Contents

1 Introduction

This project is a first version of a heating regulation prototype. It is a first draft a more complex heating regulation device (that will come latter) made to start looking at the heat variations in a house. It allow to measure the temperature of heating pipes thanks to four PT1000 sensors and display the temperature on a screen. The temperatures are also save in a SD card periodically to follow heat variation during time. For this prototype we need:

- an arduino nano,
- a LCD screen,
- a rotary encoder (to select the temperature (sensor) displayed on the screen),
- a RTC,
- a SD card slot,
- a wadtchdog,
- 4 PT1000 sensors,
- a pump.

2 The modules

2.1 Arduino nano

For this project, we use an arduino nano, because it is compact card so we will have a reduce prototype size. Moreover, we need quite a lot GPIO in this project, an arduino ProMini won't enougth IO.

2.2 LCD screen

We use a LCD screen of 16 caracters displayed on 2 lines. This type of screen can be interfaced with an arduino using the LiquidCrystal.h library (see it's github page). We can use only the D4 to D7 pins of the screen and let the pins D0 to D3 unconnected.

2.2.1 Wiring

The wiring of the screen is shown on Figure 1, where we have display the top view of the scheme of the module. The pins remaining free will be connected to the ardiuno. The potentiometer is used to set the backlight of the screen.

2.2.2 Usage

We call the constructof with:

```
LiquidCrystal lcd(RS,E,D4,D5,D6,D7);
```

by replacing RS, E, ... with the corresponding pins of the arduino. The screen can be connected on digital pins or analog pins as well except for analog pin higher than A5 apparently.

The we initilise the screen in the setup function with:

```
lcd.begin(Nc,Nl);
```

where Nc is the number of caracters per line and Nl is the number of lines. In our case Nc=16 and Nl=2. We write on the screen with the print() function. Read the doc for further informations on the functions available in the library.

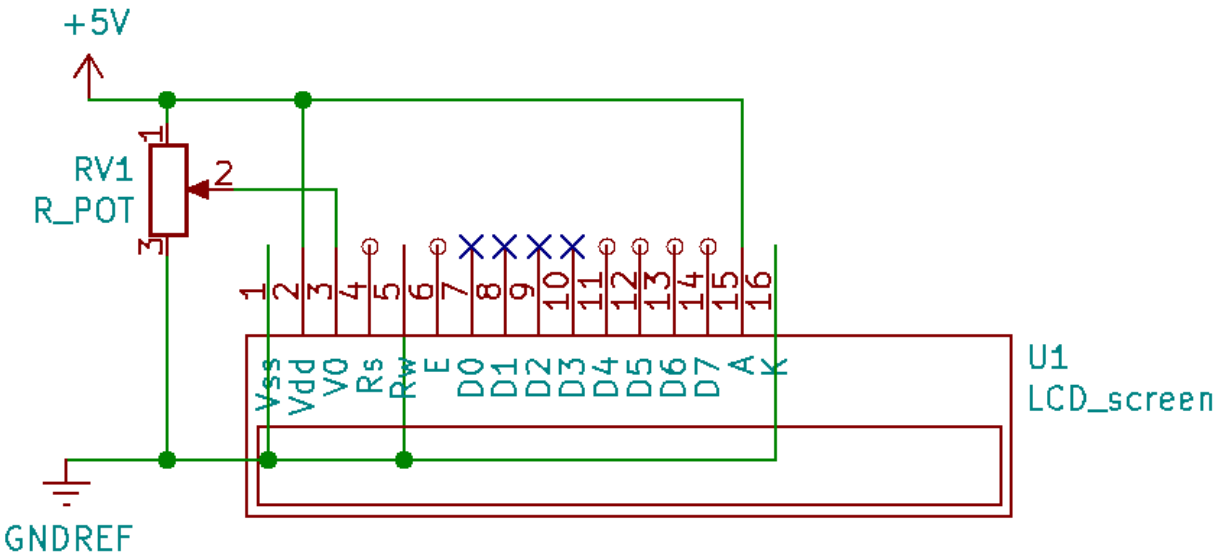


Figure 1: Wiring of the LCD screen.

2.3 Rotary encoder

We use a clickable rotary encoder, on can find an exemple on how to use this module here (in french). In the arduino code, we use the functions presented in this page as is to determine the rotation of the encoder.

2.3.1 Wiring

The wiring of this module is presented Figure 2. The SW pin is used to determine the click state of the encoder. The two other pins are used to determine the rotation of the encoder.

2.3.2 Remarques

Note that it is preferable to use interrupt for the pins DT and CLK so we have to plug them to pins 2 and 3 of the arduino.

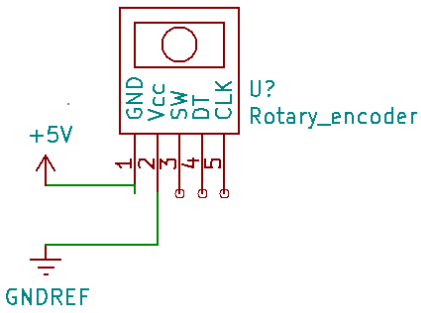


Figure 2: Rotary encoder wiring.

2.4 RTC

A RTC (acronym of Real Time Clock) is a module including a precise autonomous external clock (thanks to a battery). This kind of module allow to know the exact date and hour with a microcontroller even if it crash or if it is not powered continuously. We use a DS3231 RTC module which is an affordable and compact module, compatible 3.3 or 5V, based on DS3231 chip. This module use I2C bus, so it must be pluged to A4 (SDA) and A5 (SCL) pins of the arduino. To interface this module with the arduino, we use the RTClib.h library (it needs the Wire.h library).

2.4.1 Wiring

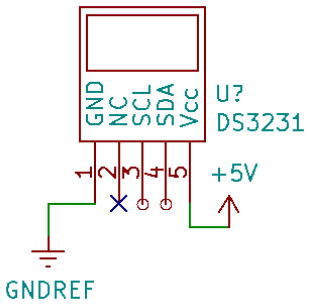


Figure 3: RTC wiring.

The wiring of the RTC is presented Figure 3, it is compatible 3.3 V and 5 V.

2.4.2 Usage

We include the library:

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

We call the constructor for the correct RTC (this library can manage different RTCs)

```
RTC_DS3231 rtc;
```

Then we initialise the RTC in the setup function

```
#ifdef AVR
  Wire.begin();
#else
  Wire1.begin();
#endif
  rtc.begin();
```

We collect a DateTime object with the now() function, this object will allow to display date and time.
We can adjust time and date of the RTC in the setup function using:

```
rtc.adjust(DateTime(__DATE__,__TIME__));
```

This function adjust the RTC to date and time of the compile timestamp. Beware, if we let this previous line in the code, the RTC will be adjust to compile time each time the microcontroller restart. So if it restart for one reason or another, the RTC will be disrupt. A good way to proceed consist of connect the RTC compile the code with the adjust function, so the RTC is adjusted ; then we power off the assembly and unplug the RTC, we comment the adjust line in the code and flash the arduino with the code will the RTC is still unpluged. No the RTC is correctly adjust and the arduino code won't disrupt it if it restart, so we can replug the RTC to the assembly.

2.5 SD card slot

We use a SD card slot module which communicate through SPI bus. We use the SD.h library (which use the SPI.h library) to communicate with this module. The dedicated arduino pin for SPI communication are pins 11 (MOSI), 12 (MISO) et 13 (CLK), the ChipSelect can be made with whatever arduino pin.

2.5.1 Wiring

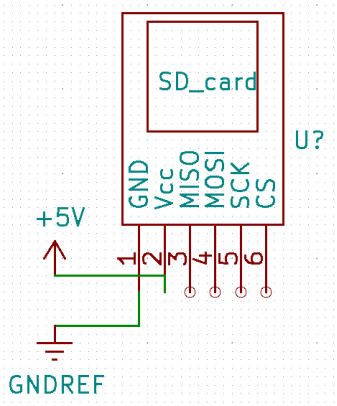


Figure 4: Wiring of SD card slot.

The wiring of the SD card slot module is presented on Figure 4, the module is display on top view. We don't show the connection with the arduino on this figure. The module is compatible 3.3 V and 5 V.

2.5.2 Usage

We include the library

```
#include <SPI.h>
#include <SD.h>
```

We initialise the object in the setup function (chipSelect is assigned to the CS pin of the module)

```
SD.begin(chipSelect);
```

We can open a myFile file on writing with

```
myFile = SD.open(file_name, FILE_WRITE);
```

The we can write into the file with

```
myFile.print();
```

Such as for writing on a serial link, we can use the println() function to insert a end of line caracter (\n) after what we have writen in the file. It is necesserary to call

```
myFile.close();
```

for writing into the SD card, otherwise it is store in a temporary buffer while this function is not called.

If the SD card is put out of the slot, we have to end the connection with the module with:

```
SD.end();
```

2.5.3 Remarques

The SPI communicatin is sensitive and is difficult to achieve by wiring using a breadboard. Our first test was made using a breadboard, and we were unable to communicate with the SD module. We had to connect directly the arduino and the SD module with female/female wires to make it work.

The SD.h library does not handle file name that have more than 12 caracters, 8 for name and 4 for the extension (.txt or .log for exemple).

The functions return does work correctly with the first version of the library we had tried. We need to update it to the last version (1.2.4 version) to obtain the expected behavior of the library (and determine if a SD card is plug using the open() function for example).

We can't use any SD card with this module. For example, the first SD card we used seemed to work correctly, we could save a certain amount of data is the card. But, relatively quickly, the SD.open() function returned null file that normally means that no SD card is plug in the module. Moreover, sometimes we didn't get null file from SD.open() function, but when we opened the SD card on a PC much more files were created thant expected and their name was writen with strange ASCII caracters. Or sometimes, the file names seemed coherent, but the strange ASCII apeares inside the files. After several tests, it appears that it was the SD card wich not work correctly with the SD card module.

2.6 Watchdog

A watchdog is a system that can restart a microcontroller when this one crash. We can make a small project without a watchdog if it run few minutes. But if it have to run for hours, days or more, it is essential to use a watchog. This prototype will have to runs for days and weeks and the future project will have to runs for months so we have to use a watchdog.

The arduino have an internal watchdog timer that we can use with the avr/wdt.h library. The watchdog time is initialised inside the setup function

```
wdt_enable(WDTO_8S)
```

here, the timer is adjust to 8 seconds, but it can be set from 15 milliseconds to 8 seconds. If the timer is not reinitialised with the function

```
wdt_reset()
```

before it ends, the watchdog will restart the arduino.

Beware, when using an arduino nano, if the arduino is a bit old and is burned with the old version of the bootloader, the watchdog timer don't work properly. In this case, if the watchdog is triggered, the timer is not reinitialised, so when the arduino restart the watchdog trigged immediatly and the arduino restart indefinitly. This can be viewed because the L LED blink fast. To solve the problem we have to use another arduino as ISP programmer and burn the new version of the bootloader (or another bootload compatible with the watchdog).

To avoid reboot problem when using a watchog timer (which can lead to not be able to reflash a new code into the arduino), it is recommended to deactivate the watchdog in the setup function with

```
wdt_disable();
```

2.7 PT1000 sensors

2.7.1 Informations

The PT1000 sensors (platine resistor thermometer, french link) are variable resistors depending on the temperature. The resistance of a PTX sensor at 0°C is X Ohms, the variation of the resistance is given by:

$$R(t) = 1 + At + Bt^2, \quad t \geq 0°C, \tag{1}$$

$$R(t) = 1 + At + Bt^2 + C(t-100)t^3, \quad t < 0°C, \tag{2}$$

with $A = 3.9083.10^{-3}$, $B = -5.775.10^{-7}$, $C = -4.183.10^{-12}$.

These sensors will be used for measuring the temperature of hot water pip in the house heating system. So measured temperature will vary between 0 and 100 °C. The normalised resistance variation of a PT1000 sensor considering this input data range is shown on Figure 5.

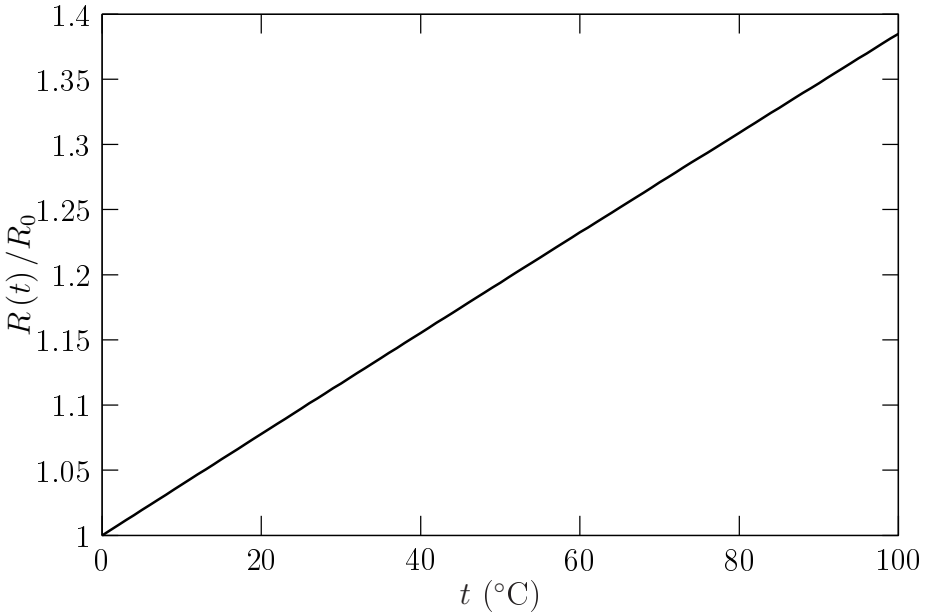


Figure 5: Resistance variation versus temperature.

We see that the normalised resistance varies between 1 and 1.4 quasi-linearly. To retrieve the temperature from the resistance, we use the quadratic formula, solution of a quadratic equation, considering we have positive temperature, we search the positif root of the equation:

$$t = \frac{-b + \sqrt{delta}}{2a} = \frac{-A + \sqrt{A^2 - 4B\,(1 - R)}}{2B}. \tag{3}$$

If we simplify and now consider just a linear variation of the resistance as a function of the temperature ($R\,(t) = 1 + At$), then the formula to retrieve the temperature is:

$$t = (R - 1)/A. \tag{4}$$

The error between these two formula is presented on Figure 6, at maximum it is 1.5 °C at 100 °C. The error is not so high, for small temperature measurement the linear formula is good enougth. But in our case, we prefer to keep the quadratic formula to have the maximum accessible resolution.

2.7.2 Implementation

To retrieve the temperature of the sensor, we need to measure the resistance of this sensor. We can do this simply by using a voltage divider bridge and mesure the tension at sensor ends, considering a reference resistor R_r (Figure fig: pt1000 schematic).

Doing this, the normalised tension at the ends of the sensor is given by:

$$\frac{V}{Vcc} = \frac{R(t)}{(R(t) + R_r}. \tag{5}$$

We can search for the value of the resistor R_r which will lead to the maximum voltage variation at the end of the sensor. The normalised voltage variation amplitude (difference between minimum and maximum tension) depending on R_r is shown Figure 8.

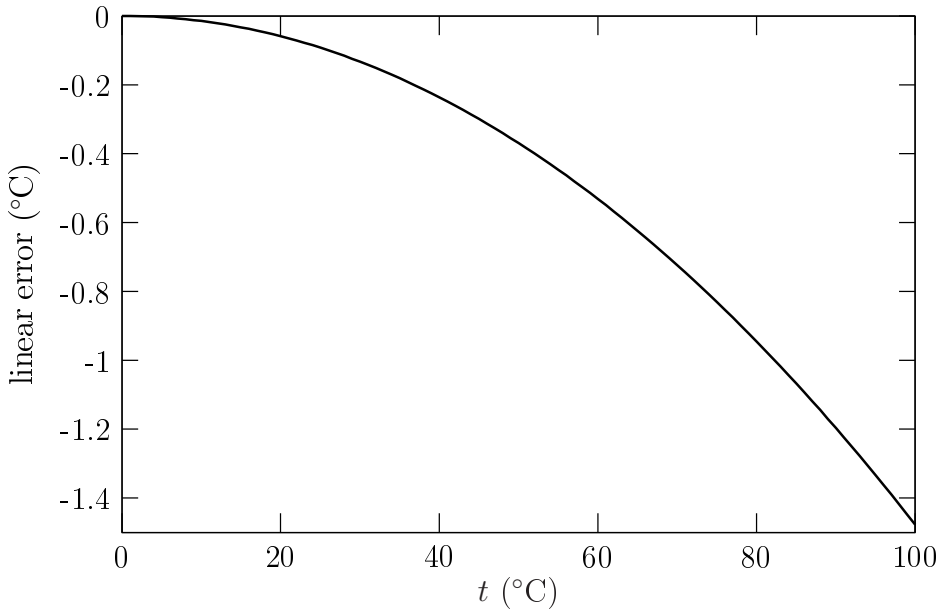


Figure 6: Error using linear approximation compare to quadratic formula to retrieve the temperature.

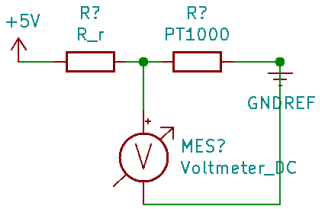


Figure 7: PT1000 implementation.

The optimum reference resistance value is 1.18 kΩ, this resistance leads to a relative tension variation amplitude of 8.13%. For a reference resistor value range between 920 and 1500 Ω, the variation is more than 8%. This is a small voltage radiation, considering the 10 bits ADC of the arduino, this leads to approximately 80 units variation of the ADC output. The theoretical ADC output of the arduino as a function of the temperature is given Figure 9, considering $R_r = 1$ kΩ.

The resolution of the measurement is between 1 and 2°C (retrieve temperature step). Considering ADC errors, measurment error, we can estimate the true resolution of this prototype around +/- 3 °C.

2.7.3 Remarques

When we will install the prototype, their will be extensions between the sensors and board. They will probably and some parasitic resitance. So it will be preferable to recalibrate the prototype on site.

To improve the sensibility of the measurment, we can use an AOP as voltage subtractor (with a LM358 for example). By designing it well, we can multiply the sensibilité by a factor between 2 and 10.

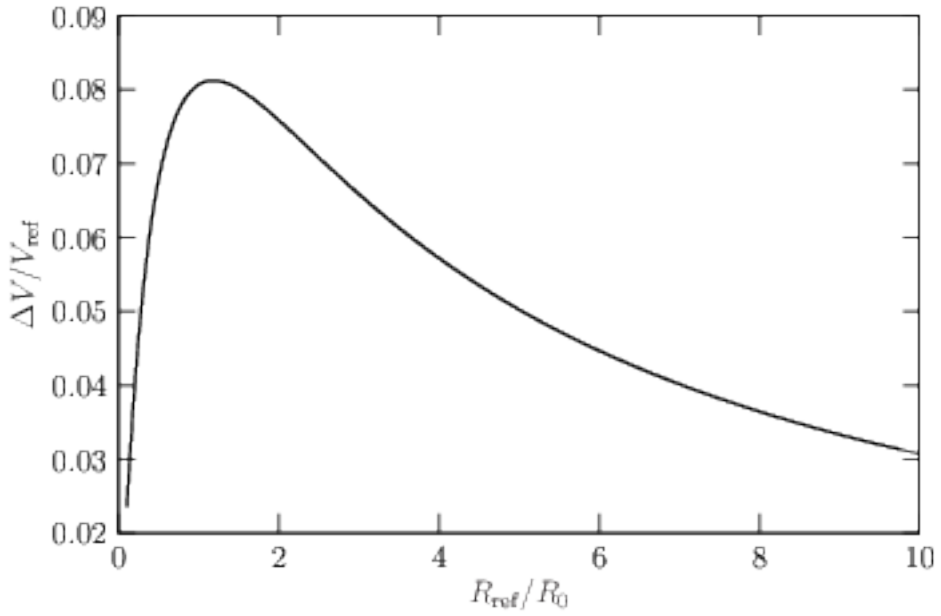


Figure 8: Voltage variation amplitude at the end of a PT1000 sensor (considering temperature range between 0 et 100 °C) versus reference resistor value.

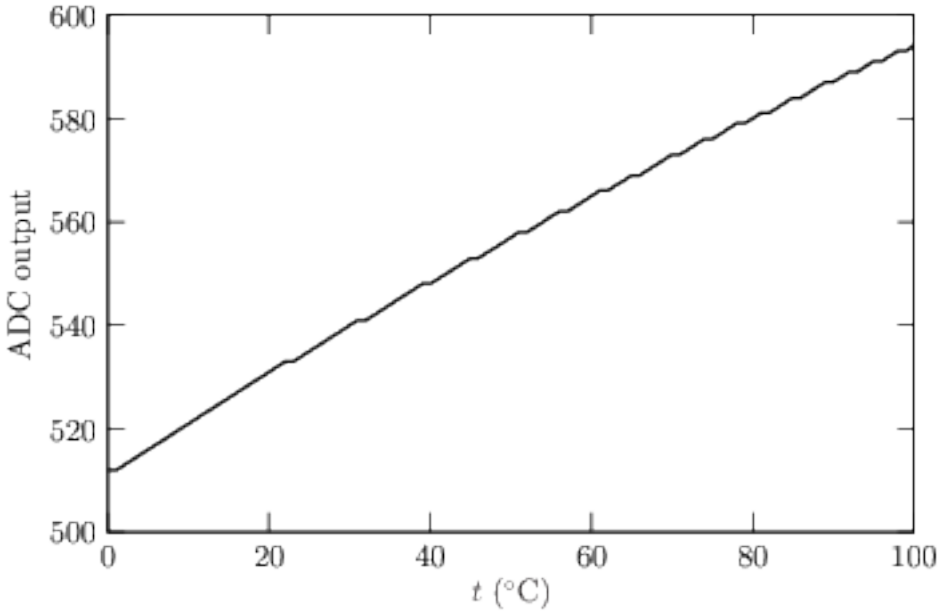
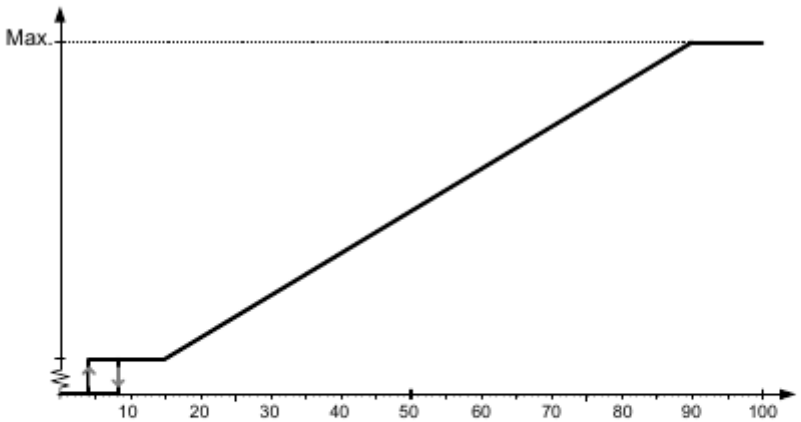


Figure 9: Arduino ADC output versus temperature.

2.8 Pump

We use a upm3 solar pump which can be driven with a PWM (tension between 4 and 24V). The pump PWM wiring is given page 52.

The speed control of the pump versus PWM duty cicle is shown Figure 10 (page 18 of the documentation).



PWM input profile C (solar)

Axis	Value
X	PWM input signal [%]
Y	Speed

PWM input signal [%]	Pump status
≤ 5	Standby mode: off
> 5 / ≤ 8	Hysteresis area: on/off
> 8 / ≤ 15	Minimum speed
> 15 / ≤ 90	Variable speed from minimum to maximum speed.
> 90 / ≤ 100	Maximum speed

Figure 10: Pump regulation with PWM.

3 Full project

3.1 Functioning

With this prototype, we display "in real time" the time (first line left) and the temperature of the selected sensor T1 to T4 (second line). The displayed temperature is average over the last 10 measurement (the prototype make aroud 5 measurement per seconds). One can change the display sensor temperature with the rotary encoder. On first line right, we display if an SD card is plug in the module or not (but be carefull, if no SD card is inserted, the initialisation of the module is quite long, around 2 seconds, so if there is no SD card in the module, the display is laggy). If a SD card is plug in the module, we save the temperatures measured by the four sensors in a text file which is name YYYYMMDD.txt each time step (default time step is 5 minutes). When clicking on the rotary encoder, one can change the time step with 10 seconds step.

The arduino code is in the soft_arduino file. The firmware works as follow, at each loop:

- we reset the watchdog,
- we ask the time to the RTC and we display it on the lcd screen,
- we measure the temperature of each sensor and save it in a "rotary memory",
- we display the temperature of the selected sensor (average over the 10 last measurement) on the lcd screen,
- if one has clicked on the rotary encoder, we launch the edition time step function,
- if no SD card was plug into the module at the last loop (or nonaccessible SD card), we reinitialise the module,
- if there was an SD card in the module at the last loop, we check it is still true by openning a text file (this text file is useless except to do this check function). If there is an error with the open() function, this mean that the SD must be unpluged so we display their is no SD card,
- if the time given by the RTC is greater or equal to next save time, we save the temperature of the fourth sensors average over the last 10 measurement. The name of the file is the actual date writen as YYYYMMDD.txt.

3.2 Montage

The schematic of the full prototype is given Figure 11. The project is relatively simple, but it use almost all the pins of the arduino, only the Rx and Tx pins are still free. For repetability and fiability purpose, we have to use $\pm 1\%$ resistor.

To not have a screen which is always power on, one can use a transistor or a switch on the screen anode (pin A).

The T1 to T4 sensors are wired respectively to socket J2 to J5 then respectively to A7, A6, A3 and A2. The pump is wire to socket J1 (GPIO 9 of the arduino), the ground of the pump PWM is the blue wire (*cf.* datasheet page 52).

3.3 Assembly

Two photos of the assembly are shown on Figures 12 and 13.

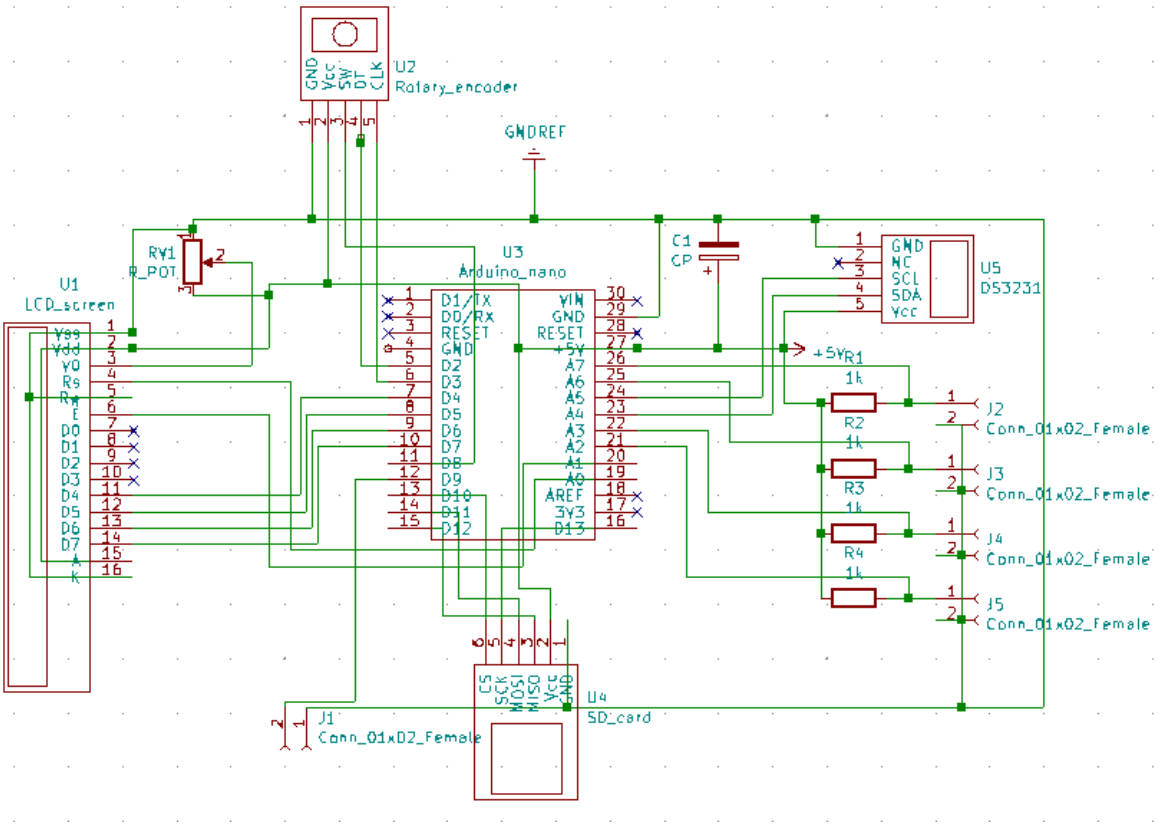


Figure 11: Full project schematic.

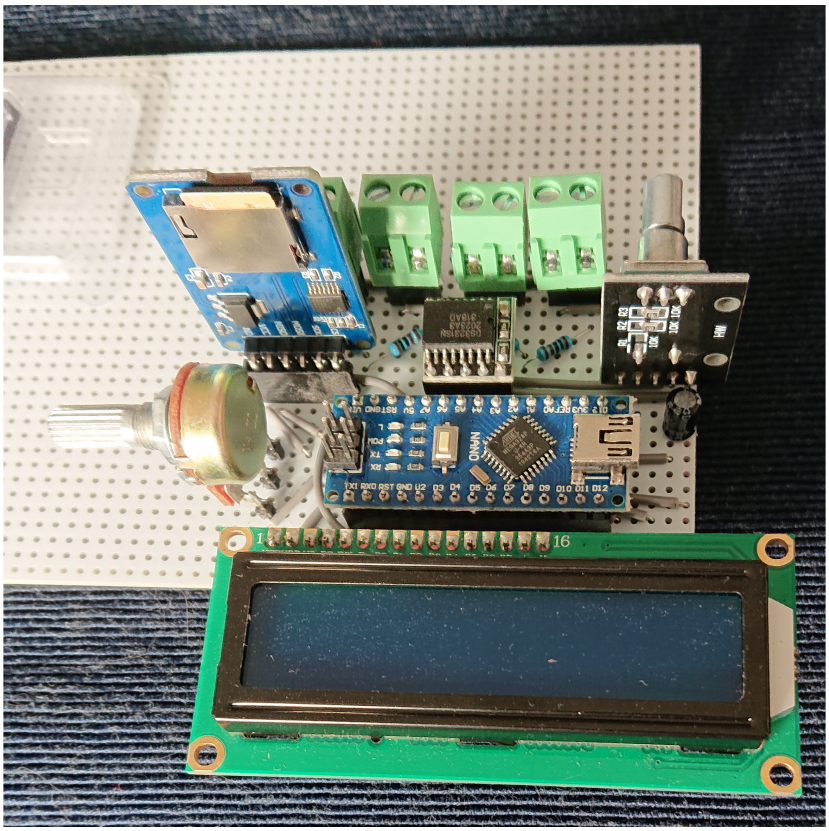


Figure 12: Photo of final assembly.

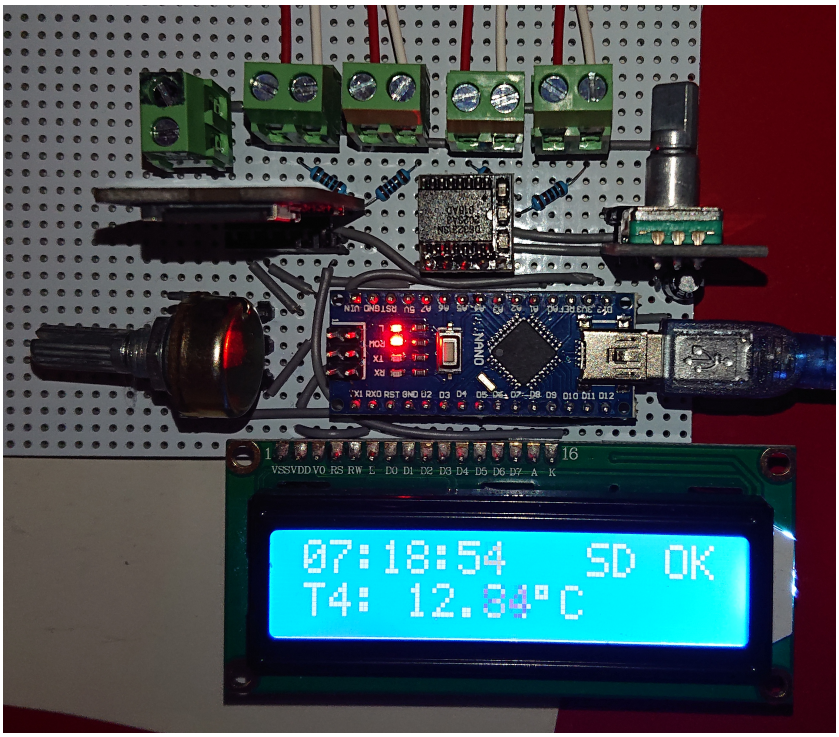


Figure 13: Photo of final assembly.