_PRINT(x)
 #define DEBUG_PRINTDEC(x)
 #define DEBUG_PRINTln(x)
#endif

// No enviar mensajes a pantalla (Uso sin computadora)
//#define SISERIAL

#ifdef SISERIAL
 #define SISERIAL_PRINT(x)     Serial.print (x)
 #define SISERIAL_PRINTDEC(x)     Serial.print (x, DEC)
 #define SISERIAL_PRINTln(x)  Serial.println (x)
#else
 #define SISERIAL_PRINT(x)
 #define SISERIAL_PRINTDEC(x)
 #define SISERIAL_PRINTln(x)
#endif

//incluir librerias necesarias para sensores
```

```
#include <BME280I2C.h>
#include <Wire.h>

#define SERIAL_BAUD 115200

//Variables médicas modificables en firmware
#define ciclosmin 16     // Minimo de ciclos por minuto
#define ciclosmax 25     // Maximo de ciclos por minuto
const float FInsp = .5; // fracción inspiratoria = Tiempo Inspiracion/Tiempo total ciclo (.42
recomendado) SE PUEDE PONER UN POTENCIOMETRO PARA VARIAR
const float TMeseta = 0;      // Tiempo de Meseta en milisegundos (Max500)(Presión alta
mantenida antes de pasar a Exhalación)

float presionmax = 2942; // Presion maxima en cualquier momento en Pascal. 30cmH2O     PARA EL
EQUIPO.
float presionmin = -800; // Presion minima durante espiracion en Pascal. -5cmH2O = -490
Señal advertencia  v09Eq07 -10
#define tempmax 39       // Temperatura maxima del aire
                         // volumen segun ambu y posicion del reostato

//Variables medicas derivadas
float CPM = ciclosmin;    // Ciclos por minuto inicio
unsigned long TCiclo = 60000L/CPM;    // duración del ciclo respiratorio (en milisegundos)
unsigned long Tinicio = millis();     // Marca tiempo de inicio de la respiración
unsigned long pausita = 0 ;               // pausita adicional para ajustar CPM

//Definición pines Digitales y Análogos // XXX Color alambre
#define ResetAlarmaPin 2   // Roj (Interrupt) Boton reset alarma
#define cierrePin 3        //     (Interrupt) Pin libre
#define origenMotPin 4     // Ama Pin de sensor optico para confirmar si Motor regresó a
origen. No se han saltado pasos (motor en posicion adecuada) despues de un ciclo
#define direccionPin 5     // Ama Pin direccion Motor
#define pasosPin 6         // Pur Pin step pasos Motor
#define botonPin 7         // Bla Pin pulsador  (encendido ciclo) y seleccion de tipo de ciclo
a partir de version 20 (RPM o Disparo)
#define enablePin 8        // Gri Pin activa y desactiva motor Motor (Verificar con el driver.
Driver HY-DIV268N 6600 activa motor con LOW.
#define origenBraPin 11    // Ama Pin (endstop) para confirmar brazo en punto de partida  ERA
PIN 9 ahora en 11
#define dataPin 10         // Ama 595 pin 14 es Serial data  input(DS o SER), Amarillo
#define latchPin 9         // Pur 595 pin 12 es latch (Latch Clock(ST_CP o RCLK), Purpura  ERA
PIN 11 ahora en 9, interferia con TONE()
#define clockPin 12        // Gri 595 pin 11 es clock (Shift clock(SH_CP o SRCLK) Gris
#define alarmaPin 13       // Roj Parlante alarma    //Ver20
//(Switch mecanico) para confirmar cierre de caja. SIN pin, solo led con switch es suficiente

//Pines Analogos
int potCPM;                // Azu A0 lectura del potenciometro de velocidad (ciclos por minuto)
,  Leída
int potPI = 0;             // Ama A1 lectura del potenciometro de volumen (pasos por ciclo) ,
Leída
#define SDAPin 4           // Ver SDA de I2C Data (BMP280 purpura, etc)
#define SCLPin 5           // Ama SCL de I2C Clock (BMP280 amarillo, etc)

int CALIBRA = 1;           // Chequeo antes de iniciar operación (1) y en marcha para busca de
punto origen motor (2)
// local var en main loop   unsigned long Tinicio = 0L;      // Tiempo de inicio de ciclo en
milisegundos,
int ALARMA = 0;            // Diferentes alarmas (estado de alarma por falla) generada
                          /* 0= Normal,
                             1= falla de motor/origen,
                             2= Falla de origen del brazo,
                             3= Exceso de presion
                             4= Presion demasiado baja
```

```
                              5= Sobre temperatura
                              6= Sensor 1  presion no detectado
                              7= Sensor 2  presion no detectado          Ver20
                              8= Humedad demasiado baja
                              9= Señal de ok
                            */
byte AlarmaLeds = 0;
volatile int ResetAlarma = 0;         // variable for reading the pushbutton status

//Variables para BMP280
BME280I2C::Settings settings1(   //Sensor Proximal si Hay 2 sensores
   BME280::OSR_X1,
   BME280::OSR_X1,
   BME280::OSR_X8,    //Oversampling pressure 8, mayor precision
   BME280::Mode_Forced,
   BME280::StandbyTime_500us,
   BME280::Filter_Off,
   BME280::SpiEnable_False,
   0x77 // I2C address. I2C specific.
);
BME280I2C::Settings settings2(   //Sensor Distal siempre
   BME280::OSR_X1,
   BME280::OSR_X1,
   BME280::OSR_X8,    //Oversampling pressure 8, mayor precision
   BME280::Mode_Forced,
   BME280::StandbyTime_500us,
   BME280::Filter_Off,
   BME280::SpiEnable_False,
   0x76 // I2C address. I2C specific.
);
BME280I2C bme1(settings1);
BME280I2C bme2(settings2);

float FactCorrTemp = 0;    //Para calibrar BMP en base a BME,  Temp1 = temp + FactCorrTemp
            // presion se calibra por ser presion relativa a presion inicial
float temp1(NAN), hum(NAN), pres1(NAN);  //se define como no es un numero
float temp2(NAN), pres2(NAN);    //se define como no es un numero
float cmH2O = -2;           // Presion ultima medida (de sensor2)en cmH2O  Ver20
float presionI1 =0;         // Presion Inicial, Ambiente antes de medición, en hPa
float presionI2 =0;         // Presion Inicial, Ambiente antes de medición, en hPa
bool primeramedida = true;  // para hacer medida dentro de loop, a ver si mide correctamente
float presionMax1 = 0;      //  en cmH2O
float presionMin1 = 0;      //  en cmH2O
float presionMax2 = 0;      //  en cmH2O
float presionMin2 = 0;      //  en cmH2O
#define  TCheqPT 100        // Tiempo en  millis entre chequeos de Presion
unsigned long TultmedPT =0; // Tiempo transcurrido desde ultima medida en  millis
bool DosSensores = false;   // Si hay dos sensores, es un equipo para activacion por disparo
bool CicloCPM = true;
float presionDisparo = -2;       // Presion de disparo de ciclo   Ver20
const float presionMinDisp = -.3;
const float presionMaxDisp = -5;
////////////////////////////////////////////////////////////

//Variables mecánicas
#define  MicroStep 2        // Micropasos utilizados en el driver para suavizar motor
(1/Microstep) v09Eq07
const int PasosInsp = 270*MicroStep;   // v20Eq8   Default 270  Pasos maximos por Inspiración
(depende de mecanismo usado), 200pasos por vuelta*Microstep, definida (define volumen)v09Eq07
const int PasosMin = 150*MicroStep;    // v20Eq8   Default 150
const bool dirEspira = false;          // dirEspira = 0 es inspira, dirEspira = 1 es espira (si
el motor esta conectado inverso, cambiar)
bool dirMotor=!dirEspira;   // Motor siempre empieza con inspiración
#define  Vueltas 2          // para 10 mm diametro, debe dar un maximo de dos vueltas
```

```
bool BrazoHome = false;     // Suponemos que el brazo no esta en posicion correcta
long Tentrepasos = 800L ;   // Tiempo entre pasos para asegurar un numero de ciclos por minuto
(Pausa en microsegundos, calculada)Driver permite hasta 20 micros v09Eq07
const long TentrepasosMin = 500L;        // tiempo minimo entre pasos
int regresaorigen = 1;      // Tipo de regreso a Origen
int ciclosCalibra = 0;      // Control de ciclos para recalibrar
//int optVuel = 0;          // Vueltas completas del motor para llegar al punto maximo
inspiracion
// varaible local float Trespiro = 0;

void setup() {

  resetearAlarma();
// inicializamos pines.
  pinMode(alarmaPin, OUTPUT);
  pinMode(cierrePin, INPUT);
  pinMode(origenMotPin, INPUT);
  pinMode(direccionPin, OUTPUT);
  pinMode(pasosPin, OUTPUT);
  pinMode(botonPin, INPUT);
  pinMode(enablePin, OUTPUT);
  pinMode(origenBraPin, INPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(dataPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(ResetAlarmaPin, INPUT_PULLUP);      // initialize the pushbutton pin as an input:
  attachInterrupt(0, ParaAlarma, FALLING);    // Attach an interrupt to the ISR vector Ver20

  Serial.begin(SERIAL_BAUD);
    while(!Serial) {} // Wait
  Wire.begin();

  Serial.println(F("************************Inicializando equipo*****************************"));

  Serial.print(F("Probando Sensores: "));
  if(!bme1.begin())  Serial.println(F("No hay BME280 sensor 1 Proximal!"));
  switch(bme1.chipModel())  {
     case BME280::ChipModel_BME280:
       Serial.println(F("Sensor1 bien BME280 sensor 1! Success."));
       break;
     case BME280::ChipModel_BMP280:
       Serial.println(F("Sensor1 bien BMP280 sensor 1! No Humidity available."));
       break;
     default:
       Serial.println(F("Found UNKNOWN sensor 1! Error!"));
       ALARMA = 6;
       alarma();
  }
  if(!bme2.begin())  {
      Serial.println(F("No hay BME280 sensor 2 Distal!"));
      ALARMA = 7;
      alarma();
  }
  switch(bme2.chipModel())  {
     case BME280::ChipModel_BME280:
       Serial.println(F("Sensor2 bien BME280 sensor 2! Success."));
       break;
     case BME280::ChipModel_BMP280:
       Serial.println(F("Sensor2 bien  BMP280 sensor 2! No Humidity available."));
       break;
     default:
       Serial.println(F("Found UNKNOWN sensor 2! Error!"));
       ALARMA = 7;
       alarma();
```

```
  }
  bme1.read(pres1, temp1, hum);
//  if (temp1 != temp1) {
//   Serial.println(F("Sensor 1 fallando."));
//     ALARMA = 6;
//     alarma ();
//   }
  bme2.read(pres2, temp2, hum);
//  if (temp2 != temp2) {
//    Serial.println(F("Sensor 2 fallando."));
//     ALARMA = 7;
//     alarma ();
//  }
  if (temp1 == temp1 && temp2 == temp2)  DosSensores = true;   // Hay dos sensores
  FactCorrTemp = temp2 - temp1;    //Luego siempre Temp1 = temp + FactCorrTemp
  if (DosSensores){
      bme1.read(pres1, temp1, hum);               // Primera medida BMP presion sensor 1:
      if (temp1 != temp1) {
        Serial.println(F("Sensor 1 fallando."));
        ALARMA = 6;
        alarma ();
      }
/*      presionI1 = pres1;
      Serial.print(presionI1/100);
      Serial.println(F(" hPa Presión Atmosférica de referencia 1 "));
  //    presionmax = pres1+presionmax;
  //    presionmin = pres1+presionmin;
  //    presionMax1 = ((pres1- presionI1)/100)*1.019744289 ;
  //    presionMin1 = presionMax1 ;
      //    Serial.print(F("pascales "));
      //    Serial.println(pres1);
      //     Serial.print(F("Presión DISTAL  Máxima: "));
      //     Serial.print(presionMax1);
      //     Serial.print(F("  Minima:"));
      //     Serial.println(presionMin1); */
  }
  bme2.read(pres2, temp2, hum);             // Primera medida BMP presion sensor 2:
  if (temp2 != temp2) {
    Serial.println(F("Sensor 2 fallando."));
    ALARMA = 7;
    alarma ();
  }
 /* presionI2 = pres2;
      Serial.print(presionI2/100);
      Serial.println(F(" hPa Presión Atmosférica de referencia 2 "));
 // presionMax2 = pres2;
 // presionMin2 = pres2;*/


   //preflight check
//  Tinicio = millis();
  digitalWrite(enablePin, HIGH);                      // desactiva el driver del motor
  if (!digitalRead(origenMotPin)) RegresaOrigen();
  Serial.print(F(" RegresaOrigen terminado. Logica Origen y brazo: "));
  Serial.print(digitalRead(origenMotPin) );
  Serial.println(digitalRead(origenBraPin));
  if (!digitalRead(origenBraPin)){
    ALARMA =2;
    alarma();
  }
  chequeoPT();

/*  delay (1000);
  if (ALARMA > 0){    //no sigue*****************************************************
```

```
    DEBUG_PRINT(F("DEBUG Alarma activada durante setup. Equipo parqueado ALARMA:"));
    DEBUG_PRINT(ALARMA);
    TODOMAL();
  }
    DEBUG_PRINT(F("DEBUG Alarma ALARMA:"));
    DEBUG_PRINT(ALARMA);
    */
  if (ALARMA == 0) {
    DEBUG_PRINT(F("DEBUG Alarma2 ALARMA:"));
    DEBUG_PRINT(ALARMA);
    CALIBRA = 0;
    TODOBIEN();   // .........
    resetearAlarma();
  }
}

void chequeoPT() {                                    //chequear presion Sensores 1 y 2
  if (DosSensores){
    bme1.read(pres1, temp1, hum);             // Lectura sensor 1 Proximal (0x77) es BME
    if (pres1 != pres1) {
      Serial.println(F("Sensor 1 fallando."));
      ALARMA = 6;
      alarma ();
    }
    if (primeramedida){
    //***   nueva lectura para evitar falla inicio presion referencia
    delay(100);
    bme1.read(pres1, temp1, hum);             // Lectura sensor 1 Proximal (0x77) es BME
      presionI1 = pres1;
      Serial.print(presionI1/100);
      Serial.println(F(" hPa Presión Atmosférica de referencia 1 operando"));
      presionmax = pres1+presionmax;
      presionmin = pres1+presionmin;
    }
    if (pres1 < presionmin){
        Serial.print(F("bajo MIN Proximal pascal leidos   "));
        Serial.print(pres1);
        Serial.print(F("    , Pascal mínimos"));
        Serial.println(presionmin);
        ALARMA = 4;
        alarma();
    }
    if (pres1 > presionmax){
        Serial.print(F("sobre MAX Proximal pascal leidos   "));
        Serial.print(pres1);
        Serial.print(F("    , Pascal máximos"));
        Serial.println(presionmax);
        ALARMA = 3;
        alarma();
    }
    pres1= ((pres1- presionI1)/100)*1.019744289;    // lo convierte en cmH2O
    if (presionMin1 > pres1) presionMin1 = pres1;   //  en cmH2O
    if (presionMax1 < pres1) presionMax1 = pres1;   //  en cmH2O
  }

  bme2.read(pres2, temp2, hum);                     //lectura del sensor 2 Distal (0x76) BMP
  if (pres2 != pres2) {
    Serial.println(F("Sensor 2 fallando."));
    ALARMA = 7;
    alarma ();
  }
  if (primeramedida){
  //***   nueva lectura para evitar falla inicio presion referencia
  delay(100);
```

```
  bme2.read(pres2, temp2, hum);                    //lectura del sensor 2 Distal (0x76) BMP
    Serial.print(pres2/100);
    Serial.println(F(" hPa Presión Atmosférica de referencia 2 operando"));
      if (!DosSensores) {
        presionI2 = pres2;
        presionmax = pres2+presionmax;
        presionmin = pres2+presionmin;
      }
    presionMax2 = pres2;
    presionMin2 = pres2;
    primeramedida = false;
  }
  if (!DosSensores) {
    if ((pres2 - presionmin) < -4000) pres2 = presionmin; //dato
    malo.................................
    if (pres2 < presionmin){
      Serial.print(pres2);
      Serial.print(F("Distal pascal leidos      pascal minimos    "));
      Serial.print(presionmin);
      ALARMA = 4;
      alarma();
    }
    if (pres2 > presionmax){
      Serial.print(pres2);
      Serial.print(F("Distal pascal leidos      pascal maximos    "));
      Serial.print(presionmax);
      ALARMA = 3;
      alarma();
    }
  }

  pres2= ((pres2 - presionI2)/100)*1.019744289;     // lo convierte en cmH2O
  if (presionMin2 > pres2) presionMin2 = pres2;
  if (presionMax2 < pres2) presionMax2 = pres2;

  TultmedPT = millis();
}

void alarma() {
  if  (ResetAlarma ==1) resetearAlarma();

  if (ALARMA >0){                                                  // Alarma sigue hasta
  interrupt
    DEBUG_PRINTln(F("DEBUG Alarma superior a 0*********************************"));
    if (ALARMA ==6)Serial.println(F("Sensor 1 distal fallando o ausente"));
    if (ALARMA ==7)Serial.println(F("Sensor 2 proximal fallando o ausente"));
    tone(alarmaPin, 1000);                                        // encender Alarma
    bitSet(AlarmaLeds, ALARMA-1);
    updateShiftRegister();
  }
}

void TODOBIEN() {                          //señal que anuncia todo bien
   resetearAlarma();
   tone(alarmaPin, 1000);
   delay (500);
   tone(alarmaPin, 2000);
   delay(1000);
   ALARMA = 0;
}

void TODOMAL(){                             // Sirena cuando para definitivo
  DEBUG_PRINT(F("DEBUG TODOMAL*********************************"));
  digitalWrite(enablePin, HIGH);        // desactiva el driver del motor
```

```
  int millisPROVI = millis();
  while(millis() < millisPROVI+180000){ // Durante 3 minutos
    for(int index = 7; index >= 0; index--){
//      bitSet(AlarmaLeds, index);
//      updateShiftRegister();
      for (int i = 100; i <= 200; i++){   // Emitir sonido sirena problema al final
        tone(alarmaPin, i*4);
        delay(20);
      }
//      AlarmaLeds =0;
//      updateShiftRegister();
    }
  }
  noTone(alarmaPin);
  parar();
}

void parar(){
   DEBUG_PRINT(F("DEBUG  PARADA ETERNA*********************************"));
   while(1);
}

void resetearAlarma(){
    AlarmaLeds = 0;
    updateShiftRegister();
    ResetAlarma = 0;
    ALARMA =0;
    noTone(alarmaPin);
}

void RegresaOrigen(){
  digitalWrite(enablePin, HIGH);            // desactiva el driver del motor
  Tentrepasos = TentrepasosMin;             // Maxima aceleracion v09Eq07

  if (CALIBRA ==2){                         // Calibracion sobre la marcha de punto de origen,
  llamada por motor fuera de punto
    dirMotor = !dirEspira;                  // Primera calibracion hacia Inspira (suponemos que
    salto pasos durante inspira
    potPI = PasosInsp;                      // PasosInsp para recalibrar origen v09Eq07
    DEBUG_PRINT(F("DEBUG  Valor de pasos a moverse durante calibración y dir: "));
    DEBUG_PRINT(potPI);
    DEBUG_PRINTln(dirMotor);
    MueveMotor();
      Serial.print(" RegresaOrigen sobre la marcha Motor Brazo: ");
      Serial.print(digitalRead(origenMotPin));
      Serial.println(digitalRead(origenBraPin));
      if (digitalRead(origenMotPin)){         // Si motor regreso a origen
                      DEBUG_PRINTln(F("DEBUG  Hasta aqui llegué8 OrigenMot
                      bien....................."));
        if (digitalRead(origenBraPin)){       // Todo bien
                      DEBUG_PRINTln(F("DEBUG  Hasta aqui llegué9 OrigeBra
                      bien....................."));
                      Serial.println(F(" Recalibró en operación "));
          resetearAlarma();   //cuidado resetea todo, talvez deba solo resetear alarma 1 i 2
          potPI = PasosInsp;
          CALIBRA = 0;
        }
        else if (!digitalRead(origenBraPin)){       // Si brazo no llego a origen, Suponemos
        que se paso una vuelta, y retrocedemos 1 vuelta
          DEBUG_PRINTln(F("DEBUG  brazo no llego a origen, Retrocedemos
          7A....................."));
          dirMotor = dirEspira;
        potPI = PasosInsp;          // v09Eq07
        MueveMotor();
```

```
        if (!digitalRead(origenBraPin)) TODOMAL();
        }
      }
      else TODOMAL();       //No regresó a origen de motor, algo anda mal. Paraliza la maquina.
      Reiniciar
  }
  else if (CALIBRA == 1){                              // Calibracion inicial de punto de origen
    int pasoscalib = 10*MicroStep;
    dirMotor = dirEspira;
    for (potPI = pasoscalib; potPI < PasosInsp; potPI= potPI+pasoscalib){        // PasosInsp
    para limite de pasos v09Eq07 por microstep
      dirMotor = !dirMotor;
      DEBUG_PRINT(F("DEBUG  Valor de pasos a moverse durante calibración y dir: "));
      DEBUG_PRINTln(potPI);
      MueveMotor();
      DEBUG_PRINT("DEBUG  RegresaOrigen inicial : ");
      DEBUG_PRINTln(digitalRead(origenMotPin) );
      if (digitalRead(origenMotPin) && digitalRead(origenBraPin)){
        DEBUG_PRINTln(F("DEBUG  DEBE PASAR POR ACA.............................: "));
        potPI = PasosInsp;
        TODOBIEN();
        CALIBRA = 0;
      }
    }
    if (!digitalRead(origenMotPin) || !digitalRead(origenBraPin)){ ;
      ALARMA = 1;
      alarma();
    }
  }
}

void MueveMotor() {
  int optLog = 0;
  int optVue = 0;
  digitalWrite(enablePin, LOW);         // Activa el driver del motor
  DEBUG_PRINT(F("DEBUG  Hasta aqui llegué4.............POTPI, dirMotor........"));
  DEBUG_PRINT(potPI);
  DEBUG_PRINTln(dirMotor);
  if ((10000 < Tentrepasos) || (Tentrepasos < TentrepasosMin)){
    Serial.print(F("  Microsegundos de pausa entre pasos (Tentrepasos)fuera de rango: "));
    Serial.println(Tentrepasos);
    Tentrepasos = TentrepasosMin;  //****************************************
  }
  for (int i = 0; i <= potPI; i++) {    // BUCLE DE PASOS que mueven motor
    digitalWrite(direccionPin, dirMotor);    // inhala o retrocede
    digitalWrite(pasosPin, HIGH);       // Aqui generamos un flanco de bajada HIGH - LOW menos
    el tiempo de medicion
    delayMicroseconds(Tentrepasos);     // Pequeño retardo para formar el pulso en STEP DQ era 10
    digitalWrite(pasosPin, LOW);        // y el driver de avanzara un paso el motor
    delayMicroseconds(Tentrepasos);     // Retardo para llegar a una velocidad de ciclo menos
    el tiempo de medicion

    if (CALIBRA == 0 ){                                          // Durante operacion normal
      if (digitalRead(origenMotPin) == optLog ){                    // Lleva control de las
      veces que se pasa el punto de origen del motor (vueltas)
        optLog = !optLog;
        if (optLog == 0) optVue = optVue + 1;
      }
      if (millis() - TultmedPT > TCheqPT) chequeoPT();
      if (ALARMA != 0) alarma();
      if (optVue > Vueltas && dirMotor == !dirEspira && i>= 100) { // En inspiracion, Parar
      avance cuando llego a punto de origen
                                                                 // (desinflado maximo AMBU),
                                                                 exceso de pasos
```

```
                                                                    PROBABLEMENTE INNECESARIO
        DEBUG_PRINT(F("DEBUG Se pasa de vueltas........"));
        digitalWrite(enablePin, HIGH);                              // desactiva el driver del
        motor
        ALARMA = 1;
        break;
      }
    }
    else if (CALIBRA >0){                                           // Durante calibracion
    punto origen, para salir en cuanto llega a origen
//    delay (1);
      if (digitalRead(origenMotPin) && digitalRead(origenBraPin)){  // Si ambos puntos de
      origen OK
        digitalWrite(enablePin, HIGH);                              // desactiva el driver del
        motor
        i=potPI;                                                    // Termina movimientos
        saliendo del bucle de pasos
      }
    }
  }                                   // FIN BUCLE DE PASOS que mueven motor

  if (DosSensores){
    bme1.read(pres1, temp1, hum);
    if (temp1 > tempmax){
      ALARMA = 5;
      alarma();
    }
  }
  else {
    bme2.read(pres1, temp1, hum);
    if (temp1 > tempmax){
      ALARMA = 5;
      alarma();
    }
  }
                      DEBUG_PRINTln(F("DEBUG  Fin de MueveMotor 6....................."));
}

void loop() {
   if (ALARMA != 0) alarma();

/*  If low and un sensor suspende bucle
  if high and un sensor ciclo RPM
  If low and dos sensores ciclo disparo
  if high and dos sensores ciclo RPM

*/
 while (digitalRead(botonPin) == LOW && !DosSensores)(1);    //En CPM un sensor no hace nada
 mientras no encienda
   Tinicio = millis();                  // Marca tiempo de inicio de la respiración
   potPI = map(analogRead(A1),0,1023,PasosMin,PasosInsp);         // leemos numero de pasos
   para inspiración, lo cual definirá volumen

   if (digitalRead(botonPin) == HIGH ){     // leemos el boton de encendido. Si esta encendido,
   es por RPM
     CicloCPM = true;
     potCPM = map(analogRead(A0),0,1023,ciclosmin*10,ciclosmax*10); // leemos Ciclos por minuto
     y adaptamos el valor a un numero de (decimos de) ciclo por minuto, entre min y max (para
     rangos mas precisos)
     CPM =  potCPM;
     CPM =  CPM/10;
     TCiclo = 60000/CPM;
    //**************************************
     Tentrepasos = ((((TCiclo*1000L-TMeseta*1000L) * FInsp)/(potPI*.8))/
```

```
     (MicroStep*sqrt(MicroStep)));    // Tiempo entre pasos en microsegundos dividido por los
     micropasos  // Ciclo en milisegundos
    //////     if (Tentrepasos > 987 + TentrepasosMin)  Tentrepasos = Tentrepasos - 987;
    //987microsegundos es tiempo adicional por medio paso dedicado a mediciones y calculos (por
    dos)
     Serial.println();
     Serial.println(F("****************Nuevo Respiro*************"));
     Serial.print(F("  Microsegundos de pausa entre pasos (Tentrepasos): "));
     Serial.println(Tentrepasos);
   }
   else if (DosSensores){  // Si el boton de encendido esta apagado, y hay dos sensores. es un
   ciclo por disparo (intento inspiratorio)
      CicloCPM = false;
      CPM =  25;
      TCiclo = 60000/CPM;
      Tentrepasos = TentrepasosMin*1.5;    //Establecemos la velocidad a un respiro normal de
      600cc a 25 CPM
      int PotpresionDisparo = map(analogRead(A0),0,1023,presionMaxDisp*100,presionMinDisp*100);
      // leemos Presion de disparo de ciclo
      presionDisparo = PotpresionDisparo/100;
      Serial.println(F("****************Esperando disparo Respiro*************"));
      while (presionDisparo < pres1){
        chequeoPT();      //****************Espera el disparo
        if (ResetAlarma) resetearAlarma();
        if (digitalRead(botonPin) == HIGH ){     // SALIR, se cambio a modo de ciclos por
        minuto sobre la marcha
          potCPM = map(analogRead(A0),0,1023,ciclosmin*10,ciclosmax*10); // leemos Ciclos por
          minuto y adaptamos el valor a un numero de (decimos de) ciclo por minuto, entre min y
          max (para  rangos mas precisos)
          CPM =  potCPM;
          CPM =  CPM/10;
          TCiclo = 60000/CPM;
          Tentrepasos = ((((TCiclo*1000L-TMeseta*1000L) * FInsp)/(potPI/2))/
          (MicroStep*sqrt(MicroStep)));
          break;   // se cambio a modo de ciclos por minuto sobre la marcha
        }
        /*DEBUG_PRINT("DEBUG Presion segun sensor en cmH2O:");
        DEBUG_PRINTln(pres1);
        DEBUG_PRINT("DEBUG Presion disparo en cmH2O:");
        DEBUG_PRINTln(presionDisparo);*/
      }
      Serial.println(F(" Respiro disparado....................."));
    }

// CICLO INSPIRACION
      dirMotor = !dirEspira;
      MueveMotor();
      unsigned long iniciomeseta = millis();                      //Meseta
      while (millis() -iniciomeseta < TMeseta){
        if (millis() - TultmedPT > TCheqPT) chequeoPT();
      }
      //      if (ALARMA == 1) RegresaOrigen();// Modificar: si brazo no en origen, o motor no
      en origen, regresaorigen. si no resuelve: ALARMA
      unsigned long Tinspira = millis()-Tinicio;                  // Guarda duracion de
      inspiración

// CICLO ESPIRACION
      Tentrepasos = TentrepasosMin;            // Maxima aceleracion v09Eq07;   // acelerando
      espiracion era 600 (microsegundos)v09Eq07
      dirMotor = dirEspira;
      MueveMotor();
      digitalWrite(enablePin, HIGH);                              // desactiva el driver del
      motor, regresa a origen si AMBU presionado
      DEBUG_PRINT(F("DEBUG  Hasta aqui llegué ESPIRA.............CicloCPM:"));
```

```
      DEBUG_PRINTln(CicloCPM);

// Pausa restante para ciclos por minuto establecidos en millis (solo en CPM)
      if (CicloCPM){
        pausita = (TCiclo - (millis()- Tinicio));
        DEBUG_PRINT(F(" TCiclo en milis:"));
        DEBUG_PRINTln(TCiclo);
        DEBUG_PRINT(F(" Pausita en milis:"));
        DEBUG_PRINTln(pausita);
        if (3000>pausita && pausita>100){
          unsigned long  iniciopausita = millis();
          DEBUG_PRINTln (millis()- iniciopausita);
          while (millis() - iniciopausita < pausita){
            //DEBUG_PRINT (millis()- iniciopausita);
            if (millis() - TultmedPT > TCheqPT) chequeoPT();        // Si paso el tiempo para
            medir, medir
          }
        }
      }

      if (!digitalRead(origenMotPin) || !digitalRead(origenBraPin) ){
        CALIBRA = 2;
              DEBUG_PRINTln(F("DEBUG Hay que recalibrar2 ......................."));
              RegresaOrigen();
                    DEBUG_PRINTln(F("DEBUG  Post recalibrada3....................."));
  Serial.println(F("Recalibró origen."));
         ciclosCalibra = 0;
         CALIBRA = 0;
      }

//cada hora, recalibra contra presion
atmosferica.....************************** 
************************
//Reporte del ciclo respiratorio

     float Trespiro = millis()- Tinicio;      //Duracion respiro
     Serial.print(F(" Fin respiro. Duración: "));
     Serial.println(Trespiro/1000);
     Serial.print(F(" Ciclos por minuto planteados (CPM) / efectivos (60/(Trespiro/1000)) : " ));
     Serial.print(CPM);
     Serial.print(F(" / "));
     Serial.println(60/(Trespiro/1000));
     Serial.print(F(" Fraccion Inspiratoria incluyendo meseta (Tinspira/Trespiro): "));
     Serial.println(Tinspira/Trespiro);
     Serial.print(F(" Milisegundos de pausa al final de espiracion (pausita): "));
     Serial.println(pausita);
     Serial.print(F(" Presión Proximal Máxima: "));
     Serial.print(presionMax1);
     Serial.print(F("  Minima: "));
     Serial.println(presionMin1);
     Serial.print(F(" Temperatura: "));
     Serial.println(temp1);
     Serial.print(F(" Minutos desde inicio del respirador:  "));
     Serial.println(millis()/60000);
/*     DEBUG_PRINTln();
     DEBUG_PRINT(F("  Numero de pasos por este ciclo (PotPI): "));
     DEBUG_PRINTln(potPI);
     DEBUG_PRINT(F("  Tiempo del ciclo en milisegundos (TCiclo): "));
     DEBUG_PRINTln(TCiclo);
     DEBUG_PRINT(F("  Microsegundos de pausa entre pasos (Tentrepasos): "));
     DEBUG_PRINTln(Tentrepasos);
     DEBUG_PRINTln();
     */
  presionMax1 = 0;              //resetea los valores Presion del ciclo
```

```
  presionMin1 = 0;
  presionMax2 = 0;
  presionMin2 = 0;
  if  (ALARMA ==0) resetearAlarma();
}
//FIN DE MAIN LOOP

void updateShiftRegister(){
   digitalWrite(latchPin, LOW);
   shiftOut(dataPin, clockPin, MSBFIRST, AlarmaLeds);
   digitalWrite(latchPin, HIGH);
}

void ParaAlarma() {    //Interrupt para leer el boton de reset Ver20
  ResetAlarma = 1;
}
```