# SEQUITO - DESHIDRATADOR DE FILAMENTO AVANZADO

#### PINES ARDUINO MINI

#define NEOPIXEL\_PIN 3

#define ALARMA 4

#define RELE\_RESISTENCIA 5

#define RELE\_VENTILADOR 6

#define PESO\_DATA2 8

#define PESO\_SCK2 9

#define PESO\_SCK 10

#define PESO\_DATA 11

#define SWITCH 12

#define SENSOR\_INT A2

#define SENSOR\_EXT A3

Nextion TX 7

Nextion RX 13

#### PROCESO INCIAL

1. Configuración por defecto. Se definen todos los valores por defecto de la EEPROM
2. Definir variables globales

SETUP

1. Iniciar Nextion - 115200
2. Iniciar Neopixel
3. Iniciar pines
4. Leer parámetros de la EEPROM. Si la penultima variable guardada en la EEPROM es true, leer de la userSettings EEPROM si es false, leer setDefaults.
5. Iniciar resistencia y ventilador (apagado)
6. Iniciar báscula 1 y 2
7. Iniciar sensores y realizar la primera lectura
8. Iniciar tiempos para el PID y temporizador
9. Inciar PID
10. Alarma de bienvenida
11. Iniciar el DEBUG + CRC EEPROM
12. Iniciar RTC externo
13. Nextion, ir a página inicial

#### loop

1. Leer tiempo actual *now* y comprobar si está encedido el temporizador *(timer)*
2. Escuchar la posible señal del Nextion
3. Leer hora actual del RTC
4. ¿Temporizador?
   1. Si no ha terminado el temporizador o no esta activado: (! temporizador\_end || ! timer )
      1. Proceso PID. Output se interpreta como una cantidad de tiempo en la cual la resistencia esta funcionando por cada ciclo. Convierte el ciclo PWM a tiempo de 0 a WindowSize (100%) Enciende y apaga el relé en consecuencia.
      2. Comprobar gap – Diferencia entre la temperatura real y programada. Cambia la agresividad del PID dependiendo de lo lejos que estemos del objetivo: La razón es que se ponga al máximo enseguida porque la resistencia es lenta.
   2. Temporizador terminado:
      1. Señal sonora de programa terminado
      2. Reiniciar variables
      3. Animación LED de programa terminado
      4. Apaga resistencia y ventilador
5. Leer pesos y temperaturas. Se lee con la frecuencia determinada por WeightReadingRefreshTime (ms) Recomendado no bajar de 1-2s. Los sensores van lentos.
6. Autoapagado del ventilador: Apaga el ventilador en el caso en el que la resistencia apenas se encienda. En este caso no es necesaria la recirculación de aire. Determinado por VentMinDutyCycle.
7. Neopixel, acción que tiene que realizar.
8. Nextion, leer en nxPagina la página actual en Nextion y comprobar si ha cambiado. Si ha cambiado refrescar (nxRefresh\_page)
9. DEBUG, enviado con la frecuencia determinada por serialTime

#### SUBPROCESOS:

void reset\_var()

* LastWeightTime = 0;
* totalTime = 0; //ms
* startTime = 0; //ms
* now = 0; //ms
* timer = 0;

## PID – Resistencia

### Variables Arduino

uint16\_t WindowSize; EEPROM Ciclo entre pasadas del PID (ms)

float vaggKp; EEPROM Very aggresive Kp (factor)

float vaggKi; EEPROM Very aggresive Ki (factor)

float vaggKd; EEPROM Very aggresive Kd (factor)

float aggKp; EEPROM Aggresive Kp (factor)

float aggKi; EEPROM Aggresive Ki (factor)

float aggKd; EEPROM Aggresive Kd (factor)

float consKp; EEPROM Conservador Kp (factor)

float consKi; EEPROM Conservador Ki (factor)

float consKd; EEPROM Conservador Kd (factor)

float setpoint; EEPROM Temperatura SET (ºC)

float input; Input es la lectura real (ºC) que se le dal al PID

float output; Output=myPID.Run(input) da el valor del tiempo que tiene que estar encendido durante un ciclo en ms

uint32\_t windowStartTime; Tiempo inicial para el PID

uint32\_t CurrentWindow Posición actual dentro del ciclo PID: now - windowStartTime;

float gap Distancia desde el setpoint. Cambia la agresividad del PID dependiendo de lo lejos que estemos del objetivo. setpoint - input

bool HeatingDONE; De momento solo utilizado para los LED

### Variables Nextion

int WindowSize

int vaggKp, vaggKp\_ENT, vaggKp\_DEC

int vaggKi, vaggKi\_ENT, vaggKi\_DEC

int vaggKd, vaggKd\_ENT, vaggKd\_DEC

int aggKp, aggKp\_ENT, aggKp\_DEC

int aggKi, aggKi\_ENT, aggKi\_DEC

int aggKd, aggKd\_ENT, aggKd\_DEC

int consKp, consKp\_ENT, consKp\_DEC

int consKi, consKi\_ENT, consKi\_DEC

int consKd, consKd\_ENT, consKd\_DEC

### Proceso

### funciones

## Ventilador

### Variables Arduino

bool vent\_auto\_off; EEPROM Sistema de autoapagado del ventilador ON/OFF

uint8\_t VentMinDutyCycle; EEPROM 0-100% Porcentaje del ciclo de trabajo en el que se apaga el ventilador

uint16\_t dutyCycle Es un ciclo de trabajo virtual, cuanto tiempo está encendida la Resistencia en cada ventana de trabajo.

output / WindowSize \*100

### Variables Nextion

### Proceso

### funciones

## Pesos

### Variables Arduino

#define RESOLUTION 1 posiciones decimales del HX711 PARA MOSTRAR (la lectura es float)

#define AVG\_FACT 30 cuantas lecturas para ponderar con el HX711

uint32\_t WeightReadingRefreshTime EEPROM ms entre lecturas de peso y temp. exterior (mínimo 2seg)

uint32\_t LastWeightTime

//bascula 1

float calibration\_factor1; EEPROM factor m

int32\_t zero\_offset1; EEPROM desfase para peso 0 (b)

uint32\_t tara1\_actual\_g; peso del rollo de filamento

int32\_t tara1\_actual\_offset; desfase para tara + peso del rollo del filamento

volatile float PESO1; lectura convertida a gramos

bool bascula1\_conectada; por defecto conectada

bool tara1\_cambio; cuando cambias de tara (rollo de filamento distinto)

bool bascula1\_calibrada; EEPROM

//bascula 2

float calibration\_factor2; EEPROM factor m

int32\_t zero\_offset2; EEPROM desfase para peso 0 (b)

uint32\_t tara2\_actual\_g; peso del rollo de filamento

int32\_t tara2\_actual\_offset; desfase para tara + peso del rollo del filamento

volatile float PESO2; lectura convertida a gramos

bool bascula2\_conectada por defecto conectada

bool tara2\_cambio cuando cambias de tara (rollo de filamento distinto)

bool bascula2\_calibrada EEPROM

int16t peso\_tara1; EEPROM peso del rollo de filamento tipo 1

int16t peso\_tara2; EEPROM peso del rollo de filamento tipo 2

int16t peso\_tara3; EEPROM peso del rollo de filamento tipo 3

int16t peso\_tara4; EEPROM peso del rollo de filamento tipo 4

int16t peso\_tara5; EEPROM peso del rollo de filamento tipo 5

uint8\_t ultima\_tara1; EEPROM ultima configuración guardada para la báscula 1 (1-5)

uint8\_t ultima\_tara2; EEPROM ultima configuración guardada para la báscula 1 (1-5)

char name\_tara1[10]; EEPROM nombre del rollo de filamento tipo 1

char name\_tara2[10]; EEPROM nombre del rollo de filamento tipo 2

char name\_tara3[10]; EEPROM nombre del rollo de filamento tipo 3

char name\_tara4[10]; EEPROM nombre del rollo de filamento tipo 4

char name\_tara5[10]; EEPROM nombre del rollo de filamento tipo 5

### Variables Nextion

### Proceso

Ecuación lineal de peso: y=mx+b.

x=lectura del HX711, y=peso conocido, m=pendiente o factor de calibración, b es la interseccion cuando y=0 (tara).

Para obtener la ecuación de la recta necesitamos 2 lecturas con 2 pesos conocidos, uno será en modo tara y el otro con un peso conocido.

Despejando:

y= (y1/x1-x0)(x-x0)

### funciones

leerBascula1()

1. Comprobar que el sensor esta conectado
2. Si está conectado, bascula1\_conectada=true
3. Leer PESO1
4. Si no está conectado, notificación en debug y Nextion sin alarma.
5. Debug

leerBascula2() idem como báscula1

calibrar\_factor1()

1. Comprobar que el sensor esta conectado. Continua o error
2. Notificación Nextion y espera: Quitar peso de la báscula
3. Tare, zero\_factor1 = lectura,Tare
4. + / - aumentar disminuir el factor en Nextion, al mismo tiempo actualizar la lectura del peso y enviar a Nexion
5. OK, enviar a arduino

calibrar\_factor2() Idem como factor1

calibrar\_peso1()

1. Comprobar que el sensor esta conectado. Continua o error
2. Notificación Nextion y espera: Quitar peso de la báscula
3. Tare. Notificación Nextion para introducir el peso calibrado que pondremos en la báscula
4. Notificación. Pon el peso de x g en la báscula
5. Callibrate\_scale

calibrar\_peso2() Idem como peso1

## Temperatura y humedad

### Variables Arduino

//sensor exterior

float dht\_T\_ext\_correccion; EEPROM corrección que se SUMA a la lectura de temp. exterior

float dht\_H\_ext\_correccion; EEPROM corrección que se SUMA a la lectura de humedad exterior

float T\_ext; Lectura de la temperatura exterior

float H\_ext; Lectura de la humedad exterior

//sensor interior

float dht\_T\_int\_correccion; EEPROM corrección que se SUMA a la lectura de temp. interior

float dht\_H\_int\_correccion; EEPROM corrección que se SUMA a la lectura de humedad interior

float T\_int; Lectura de la temperatura interior

float H\_int; Lectura de la humedad interior

float start\_temp; Lectura de la temperature interior al iniciar el programa (para el progeso)

float target\_temp; Temperatura interior final configurada

float current\_temp; Temperatura interior actual

uint32\_t tempTime;

uint32\_t tempTimeInterval = 4000; Tiempo entre lecturas del sensor de temperatura interior

//temperaturas programadas

uint8\_t temp\_prog1; EEPROM Programa de temperatura 1

uint8\_t temp\_prog2; EEPROM Programa de temperatura 2

uint8\_t temp\_prog3; EEPROM Programa de temperatura 3

uint8\_t temp\_prog4; EEPROM Programa de temperatura 4

uint8\_t temp\_prog5; EEPROM Programa de temperatura 5

char name\_prog1[6]; EEPROM Nombre del programa 1 (tiempo,temperatura)

char name\_prog2[6]; EEPROM Nombre del programa 1 (tiempo,temperatura)

char name\_prog3[6]; EEPROM Nombre del programa 1 (tiempo,temperatura)

char name\_prog4[6]; EEPROM Nombre del programa 1 (tiempo,temperatura)

char name\_prog5[6]; EEPROM Nombre del programa 1 (tiempo,temperatura)

### Variables Nextion

### Proceso

leer\_temp\_int()

1. Lee temperatura y humedad y se suma la correción (calibración) en intervalos marcados por la variable tempTimeInterval
2. Si algún sensor de temperatura no es detectado:
   1. ERROR – Debug y Nextion
   2. apagar resistencia
   3. alarma de error critico, también si el beep no esta activado
3. Debug

leer\_temp\_ext()

1. Lee temperatura y humedad y se suma la correción (calibración) en intervalos marcados por la variable tempTimeInterval
2. Si algún sensor de temperatura no es detectado:
   1. ERROR – Debug y Nextion
   2. apagar resistencia
   3. alarma de error critico, también si el beep no esta activado
3. Debug

## Temporizador

### Variables Arduino

bool timer; EEPROM Activa y desactiva el temporizador

bool timer\_beep; EEPROM Activa y desactiva la señal acustica de fin de programa

uint16\_t set\_time\_min; EEPROM Temporizador configurado en minutos

uint32\_t now; Tiempo actual

uint32\_t startTime; Tiempo de inicio del temporizador

uint32\_t totalTime; Tiempo final del programa

bool temporizador\_end; Verdadero si el programa/temporizador ha terminado

bool empezar\_programa; Empieza el temporizador, iniciar variables.

//temporizadores programados

uint8\_t time\_prog1; EEPROM Tiempo programado 1

uint8\_t time\_prog2; EEPROM Tiempo programado 2

uint8\_t time\_prog3; EEPROM Tiempo programado 3

uint8\_t time\_prog4; EEPROM Tiempo programado 4

uint8\_t time\_prog5; EEPROM Tiempo programado 5

### Variables Nextion

### Proceso

bool end\_time() Comprueba si es el final del programa/temporizador

## EEPROM

### Variables Arduino

typedef struct{

bool SERIAL\_DEBUG;

uint8\_t temp\_prog1;

uint8\_t temp\_prog2;

uint8\_t temp\_prog3;

uint8\_t temp\_prog4;

uint8\_t temp\_prog5;

bool timer;

uint16\_t set\_time\_min;

bool timer\_beep;

bool vent\_auto\_off;

uint8\_t VentMinDutyCycle;

float calibration\_factor1;

int32\_t zero\_factor1;

float calibration\_factor2;

int32\_t zero\_factor2;

float vaggKp;

float vaggKi;

float vaggKd;

float aggKp;

float aggKi;

float aggKd;

float consKp;

float consKi;

float consKd;

uint16\_t WindowSize;

float dht\_T\_ext\_correccion;

float dht\_T\_int\_correccion;

float dht\_H\_ext\_correccion;

float dht\_H\_int\_correccion;

uint16\_t time\_prog1;

uint16\_t time\_prog2;

uint16\_t time\_prog3;

uint16\_t time\_prog4;

uint16\_t time\_prog5;

uint32\_t WeightReadingRefreshTime;

float setpoint;

int16\_t neopixel\_type;

uint8\_t neopixel\_pixels;

char ver[sizeof(SETTING\_VER)];

} ParametersType;

### Variables Nextion

### Proceso

Cualquier cambio en se señala con la flag userSettings de la EEPROM.

Si la flag es true, leer setUser, si es false leer setDefaults

userSettings = setUser.parameters->userSettings;

unsigned long eeprom\_crc()

settings.save();

## RTC – Arduino

### Variables Arduino

uint8\_t hora\_actual;

uint8\_t minuto\_actual;

DateTime ahora;

### Variables Nextion

### Proceso

## Neopixel

### Variables Arduino

#define NEOPIXEL\_LED comentar si no hay tira LED cam

#define NEOPIXEL\_IS\_SEQUENTIAL

#define NEOPIXEL\_BRIGHTNESS 127 – convertir a variable

#define NEOPIXEL\_STARTUP\_TEST

#define NEOPIXEL\_BKGD\_LED\_INDEX

#define NEOPIXEL\_BKGD\_COLOR

int16\_t neopixel\_type; EEPROM Tipo de tira LED. Selección en Nextion

uint8\_t NEOPIXEL\_PIXELS; EEPROM Número de pixeles de la tira led

bool ledStatus; EEPROM LED encendido o apagado

bool ledFlip; ¿?

bool LEDHeatingDONE;

uint32\_t ledTime;

uint32\_t ledTimeInterval = 300; //Tiempo de cambio de los LED ms

uint32\_t colorOff = neo->Color( 0, 0, 0);

uint32\_t colorRed = neo->Color(255, 0, 0);

uint32\_t colorOrange = neo->Color(255, 80, 0);

uint32\_t colorYellow = neo->Color(255, 255, 0);

uint32\_t colorGreen = neo->Color( 0, 255, 0);

uint32\_t colorBlue = neo->Color( 0, 0, 255);

uint32\_t colorIndigo = neo->Color( 0, 255, 255);

uint32\_t colorViolet = neo->Color(255, 0, 255);

uint32\_t colorWhite = neo->Color(255, 255, 255);

uint16\_t hue = 65536/3; //Color Verde

### Variables Nextion

### Proceso

## Serial - debug

### Variables Arduino

bool SERIAL\_DEBUG; EEPROM Activa y desactiva el debug a traves del serial

uint32\_t serialTime;

char received; Buffer para la función SerialReceive

### Variables Nextion

### Proceso

void SerialSend() envia valores de setpoint/input/output al serial

char SerialReceive() Buffer – recepción del serial

settings.printSerial(); muestra toda los datos guardados de configuración en el serial

## ALARMA - BUZZER

### Variables Arduino

### Variables Nextion

### Proceso

beep(int HZ, int on, int off)

alarma(int modo)

MODO 0 = BIENVENIDA - ENCENDIDO

MODO 1 = ERROR CRITICO SENSOR DE TEMPERATURA

MODO 2 = PROGRAMA TERMINADO

MODO 3 = RTC no conectado o sin batería

## Conexión Nextion

### Variables Arduino

uint32\_t DATA\_REFRESH\_RATE Frecuencia de refresco de páginas de Nextion

uint32\_t pageRefreshTimer = millis() Timer for DATA\_REFRESH\_RATE

bool newPageLoaded true when the page is first loaded ( lastCurrentPageId != currentPageId )

uint8\_t nxPagina; Página actual en Nextion

### Variables Nextion

### Proceso

writeString(String stringData) Usado para mandar por el Serial un String con Serial.write()

RefreshCurrentPage() Se refresca la página

Las funciones de refresco de página se hayan en los archivos nxFunctions.h y .cpp

refreshPage0() página: main

refreshPage1() página: hora

refreshPage2() página: setTiempo

refreshPage3() página: setTemp

refreshPage4() página: configA

refreshPage5() página: configB

refreshPage6() página: configC

refreshPage7() página: configD

refreshPage8() página: configE

refreshPage9() página: configF

refreshPage10() página: Teclado

refreshPage11() página: numpad

refreshPage12() página: peso1Met1

refreshPage13() página: peso1Met2

refreshPage14() página: configDHTtemp

refreshPage15() página: configDHThum

refreshPage16() página: peso2Met1

refreshPage17() página: peso2Met2

refreshPage18() página: graficaTemp

refreshPage19() página: graficaHum

refreshPage20() página: graficaPeso1

refreshPage21() página: graficaPeso2