**Projet Arduino chauffage**

Description

L’installation permet de chauffer une résistance de 100Ω pendant une durée déterminée.

Le but est de réguler la température de façon proportionnelle (PID).

Le programme doit fonctionner selon le concept du graph d’état.

Informations complémentaires

Tout le code ainsi que des informations supplémentaires du projet sont disponible sur GitHub : <https://github.com/CrBast/distillation_column>

Composants

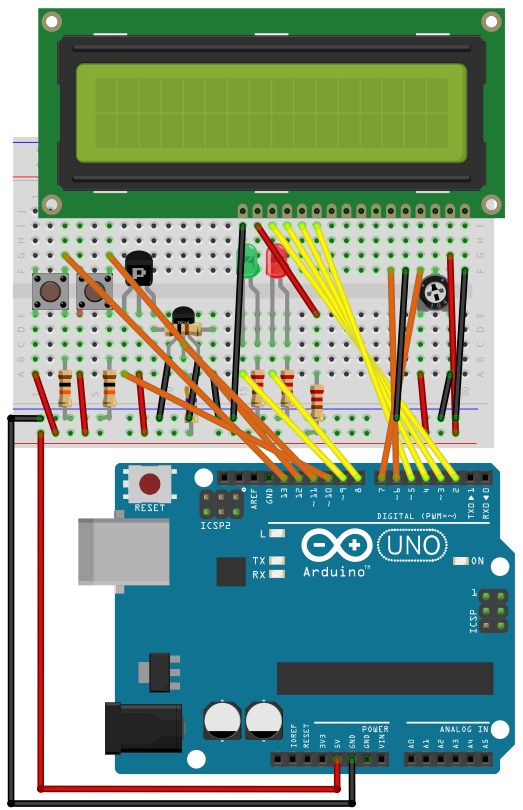
* Ecran LCD
* Capteur de température (Dallas – DS18B20)
* Transistor (BD675)
* Résistance de 100 Ω

Dépendances

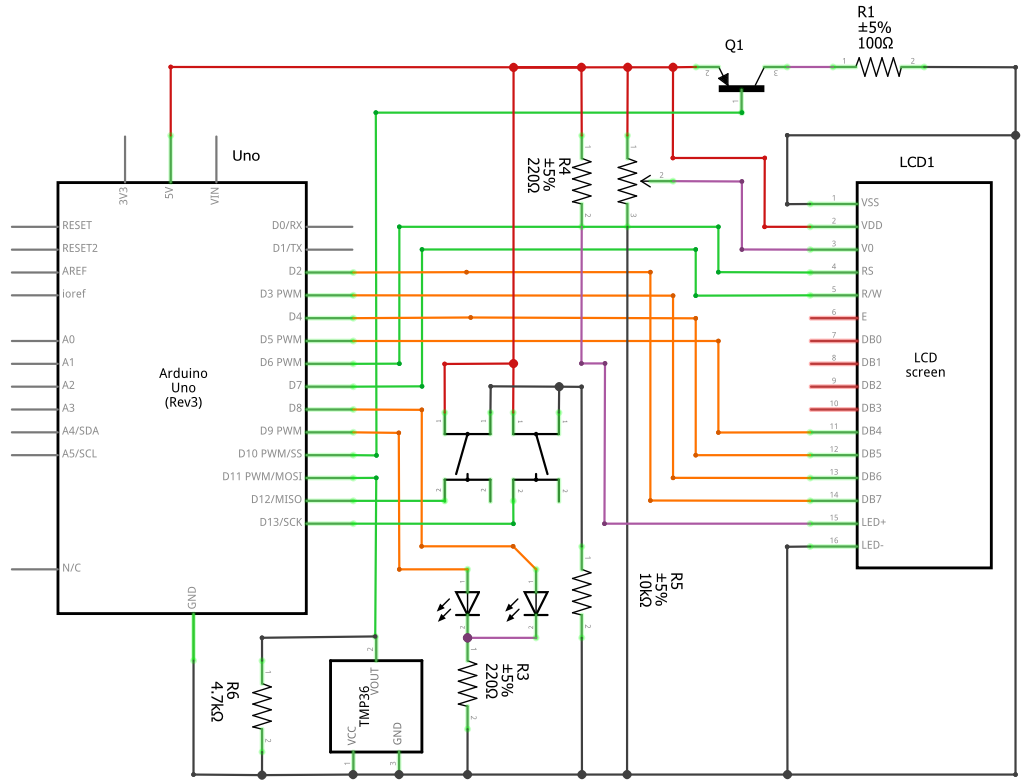
Voici les librairies open source utilisées :

* LiquidCrystal
* OneWire
* DallasTemperature

**Partie Arduino**



Vue schématique



**DS18B20**

Vue plaquette d’expérimentation

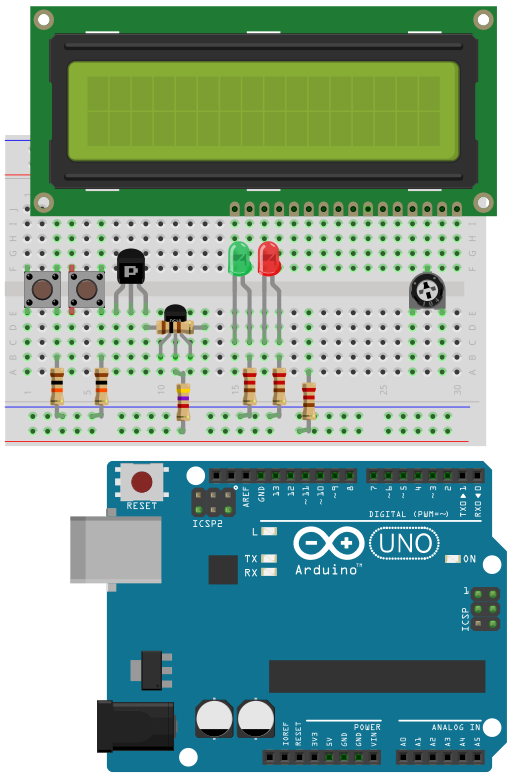
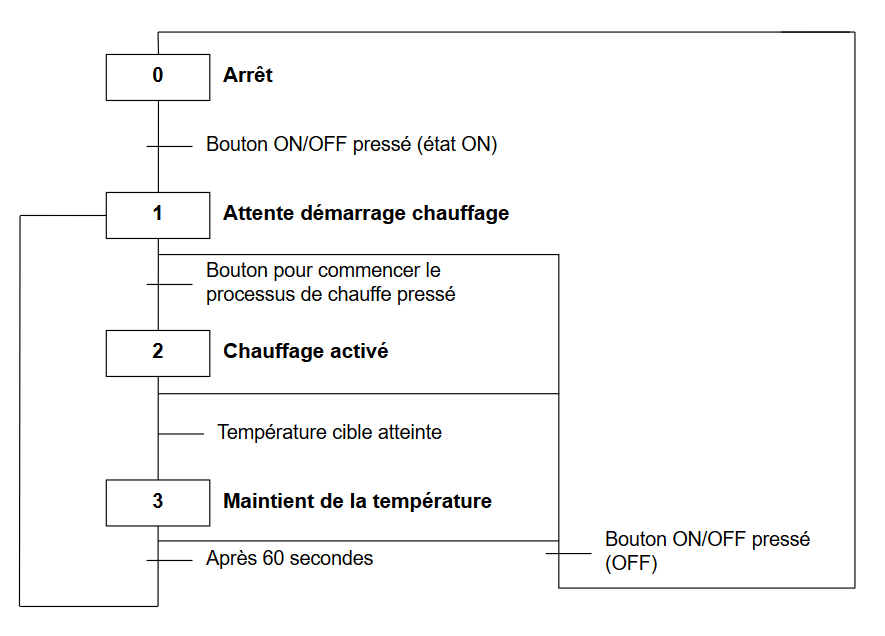
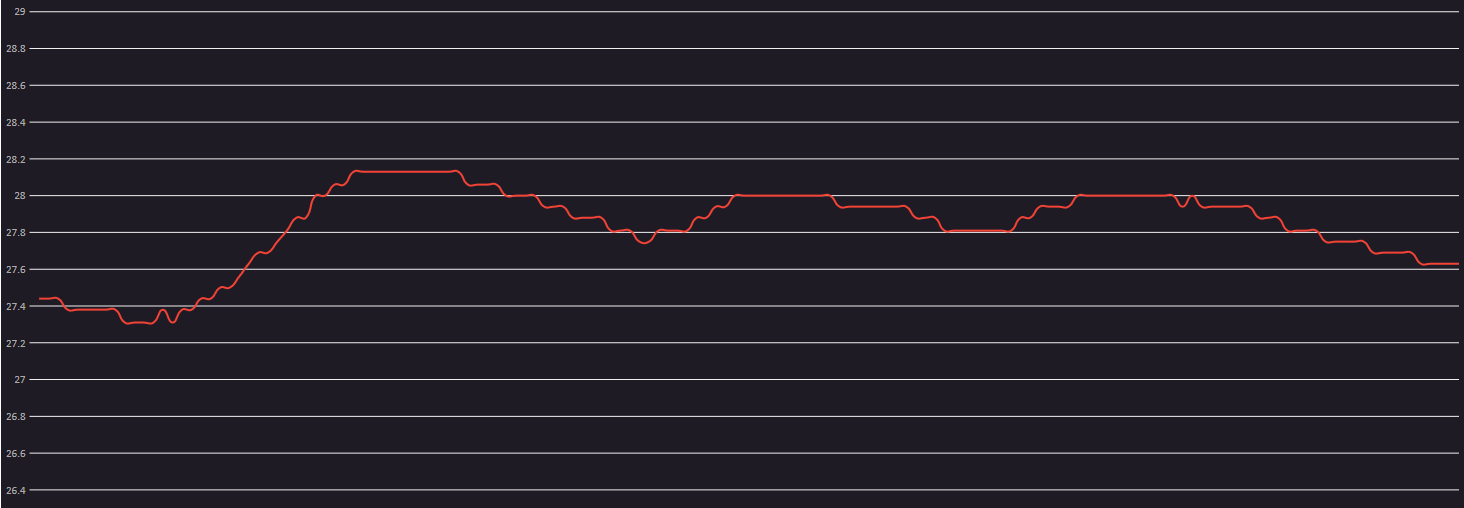


Diagramme états-transitions



Graphique test température (PID)



Etat 1

Etat 3

Etat 2

Etat 1

Code

Github : <https://github.com/CrBast/distillation_column/blob/master/arduino/arduino.ino>

/\*\*

Bastien Crettenand

Projet pratique - Module 121

Source : https://github.com/CrBast/distillation\_column/

Librairies :

- LiquidCrystal

- OneWire

- DallasTemperature

\*/

#include <LiquidCrystal.h>

#include <OneWire.h>

#include <DallasTemperature.h>

const int rs = 6, en = 7, d4 = 5, d5 = 4, d6 = 3, d7 = 2;

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

#define ONE\_WIRE\_BUS 11

OneWire oneWire(ONE\_WIRE\_BUS);

DallasTemperature sensors(&oneWire);

// Variables générales

int state = 0;

int ledOnOff = 9;

int ledBusy = 8;

int btnOnOff = 13;

bool btnOnOff\_lastState = false;

int btnStartWork = 12;

bool btnStartWork\_lastState = true;

// Variables nécessaire pour la régulation proportionnelle

int activity = 0;

int loop\_occurence = 1000;

int loop\_numberOccurence = 1000;

int loop\_onSameTemp = 0;

int loop\_startingProportion;

double loop\_lastTemp;

// Spécifique température

double todo\_temp = 28;

double ambiant\_temp = 25;

double actual\_temp;

int trans = 10;

long testInt = 0;

void setup(void)

{

lcd.begin(16, 2);

Serial.begin(9600);

pinMode(trans, OUTPUT);

pinMode(ledBusy, OUTPUT);

pinMode(ledOnOff, OUTPUT);

pinMode(btnOnOff, INPUT);

pinMode(btnStartWork, INPUT);

digitalWrite(trans, LOW);

digitalWrite(ledBusy, LOW);

digitalWrite(ledOnOff, LOW);

sensors.begin();

sensors.requestTemperatures(); // Send the command to get temperature readings

ambiant\_temp = 25; //sensors.getTempCByIndex(0);

Serial.println("-s " + (String)state);

bool btnOnOff\_lastState = false;

}

void loop(void)

{

switch (state)

{

case 0:

if (digitalRead(btnOnOff) == HIGH && btnOnOff\_lastState != true)

{

btnOnOff\_lastState = true;

state = 1;

Serial.println("-s " + (String)state);

digitalWrite(ledOnOff, HIGH);

}

if(digitalRead(btnOnOff) == LOW){

btnOnOff\_lastState = false;

}

break;

case 1:

sensors.requestTemperatures(); // Send the command to get temperature readings

actual\_temp = sensors.getTempCByIndex(0);

Serial.println("-i " + (String)actual\_temp);

ldcSetTextByLine(0, "\t" + (String)actual\_temp + "\tC");

ldcSetTextByLine(1, "\t" + (String)state);

if (digitalRead(btnStartWork) == HIGH)

{

state = 2;

Serial.println("-s " + (String)state);

}

break;

case 2:

if(proportional\_control()){ // Wait the sensor go to <todo\_temp>

state = 3;

Serial.println("-s " + (String)state);

}

break;

case 3:

digitalWrite(ledBusy, HIGH);

if(testInt <= 76433){ // 76433 ~= 60 seconds

testInt++;

proportional\_control();

} else {

digitalWrite(trans, LOW);

digitalWrite(ledBusy, LOW);

state = 1;

Serial.println("-s " + (String)state);

Serial.println("-p " + (String)activity);

testInt = 0;

}

break;

}

// Evènement lors du clique bouton OFF

if (digitalRead(btnOnOff) == HIGH && btnOnOff\_lastState == false)

{

//Remise état "éteint"

state = 0;

Serial.println("-s " + (String)state);

digitalWrite(ledOnOff, LOW);

digitalWrite(ledBusy, LOW);

btnOnOff\_lastState = true;

ldcSetTextByLine(0, "");

ldcSetTextByLine(1, "");

digitalWrite(trans, LOW);

}

if(digitalRead(btnOnOff) == LOW){

btnOnOff\_lastState = false;

}

}

// V1.2 Propotional Control Algo

// One full loop = 0.785 seconds <-> 785 miliseconds

//

// Méthode de régulation proportionnelle - Non bloquante

bool proportional\_control()

{

double diff = todo\_temp - ambiant\_temp;

if (loop\_numberOccurence >= loop\_occurence)

{

if (ambiant\_temp <= 25)

{

loop\_startingProportion = (int)95 + ((25 - ambiant\_temp) \* 2);

}

else

{

loop\_startingProportion = (int)95 - ((ambiant\_temp - 25) \* 2);

}

loop\_numberOccurence = 0;

sensors.requestTemperatures(); // Send the command to get temperature readings

actual\_temp = sensors.getTempCByIndex(0);

/\*Serial.print(", ");

Serial.println(activity);\*/

ldcSetTextByLine(0, "\t" + (String)actual\_temp + "\tC");

ldcSetTextByLine(1, "\t" + (String)state + "\t" + (String)activity);

Serial.println("-i " + (String)actual\_temp);

Serial.println("-s " + (String)state);

Serial.println("-p " + (String)activity);

int diffBtw25AndActualTemp = 25 - ambiant\_temp;

if (diffBtw25AndActualTemp <= 0) {

diffBtw25AndActualTemp = 1;

}

if (actual\_temp >= todo\_temp)

{

if ((double)(actual\_temp - todo\_temp) >= (double)(diff \* 3 / 100))

{

activity = 0;

}

else

{

if (actual\_temp == todo\_temp)

{

activity = 100 \* diffBtw25AndActualTemp;

}

else

{

activity = 50 \* diffBtw25AndActualTemp;

}

}

}

else

{

double actual\_diff = todo\_temp - actual\_temp;

if (actual\_diff > ((100 - loop\_startingProportion) \* diff / 100))

{

activity = 1000;

}

else

{

int temp\_activity = (int)(actual\_diff \* 100 / ((100 - loop\_startingProportion) \* diff / 100));

if (actual\_temp == loop\_lastTemp && actual\_temp <= todo\_temp)

{

loop\_onSameTemp++;

if (loop\_onSameTemp > 1)

{

temp\_activity += 0.5 \* diffBtw25AndActualTemp;

}

if (loop\_onSameTemp > 2)

{

temp\_activity += 1 \* diffBtw25AndActualTemp;

}

}

else

{

loop\_onSameTemp = 0;

}

if (temp\_activity < 0)

{

activity = 0;

}

else

{

if (temp\_activity <= 35)

{

activity = (int)temp\_activity \* 10 - 100;

}

else

{

activity = (int)temp\_activity \* 10;

}

}

}

loop\_lastTemp = actual\_temp;

}

if (actual\_temp >= todo\_temp)

{

return true;

}

else

{

return false;

}

}

if (loop\_numberOccurence < activity)

{

digitalWrite(trans, HIGH);

}

else

{

digitalWrite(trans, LOW);

}

loop\_numberOccurence++;

return false;

}

void ldcSetTextByLine(int line, String content)

{

lcd.setCursor(0, line);

lcd.print(" ");

lcd.setCursor(0, line);

lcd.print(content);

}

**Application Windows**

Description

L’application permet de visualiser en temp-réel diverses informations concernant le montage Arduino.

Il est possible d’exporter les données de température en CSV. Celles-ci se trouvent dans le répertoire /data/ à la racine de l’application.

L’Application a été développée en WPF (C#) et est seulement disponible pour Windows(32/64bits).

Le code complet de l’application se trouve sur GitHub : <https://github.com/CrBast/distillation_column/tree/master/windows-manager>

Convention de transfert de données Arduino – PC : <https://github.com/CrBast/distillation_column/wiki>

Dépendances

Voici les librairies open source utilisées

* LiveCharts (<https://lvcharts.net>)

Interface graphique

Les informations disponibles en temps réel :

* Température du capteur
* Etat (Selon diagramme états-transitions)
* Puissance de chauffe
* Toutes les informations transmises par l’Arduino

