/\* DEFINE \*/

# define START\_STATE 0

# define COM\_CHECK\_STATE 1

# define COM\_SERIAL\_STATE 2

# define COM\_MANUAL\_STATE 3

# define REGULATION\_STATE 4

# define UPDATE\_VARIABLES\_STATE 5

# define COMMAND\_STATE\_PIN 8 // pin connected to the button that select the command mode

# define PWM\_PIN 7

# define LCD\_MENU\_PIN 6

# define ON\_OFF\_PIN 9 // button to switch on or off the regulation

// Pin ou étape ??!!?!

# define CHECK\_NIVEAU\_STATE 10 // Pin de signal du niveau bas atteint

//# define CHECK\_TEMP\_STATE 11 // Pin de signal de la température atteinte

# define NIVEAU\_BAS 10 // Pin de signal du niveau bas atteint

# define TEMP\_ATTEINT 13 // Pin de signal de la température atteinte

/\* LIBRARIES \*/

#include <PID\_v1.h>

#include <LiquidCrystal.h>

#include <math.h>

/\* GLOBAL VARIABLES TABLE \*/

volatile uint8\_t duty\_cycle =0; // duty\_cycle en sortie de régulateur

volatile uint16\_t pwm\_time =0;

volatile bool pwm\_state =0;

word time\_elapsed\_start =0;

double setpoint =0;

double previous\_setpoint =1000;

double input ;

double output ;

unsigned int setpoint\_time =0;

unsigned long time\_elapsed =0; // temps écoulé depuis atteinte consigne

bool setpoint\_reached =0; // flag to know if setpoint has been reached

byte k\_parameter =0;

byte d\_parameter =0;

byte i\_parameter =0;

float temper =0;

float previous\_temper =1000;

byte previous\_menu =3;

bool onoff\_flag ;

bool setpoint\_flag ;

// Screen

bool selectionMenu = false ;

bool comparaison ;

float val;

int sensorValue;

/\* OBJECTS \*/

PID myPID(&input, &output, &setpoint,0.5,1,0, DIRECT);

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

void setup()

{

 // SERIAL initialiszation

 Serial.begin(115200);

 lcd.begin(16, 2);

 // TIMER1 initialization for PWM generation

 cli(); //stop interrupts

 TCCR1A = 0; // set entire TCCR1A register to 0

 TCCR1B = 0; // same for TCCR1B

 TCNT1 = 0; //initialize counter value to 0

 OCR1A = 781; // = (16\*10^6) / (1\*1024) - 1 (must be <65536)

 TCCR1B |= (1 << WGM12); // turn on CTC mode

 TCCR1B |= (1 << CS12) | (1 << CS10); // Set CS10 and CS12 bits for 1024 prescaler

 TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt : déclenche l'interruption

 sei(); //allow interrupts

 // PID parameters

 myPID.SetOutputLimits(0,100);

 myPID.SetMode(AUTOMATIC);

 // Pin configuration

 pinMode(COMMAND\_STATE\_PIN,INPUT);

 pinMode(LCD\_MENU\_PIN,INPUT);

 pinMode(PWM\_PIN,OUTPUT);

 pinMode(NIVEAU\_BAS,OUTPUT);

 pinMode(TEMP\_ATTEINT,OUTPUT);

 // LCD screen

 lcd.setCursor(0, 0);

 lcd.print(" ");

 lcd.setCursor(0, 1);

 lcd.print(" ");

}

void loop()

{

/\* VARIABLES \*/

 static byte state=0; // variable used for switch function

 switch (state)

 {

 case START\_STATE:

 // First state when booting the Arduino

 state = COM\_CHECK\_STATE;

 //Serial.println("start");

 break;

 case COM\_CHECK\_STATE:

 // Check in which command state we are : Manual (from the Arduino) or Automatic (from Raspberry)

 //Serial.println("check");

 if (digitalRead(COMMAND\_STATE\_PIN)==1)

 state = COM\_MANUAL\_STATE;

 else // if in automatic state, only allow to display information on small screen

 {

 if( digitalRead(LCD\_MENU\_PIN) == false )

 {

 if (temper != previous\_temper || previous\_menu != 1)

 {

 lcd.setCursor(0, 0);

 lcd.print(" ");

 lcd.setCursor(0, 1);

 lcd.print(" ");

 lcd.setCursor(0, 0);

 lcd.print("Temp actuelle");

 lcd.setCursor(0, 1);

 lcd.print("T :");

 lcd.setCursor(4,1);

 lcd.print(temper);

 previous\_temper = temper;

 previous\_menu= 1;

 }

 }

 else

 {

 if (setpoint != previous\_setpoint || previous\_menu != 0)

 {

 lcd.setCursor(0, 0);

 lcd.print(" ");

 lcd.setCursor(0, 1);

 lcd.print(" ");

 lcd.setCursor(0, 0);

 lcd.print("Consigne");

 lcd.setCursor(0, 1);

 lcd.print("T :");

 lcd.setCursor(4,1);

 lcd.print(setpoint);

 previous\_setpoint = setpoint;

 previous\_menu = 0;

 }

 }

 state = COM\_SERIAL\_STATE;

 }

 break;

 case COM\_SERIAL\_STATE:

 // State where the serial comunication is managed: sending data from the variables table or writing data in the variables table

 //Serial.println("serial");

 Serial\_communication();

 state = REGULATION\_STATE;

 break;

 case COM\_MANUAL\_STATE:

 // State where the manual communication is managed: modifying regulation parameters

 state = REGULATION\_STATE;

 if( digitalRead(LCD\_MENU\_PIN) == false )

 {

 if (temper != previous\_temper || previous\_menu != 1)

 {

 lcd.setCursor(0, 0);

 lcd.print(" ");

 lcd.setCursor(0, 1);

 lcd.print(" ");

 lcd.setCursor(0, 0);

 lcd.print("Temp actuelle");

 lcd.setCursor(0, 1);

 lcd.print("T :");

 lcd.setCursor(4,1);

 lcd.print(temper);

 previous\_temper = temper;

 previous\_menu= 1;

 }

 }

 else

 {

 sensorValue = analogRead(A1); // Mesure du potentiomètre, l'arduino mesure une tension comprise entre 0V et 5V et la transpose en un int compris entre 0 et 1023

 val = sensorValue ;

 val = val/10,23; // Valeur comprise entre 0 et 1023 , on divise par 10,23 pour avoir une gamme de t° comprise entre 0°C et 100°C

 val = round(val); // Il n'est pas nécessaire d'avoir une température trop précise sur l'afficheur. On l'arrondit à l'unité

 setpoint = val ; // sensorValue est un int , permet de ne pas afficher les virgules sur l'écran LCD

 if (setpoint != previous\_setpoint || previous\_menu != 0)

 {

 lcd.setCursor(0, 0);

 lcd.print(" ");

 lcd.setCursor(0, 1);

 lcd.print(" ");

 lcd.setCursor(0, 0);

 lcd.print("Consigne");

 lcd.setCursor(0, 1);

 lcd.print("T :");

 lcd.setCursor(4,1);

 lcd.print(setpoint); // affichage de la température de consigne

 previous\_setpoint = setpoint;

 previous\_menu = 0;

 }

// lcd.setCursor(0, 0);

// lcd.print("Consigne");

// lcd.setCursor(0, 1);

// lcd.print("T :");

// lcd.setCursor(4,1);

// sensorValue = analogRead(A1); // Mesure du potentiomètre, l'arduino mesure une tension comprise entre 0V et 5V et la transpose en un int compris entre 0 et 1023

// val = sensorValue ;

// val = val/10,23; // Valeur comprise entre 0 et 1023 , on divise par 10,23 pour avoir une gamme de t° comprise entre 0°C et 100°C

// val = round(val); // Il n'est pas nécessaire d'avoir une température trop précise sur l'afficheur. On l'arrondit à l'unité

// sensorValue = val ; // sensorValue est un int , permet de ne pas afficher les virgules sur l'écran LCD

// lcd.print(sensorValue); // affichage de la température de consigne

 }

 break;

 case REGULATION\_STATE:

 // State where the regulation process occurs

 input = Calcul\_temperature();

 myPID.Compute();

// Serial.print ("output: ");

// Serial.println (output);

// Serial.print("temp: ");

// Serial.println(temper);

// Serial.print ("Setpoint: ");

// Serial.println(setpoint);

 if (digitalRead(ON\_OFF\_PIN)==1)

 duty\_cycle=(uint8\_t)output;

 else

 duty\_cycle=0;

 if ((((time\_elapsed/1000)>setpoint\_time)&&setpoint\_flag==1)/\*||onoff\_flag==0\*/)

 //setpoint=0;

 setpoint\_flag=0;

 state = UPDATE\_VARIABLES\_STATE;

 break;

 case UPDATE\_VARIABLES\_STATE:

 // State where the variables values are updated

 if (setpoint\_flag==0 && temper>0,95\*setpoint)

 {

 setpoint\_flag =1;

 time\_elapsed\_start=0;

 }

 if (setpoint\_flag==1)

 {

 time\_elapsed = millis()-time\_elapsed\_start;

 }

 state = CHECK\_NIVEAU\_STATE;

 break;

 case CHECK\_NIVEAU\_STATE:

 float Sharp\_analog = 0;

 Sharp\_analog = (float)analogRead(A2);

 if (Sharp\_analog <= 3.2)

 digitalWrite(NIVEAU\_BAS, HIGH);

 else

 digitalWrite(NIVEAU\_BAS, LOW);

 state = COM\_CHECK\_STATE;

 break;

// case CHECK\_TEMP\_STATE:

// if (setpoint\*0.95 <= temper)

// digitalWrite(TEMP\_ATTEINT, HIGH);

// else

// digitalWrite(TEMP\_ATTEINT, LOW);

// state = COM\_CHECK\_STATE;

// break;

 }

 delay(1); // delay between each switch loop

}

void Serial\_communication()

{

 byte rxdata[4],txdata[4],instruction\_nbr,valueH,valueL; // arrays of data for serial communication

 float t\_value;

 int temp\_int = 0;

 bool Flag\_Niveau\_Bas;

 bool Flag\_Temp\_Atteint;

 byte \* byte\_pointer =(byte \*) &temp\_int;

 byte txbuffer[10];

 float temp\_float=0;

 instruction\_nbr=Readdata(rxdata) ;

 switch(instruction\_nbr)

 {

 case(0): // No communication

 //Serial.print("error");

 break;

 case(1): // New setpoint/time received

 setpoint = Byte\_to\_int(rxdata[0],rxdata[1]); // New setpoint

 setpoint\_time = Byte\_to\_int(rxdata[3],rxdata[4]); // New time

 Clear\_array(rxdata,4);

 //Serial.print(setpoint);

 break;

 case(2): // New PID parameters

 k\_parameter = rxdata[0];

 d\_parameter = rxdata[1];

 i\_parameter = rxdata[2];

 if (rxdata[3]==0)

 onoff\_flag = 0;

 if (rxdata[3]==1)

 onoff\_flag = 1;

 Clear\_array(rxdata,4);

 //Serial.print(k\_parameter);

 break;

 case(3): // current T° , time ellapsed(s)

 txbuffer[0]=255;

 txbuffer[1]=255;

 txbuffer[2]=3;

 temp\_float=temper\*100;

 temp\_int = (int)temp\_float;

 txbuffer[3]=byte\_pointer[1];

 txbuffer[4]=byte\_pointer[0];

 temp\_float=time\_elapsed/1000;

 temp\_int = (int)temp\_float;

 txbuffer[5]=byte\_pointer[1];

 txbuffer[6]=byte\_pointer[0];

 delay(50);

 Serial.write(txbuffer,7);

 Serial.flush();

 Clear\_array(rxdata,4);

 break;

 case(4): // Low level and temp atteinte

 txbuffer[0]=255;

 txbuffer[1]=255;

 txbuffer[2]=4;

 //Test Valeur à remplacer

 Flag\_Niveau\_Bas = CHECK\_NIVEAU();

 if (Flag\_Niveau\_Bas==1)

 txbuffer[3]=1;

 else

 txbuffer[3]=0;

 txbuffer[4]=1;

 txbuffer[5]=0;

 txbuffer[6]=0;

 delay(50);

 Serial.write(txbuffer,7);

 Serial.flush();

 Clear\_array(rxdata,4);

 break;

 default:

 // if nothing else matches, do the default

 break;

 }

}

float Calcul\_temperature()

{

 float pt100\_analog = 0;

 float pt100\_r =0;

 pt100\_analog = (float)analogRead(A0);

 pt100\_r = (950/(5-(pt100\_analog\*5/(1024\*7.67))))\*(pt100\_analog\*5/(1024\*7.67)) ; // divise by power amplification ratio

 temper = (pt100\_r/(0.385))-(1/(0.00385));

 return temper;

}

float CHECK\_NIVEAU()

{

 float Sharp\_analog = 0;

 bool Flag\_Niveau\_Bas;

 Sharp\_analog = (float)analogRead(A2);

 if (Sharp\_analog <= 3.2)

 {digitalWrite(NIVEAU\_BAS, HIGH);

 Flag\_Niveau\_Bas = 1;}

 else

 {digitalWrite(NIVEAU\_BAS, LOW);

 Flag\_Niveau\_Bas = 0;}

 return Flag\_Niveau\_Bas;

}

unsigned int Byte\_to\_int(byte valueL, byte valueH)

{

 unsigned int value = 0;

 value=valueH;

 value=value<<8;

 value=(value|valueL);

 return value;

}

void Clear\_array(byte array[],byte array\_length)

{

 for (int i=0;i<array\_length;i++)

 array[i]=0;

}

//timer1 interrupt , CTC mode

ISR(TIMER1\_COMPA\_vect)

{

pwm\_time=pwm\_time+50; // keep track of time for PWM generation

if (!pwm\_state&&(pwm\_time>(100-duty\_cycle)\*50))

 {

 //passer la pin à l'état haut

 digitalWrite(PWM\_PIN,HIGH);

 pwm\_time=0;

 pwm\_state=1;

 }

else if(pwm\_state&&(pwm\_time>(duty\_cycle\*50)))

 {

 // passer la pin à l'état bas

 digitalWrite(PWM\_PIN,LOW);

 pwm\_time=0;

 pwm\_state=0;

 }

}

// Function to process data coming from Serial port

byte Readdata(byte rxdata[]) // return instruction nbr to process : 0 is nothing to process

{

 //Serial.print("r");

 byte incomingbyte,instruction\_nbr,itt,i;

 while (Serial.available()>0)

 {

 incomingbyte=Serial.read();

 if (Serial.available()==0&&incomingbyte==255) // to be sure that we don't miss a frame if only the first 255 byte has arrived

 delay(10);

 if(incomingbyte==255 && Serial.peek()==255)

 {

 incomingbyte=Serial.read(); // second 0xFF

 while (Serial.available()<5) // while frame not completely received : wait (max 200 ms)

 {

 delayMicroseconds(10);

 itt++;

 if (itt>19)

 {

 EmptyRxSerial();

 return 254;

 }

 }

 instruction\_nbr =Serial.read(); //Instruction Id

 for (i=0;i<4;i++)

 rxdata[i]=Serial.read();

 return instruction\_nbr;

 }

 }

 instruction\_nbr=0;

 return instruction\_nbr;

}

// Function to send data on the Serial port

//void Senddata (byte instruction\_nbr ,byte txdata[])

//{

// byte txbuffer[30];

// txbuffer[0]=255;

// txbuffer[1]=255;

// txbuffer[2]=\*instruction\_nbr;

// //\*instruction\_nbr=0;

// for (byte i =0;i<4;i++)

// {

// txbuffer[i+3]=txdata[i];

// txdata[i]=0;

// }

// Serial.write(txbuffer,7);

//}

void EmptyRxSerial()

{//Empty serial channel

 byte empty;

 while(Serial.available()>0) //empty the rx buffer

 {

 empty=Serial.read();

 }

}