



ThermistorNTC library

ARDUINO®

THERMISTORNTC

Library to used to derive a precise temperature of a thermistor, fastest Calc. (14~29% faster than others)

Introduction

The thermistor changes its resistance drastically with temperature. "Thermistor" word comes from "thermally sensitive resistor". The resistance of ordinary materials rises slightly as temperature rises, whereas NTC (negative temperature coefficient) thermistors exhibit a sharp decrease in resistance. For information about thermistor see manufacturer information. Shibaura manufacturer has a lot of [technical information](#) about NTC thermistors.

The Steinhart-Hart equation is the most widely used tool to interpolate the NTC thermistor resistance/temperature curve. It is a third order polynomial equation which provides very good curve fitting.

$$\frac{1}{T} = A + B * \ln(R) + C * \ln^2(R) + D * \ln^3(R)$$

In the standard Steinhart-Hart equation the C parameter is set to zero. However, some manufacturers use all 4 coefficients. So we can use standard Steinhart-Hart equation with 3 coefficients.

$$\frac{1}{T} = A + B * \ln(R) + D * \ln^3(R)$$

where:

- T is the temperature (in kelvins),
- R is the resistance at T (in ohms),
- A, B, C and D are the Steinhart–Hart coefficients, which vary depending on the type and model of thermistor and the temperature range of interest. These can usually be found in the data sheet.

Other expressions

Other form of the equation is the use of B (beta) parameter

where:

- T is the temperature (in kelvins)
- T₀ is 298'15 °K (25 °C)
- R is the resistance at T (in ohms),
- NTC is the resistance of thermistor at 298'15 °K (25 °C)
- beta is the Steinhart-Hart beta coefficient that vary depending on the type and model of Thermistor. It can usually be found in the data sheet.

$$R = NTC * e^{\left(\beta * \left(\frac{1}{T} - \frac{1}{T_0}\right)\right)}$$

Some manufacturers have begun providing regression coefficients as an alternative to Steinhart–Hart coefficients. See this document for more information. ["Comments on the Steinhart–Hart Equation"](#) (PDF). Building Automation Products Inc. 11 November 2015. Retrieved 8 July 2020.

The most general form of the equation can be derived from extending the B parameter equation to an infinite series:

$$R = R_0 * e^{\beta \left(\frac{1}{T} - \frac{1}{T_0} \right)}$$

$$\frac{R}{R_0} = e^{\beta * \left(\frac{1}{T} - \frac{1}{T_0} \right)}$$

$$\ln \left(\frac{R}{R_0} \right) = \beta * \left(\frac{1}{T} - \frac{1}{T_0} \right)$$

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{\beta} \left(\ln \frac{R}{R_0} \right) = a_0 + a_1 \ln \frac{R}{R_0}$$

$$\frac{1}{T} = \sum_{n=0}^{\infty} a_n \left(\ln \frac{R}{R_0} \right)^n$$

You can get more information in this document: Matus, Michael (October 2011). [Temperature Measurement in Dimensional Metrology](#) – Why the Steinhart–Hart Equation works so well. MacroScale 2011. Wabern, Switzerland

More Information

[Thermistor](#) is the principal element of temperature sensor.

[Look here](#) for more information about Steinhart-Hart equations.

See this page for info about [NTC Thermistors Steinhart and Hart Equation](#)

You can learn more about [temperature coefficient here](#).

More information:

- [Arrhenius equation](#).
- [Q10 temperature coefficient](#).

Fast Calc

In this library, it is take the beta equation and calculate temperature of the thermistor from it.

$$\frac{R}{NTC} = e^{\left(\beta * \left(\frac{1}{T} - \frac{1}{T_0}\right)\right)}$$

$$\ln\left(\frac{R}{NTC}\right) = \beta * \left(\frac{1}{T} - \frac{1}{T_0}\right)$$

$$\frac{\ln\left(\frac{R}{NTC}\right)}{\beta} = \frac{1}{T} - \frac{1}{T_0}$$

$$\frac{1}{T} = \frac{1}{T_0} + \frac{\ln\left(\frac{R}{NTC}\right)}{\beta}$$

$$\frac{1}{T} = \frac{\beta + T_0 * \ln\left(\frac{R}{NTC}\right)}{\beta * T_0}$$

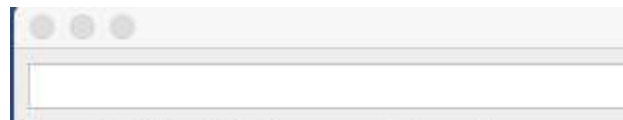
So we can solve using this formula:

where:

- T is the temperature (in kelvins)
- T0 is 298'15 °K (25 °C)
- R is the resistance at T (in ohms),
- NTC is the resistance of thermistor at 298'15 °K (25 °C)
- **beta** is the Steinhart-Hart beta coefficient that vary depending on the type and model of Thermistor.

$$T = \frac{\beta * T_0}{\beta + T_0 * \ln\left(\frac{R}{NTC}\right)}$$

The library is 14-29% faster than others libraries to get temperature from Thermistor if we compare this library with libraries based on beta equation. And it depends on the board used. Tested on LGT8F328P-SOPP, LGT8F328P-QF32 and Arduino pro mini boards. If it is compared Fast Calc with Steinhart-Hart three order equation, it is 35-44% faster, depends on microcontroller used. See example for test.



```
Sensor0(°C): 50 (microsecs) to calc.  
Sensor1(°C): 30 (microsecs) to calc.  
Sensor0(°C): 48 (microsecs) to calc.  
Sensor1(°C): 30 (microsecs) to calc.  
Sensor0(°C): 48 (microsecs) to calc.  
Sensor1(°C): 32 (microsecs) to calc.  
Sensor0(°C): 48 (microsecs) to calc.  
Sensor1(°C): 26 (microsecs) to calc.  
Sensor0(°C): 46 (microsecs) to calc.  
Sensor1(°C): 14 (microsecs) to calc.  
Sensor0(°C): 50 (microsecs) to calc.  
Sensor1(°C): 32 (microsecs) to calc.  
Sensor0(°C): 48 (microsecs) to calc.  
Sensor1(°C): 32 (microsecs) to calc.  
Sensor0(°C): 48 (microsecs) to calc.  
Sensor1(°C): 26 (microsecs) to calc.
```

Testing **sensor0** Steinhart-Hart three order equation, **sensor1** Fast Calc equation. LGT8F328P-SOPP board. (32 MHz 5v.)

```
Sensor0(°C): 48 (microsecs) to calc.  
Sensor1(°C): 38 (microsecs) to calc.  
Sensor1_fast(°C): 30 (microsecs) to calc.  
Sensor0(°C): 50 (microsecs) to calc.  
Sensor1(°C): 36 (microsecs) to calc.  
Sensor1_fast(°C): 16 (microsecs) to calc.  
Sensor0(°C): 50 (microsecs) to calc.  
Sensor1(°C): 40 (microsecs) to calc.  
Sensor1_fast(°C): 28 (microsecs) to calc.  
Sensor0(°C): 46 (microsecs) to calc.  
Sensor1(°C): 38 (microsecs) to calc.  
Sensor1_fast(°C): 32 (microsecs) to calc.  
Sensor0(°C): 48 (microsecs) to calc.  
Sensor1(°C): 38 (microsecs) to calc.  
Sensor1_fast(°C): 28 (microsecs) to calc.
```

Testing **sensor0** Steinhart-Hart three order equation, **sensor1** beta equation and **sensor1_fast** Fast Calc equation. LGT8F328P-QF32 board. (32 MHz 5v.)

```

Sensor0(°C): 112 (microsecs) to calc. Temp(°C): -5.26
Sensor1(°C): 84 (microsecs) to calc. Temp(°C): -5.76
Sensor1_fast(°C): 68 (microsecs) to calc. Temp(°C): -5.13
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): -5.67
Sensor1(°C): 84 (microsecs) to calc. Temp(°C): -6.49
Sensor1_fast(°C): 72 (microsecs) to calc. Temp(°C): -5.11
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): -6.08
Sensor1(°C): 84 (microsecs) to calc. Temp(°C): -6.97
Sensor1_fast(°C): 72 (microsecs) to calc. Temp(°C): -5.19
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): -6.61
Sensor1(°C): 84 (microsecs) to calc. Temp(°C): -7.23
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): -5.17
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): -7.20
Sensor1(°C): 84 (microsecs) to calc. Temp(°C): -7.23
Sensor1_fast(°C): 72 (microsecs) to calc. Temp(°C): -5.23
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): -7.71
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): -7.07
Sensor1_fast(°C): 72 (microsecs) to calc. Temp(°C): -5.38

```

Testing **sensor0** Steinhart-Hart three orden equation, **sensor1** beta equation and **sensor1_fast** Fast Calc equation. Atmega328p board. (16 MHz 5v.)

The screenshot shows the Arduino IDE interface. On the left, the 'Serial Monitor' window displays the output of the sketch, showing temperature calculations for three sensors (sensor0, sensor1, sensor1_fast) using different equations. On the right, the 'Sketch' window shows the code for the SteinhartHart sketch, which includes the SteinhartHart.h library and defines two thermistors (thermistor0 and thermistor1) with their respective parameters. The code also includes a setup function that initializes the serial communication and sets the debug level for the thermistors.

```

COM3
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): 26.12
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.21
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): 34.20
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): 25.95
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.19
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): 34.11
Sensor0(°C): 112 (microsecs) to calc. Temp(°C): 25.93
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.17
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): 34.20
Sensor0(°C): 124 (microsecs) to calc. Temp(°C): 25.80
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.19
Sensor1_fast(°C): 80 (microsecs) to calc. Temp(°C): 34.12
Sensor0(°C): 116 (microsecs) to calc. Temp(°C): 25.89
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.17
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): 34.12
Sensor0(°C): 116 (microsecs) to calc. Temp(°C): 25.68
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.20
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): 34.20
Sensor0(°C): 116 (microsecs) to calc. Temp(°C): 25.80
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.18
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): 34.01
Sensor0(°C): 120 (microsecs) to calc. Temp(°C): 25.94
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.10
Sensor1_fast(°C): 80 (microsecs) to calc. Temp(°C): 34.20
Sensor0(°C): 116 (microsecs) to calc. Temp(°C): 25.80
Sensor1(°C): 88 (microsecs) to calc. Temp(°C): 34.27
Sensor1_fast(°C): 76 (microsecs) to calc. Temp(°C): 34.10
Sensor0(°C): 116 (microsecs) to calc. Temp(°C): 25.88

Autoscroll  Mostrar marca temporal  Ambos NL & CR

SteinhartHart Arduino 1.8.13 (Windows Store 1.8.42.0)
Archivo Editar Programa Herramientas Ayuda
SteinhartHart §
*/
#include <SteinhartHart.h>

Thermistor thermistor0( /* PIN */ A0,
/* RESISTOR */ 21900L,
/* NTC 25°C */ 97000L,
/* A */ 3354016e-9,
/* B */ 2569850e-10,
/* C */ 2620131e-12,
/* D */ 6383091e-14,
/* Vref */ 4.97);

Thermistor thermistor1( /* PIN */ A1,
/* RESISTOR */ 21900L,
/* NTC 25°C */ 97000L,
/* BETA */ 4390.0,
/* Vref */ 4.97);

void setup(void)
{
  Serial.begin(57600);
  thermistor0.setDEBUG(1);
  thermistor1.setDEBUG(1);
}

Subido
El Sketch usa 5722 bytes (18%) del espacio de almacenami
Las variables Globales usan 390 bytes (19%) de la memori

```

Testing with 100k thermistors and 22kohms. Testing **sensor0** Steinhart-Hart three orden equation, **sensor1** beta equation and **sensor1_fast** Fast Calc equation. Atmega328p board. (16 MHz 5v.)

Calculation of beta

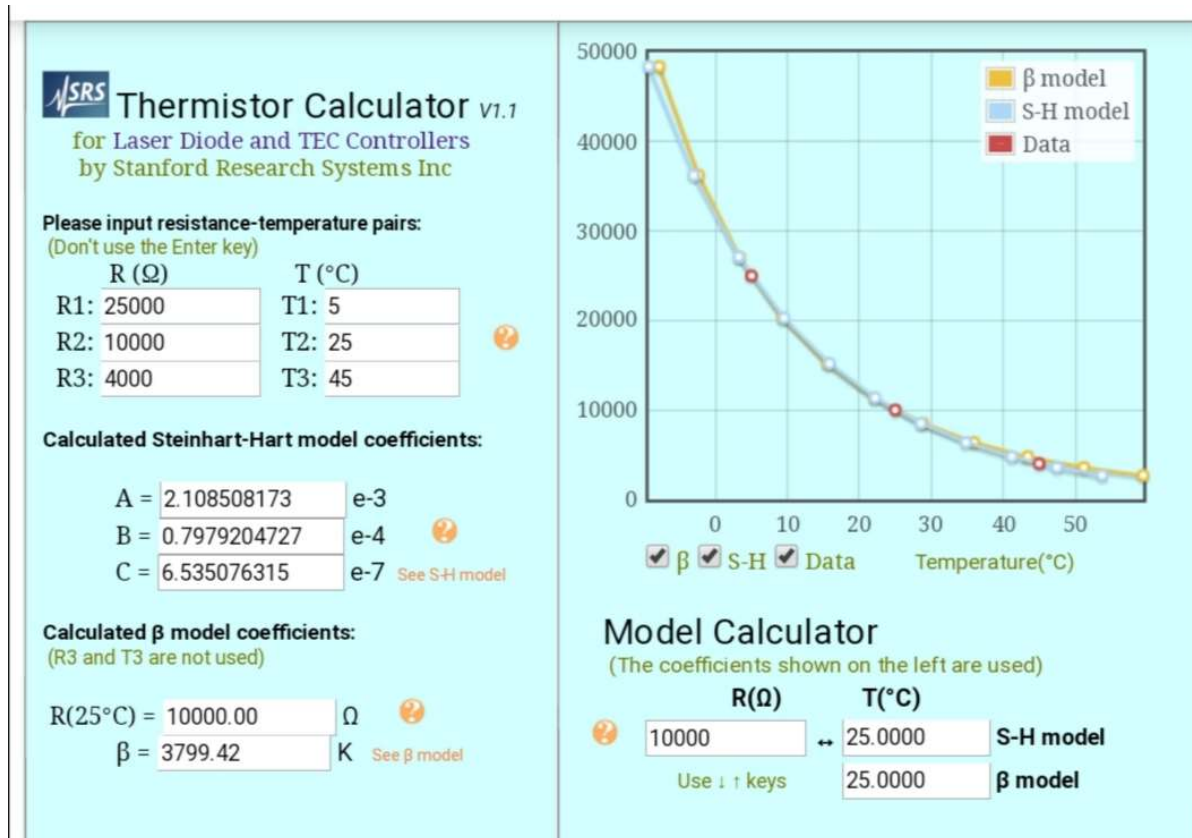
Beta is measured in degrees Kelvin (K) and is computed based on this equation:

$$\beta = \left(\frac{\ln \left(\frac{R_{T_1}}{R_{T_2}} \right)}{\frac{1}{T_1} - \frac{1}{T_2}} \right)$$

Where:

- R_{T1} = Resistance at Temperature 1 (ohms)
- R_{T2} = Resistance at Temperature 2 (ohms)
- T_1 = Temperature 1 in (Kelvin)
- T_2 = Temperature 2 in (Kelvin)

How to Calc beta of NTC Thermistor, you can [see this page](#).



Using the [Thermistor Calculator V1.1](#) you can determine the unknown parameters of a thermistor.

For more information about NTC thermistors and Steinhart-Hart equation to calculate the parameters using three pairs of values (temperature, NTC resistance) see [NTC Thermistors Steinhart and Hart Equation](#)

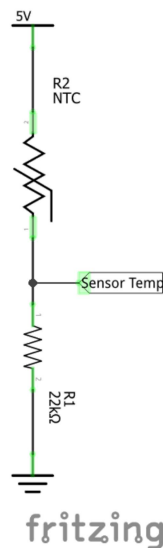
How to use Thermistor as temperature sensor

Thermistor values denote their resistance at 25°C. A popular type would be an NTC 10K which would give roughly 10 kOhms at that temperature point.

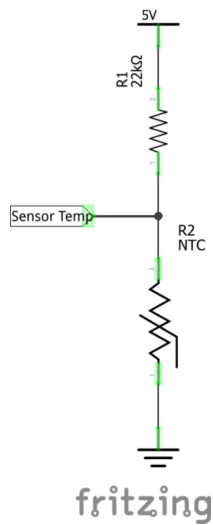
Although there are ways to calculate the coefficients yourself experimentally it might be cheaper and easier to just buy a thermistor with known specs. This page can help to calculate those coefficients. [Thermistor Calculator V1.1](#).

To get readings from a thermistor into your Arduino you will have to use a conventional voltage divider circuit. It can be used in two forms of configurations.

Connecting NTC thermistor to VCC:



Or connecting NTC thermistor to GND:



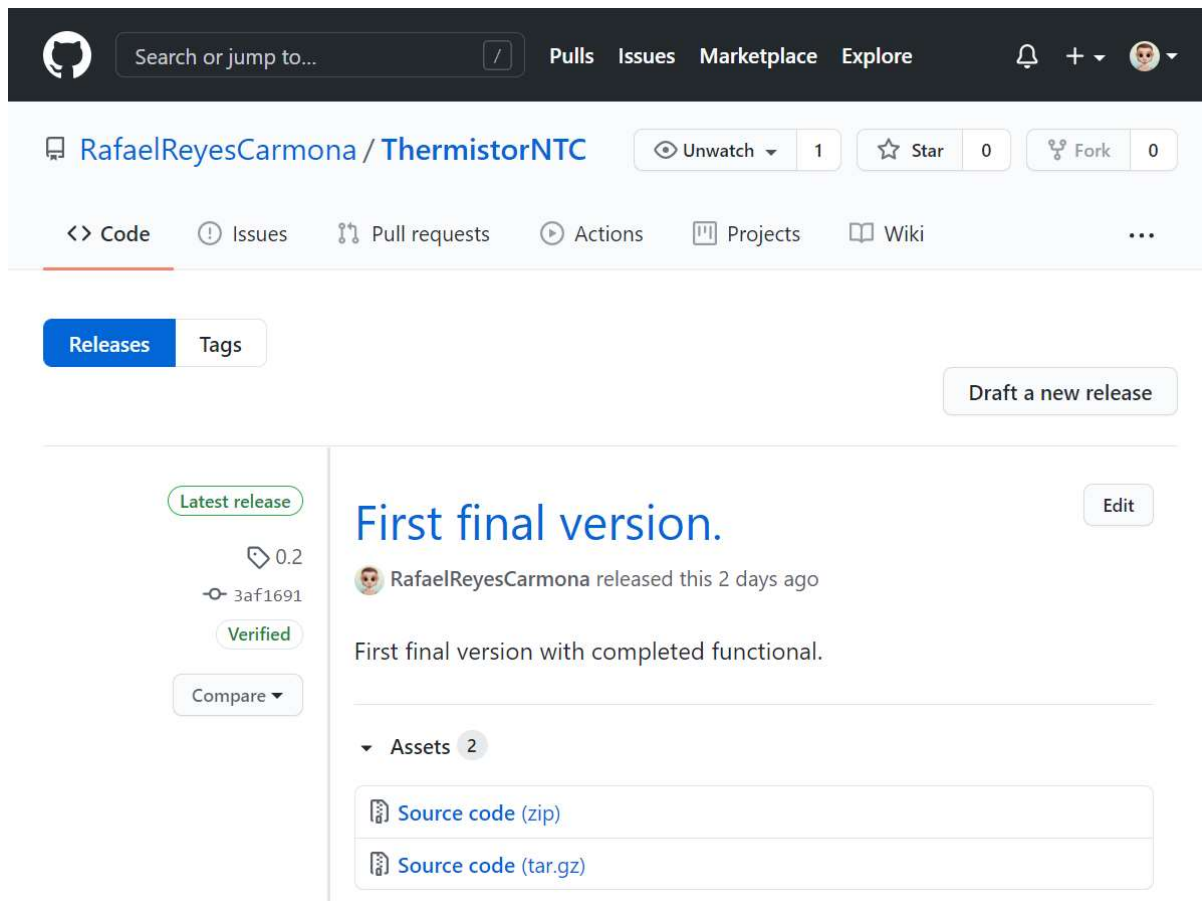
Installation

For a tutorial on how to install new libraries for use with the Arduino development environment please refer to the following website:

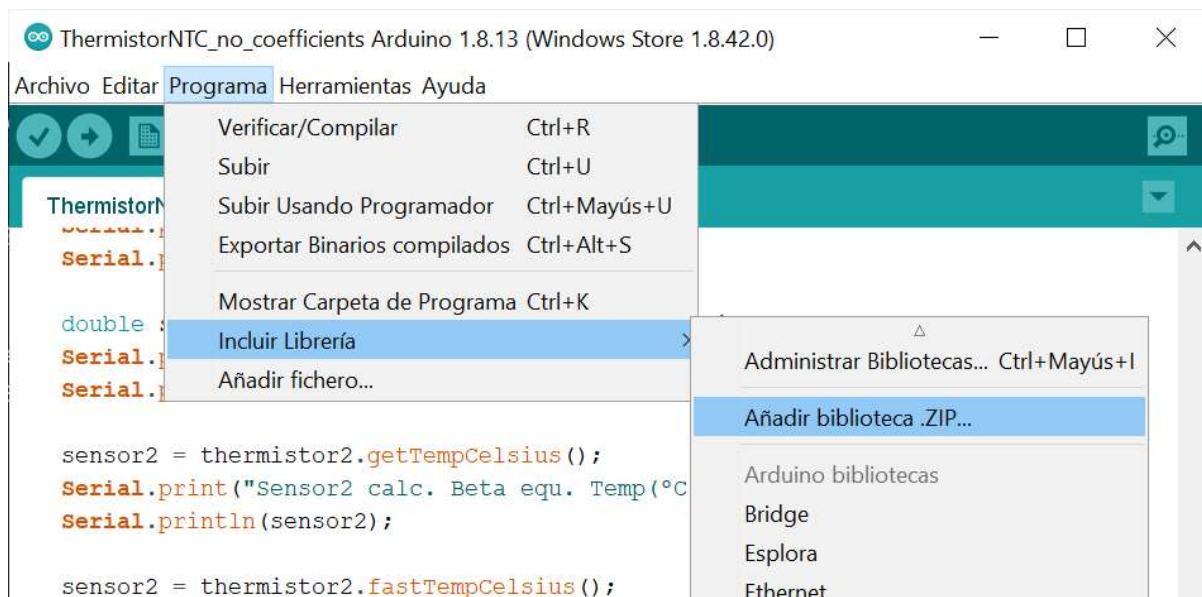
<http://www.arduino.cc/en/Reference/Libraries>

--- or ---

1. Download the ZIP file from the page [releases](#) to your machine.



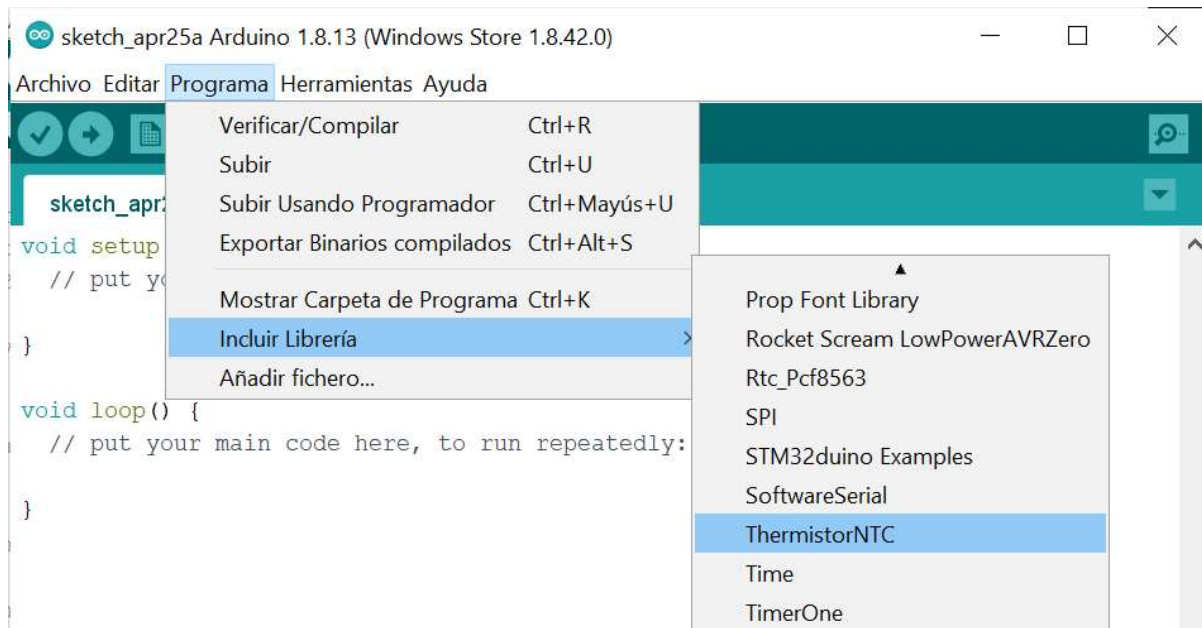
2. In the Arduino IDE, choose Sketch/Include Library/Add Zip Library.



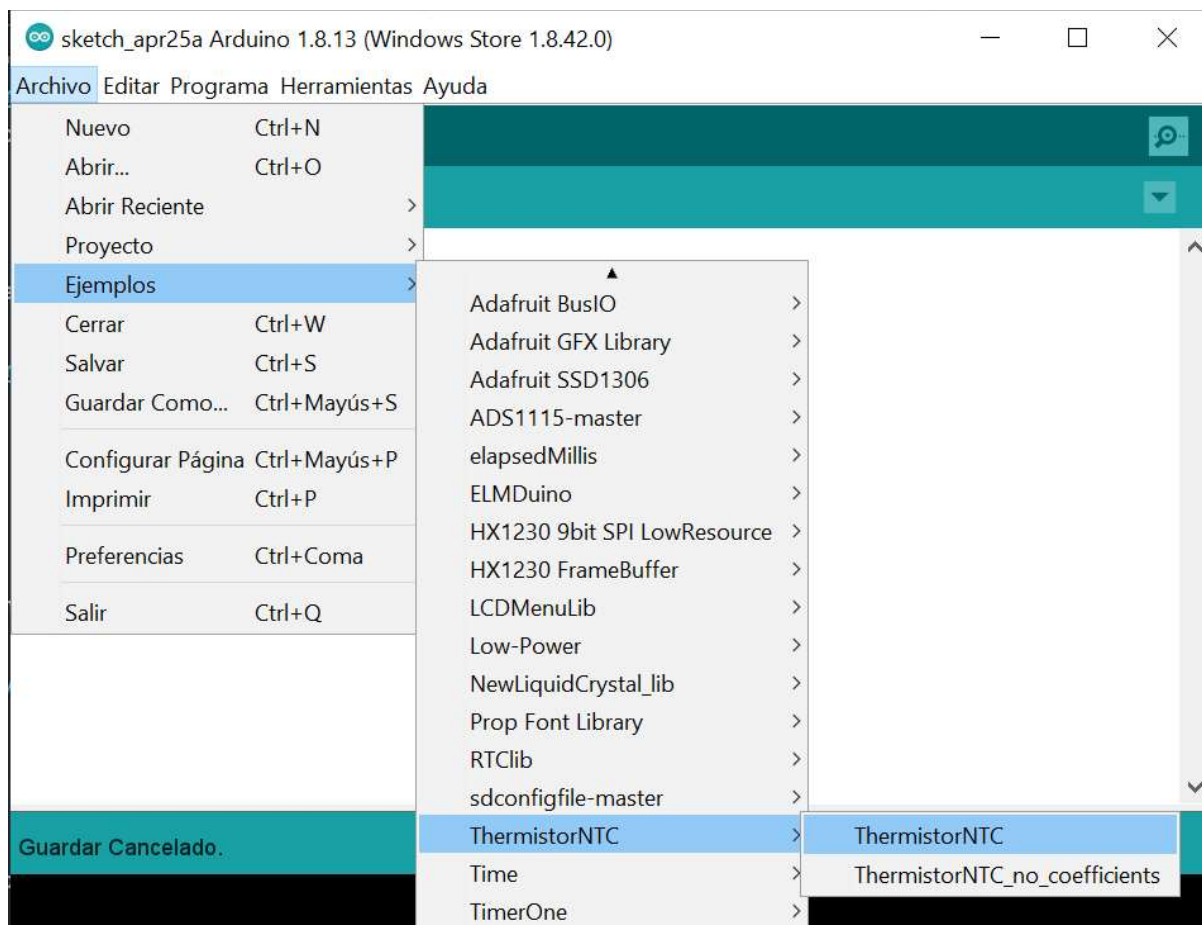
3. Navigate to the ZIP file, and click Open.

How to use the library

In Arduino IDE, Choose Sketch/Include Library/Scroll and select "ThermistorNTC".



There are two example files with the library. In the Arduino IDE, choose File/Examples/ThermistorNTC, and you can see "ThermistorNTC" and "ThermistorNTC_no_coefficients".



--- or ---

See the example code in section [Simple example](#) above.

The library implement the type:

```
enum Thermistor_connection {
  VCC,
  GND
};
```

Usage:

```
// VCC or GND where thermistor configuration.
// If no value, use VCC as default.
//
double sensor0 = thermistor0.getTempCelsius(VCC);
//double sensor0 = thermistor0.getTempCelsius(GND);
//double sensor0 = thermistor0.getTempCelsius(); /* VCC as default. */
```

Constructors

```
// Constructor for thermistor 4 coefficients Steinhart-Hart equation.
```

```
Thermistor::Thermistor(int PIN,  
                        long RESISTOR,  
                        long NTC_25C,  
                        double A,  
                        double B,  
                        double C,  
                        double D,  
                        float VREF)
```

```
// Constructor for thermistor 3 coefficients Steinhart-Hart equation.
```

```
// Some manufacturers use C coefficient equal to 0 for simplicity.
```

```
Thermistor::Thermistor(int PIN,  
                        long RESISTOR,  
                        long NTC_25C,  
                        double A,  
                        double B,  
                        double D,  
                        float VREF)
```

```
// Constructor for thermistor beta equation.
```

```
Thermistor::Thermistor(int PIN,  
                        long RESISTOR,  
                        long NTC_25C,  
                        float BETA,  
                        float VREF)
```

```
// Constructor for unknowns thermistor parameters. (3 measures)
```

```
Thermistor::Thermistor(int PIN,  
                        long RESISTOR,  
                        long NTC_1,  
                        float TEMP_1,  
                        long NTC_2,  
                        float TEMP_2,  
                        long NTC_3,  
                        float TEMP_3,  
                        float VREF)
```

```
// Constructor for unknowns thermistor parameters. (4 measures)
```

```
Thermistor::Thermistor(int PIN,  
                        long RESISTOR,  
                        long NTC_1,  
                        float TEMP_1,  
                        long NTC_2,  
                        float TEMP_2,
```

```
long NTC_3,  
float TEMP_3,  
long NTC_4,  
float TEMP_4,  
float VREF)
```

Where:

- **PIN** - Analog port for get ADC (analogRead() function)
- **RESISTOR** - Value in ohms of resistor in voltage divisor.
- **NTC_25C** - Resistance value of NTC thermistor at 298.15°K (25°C)
- **A, B, C, D** - NTC Thermistor coefficients
- **BETA** - Beta coefficient of NTC thermistor.
- **VREF** - Voltage applied to voltage divisor (usually VCC.)
- **NTC_1, NTC_2, NTC_3, NTC_4** - Resistance value of NTC thermistor at different temperatures.
- **TEMP_1, TEMP_2, TEMP_3, TEMP_4** - Temperature value in (°C). Measures of Temperature should be $TEMP_1 < TEMP_2 < TEMP_3 < TEMP_4$, for get better estimated values, but not mandatory. The temperatures should be evenly spaced and at least 10 degrees apart for better results.

Functions implemented

```
void setADC(int);  
void setEMA(float);  
  
double getTempKelvin(Thermistor_connection ConType);  
double getTempCelsius(Thermistor_connection ConType);  
double getTempFahrenheit(Thermistor_connection ConType);  
  
double fastTempKelvin(Thermistor_connection ConType);  
double fastTempCelsius(Thermistor_connection ConType);  
double fastTempFahrenheit(Thermistor_connection ConType);  
  
double getTempKelvin_SteinHart(Thermistor_connection ConType);  
double getTempCelsius_SteinHart(Thermistor_connection ConType);  
double getTempFahrenheit_SteinHart(Thermistor_connection ConType);
```

- **setADC** - Set maximal value of ADC. For 10-bits ADC resolution, will be $2^{10} = 1024$. For 12-bits ADC resolution, the value will be $2^{12} = 4096$. Library autodetect for Atmega328, ATmega168 and LGT8F328P.
- **setEMA** - Set value for EMA Filter ADC readings. Default is 0,91. The library gets 15 values from ADC port at maximal resolution every time it is called for estimated the temperature. It is used the EMA filtered ADC value with formula: $Y[n] = \alpha * X[n] + (1 - \alpha) * Y[n-1]$, to estimate the temperature. For more info about EMA (Exponential Moving Average) see [Exponential Moving Average](#), and how to implement it [EMA C++ Implementation](#).
- **getTemp...** Return the temperature in the respective range using Beta equation.
- **fastTemp...** Return the temperature in the respective range using Fast equation. More fast than the other methods. It is used beta parameter.
- **getTemp..._Steinhart** Return the temperature in the respective range using Steinhart-Hart equation.

Simple Example

```
#include <ThermistorNTC.h>

Thermistor thermistor0(/* PIN */      A0,
                       /* RESISTOR */ 21900L,
                       /* NTC 25°C */ 9950L,
                       /* A */        3354016e-9,
                       /* B */        2569850e-10,
                       /* C */        2620131e-12,
                       /* D */        6383091e-14,
                       /* Vref */      5.03);

Thermistor thermistor1(/* PIN */      A1,
                       /* RESISTOR */ 21900L,
                       /* NTC 25°C */ 9950L,
                       /* BETA */     4190.0,
                       /* Vref */     5.03);

void setup(void)
{
    Serial.begin(57600);
}
```



```

void loop(void)
{
    double sensor0 = thermistor0.getTempCelsius_SteinHart(VCC);
    Serial.print("Sensor0 - Temp(°C): ");
    Serial.println(sensor0);

    double sensor1 = thermistor1.getTempCelsius(); // VCC default if not expressed.
    Serial.print("Sensor1 - Temp(°C): ");
    Serial.println(sensor1);

    double sensor1_fast = thermistor1.fastTempCelsius();
    Serial.print("Sensor1 fast calc - Temp(°C): ");
    Serial.println(sensor1_fast);

    delay(1000);
}

```

Thermistor with unknowns coefficients

```

Thermistor thermistor1(/* PIN */      A1,
                        /* RESISTOR */ 22170L,
                        /* NTC_T1 */   355000L,
                        /* T1 (°C) */   0.0, // 273,15 °K
                        /* NTC_T2 */   157500L,
                        /* T1 (°C) */   28.0, // 301,15 °K
                        /* NTC_T3 */   58300L,
                        /* T1 (°C) */   35.0, // 308,15 °K
                        /* Vref */      5.072);

```

Or:

```

Thermistor thermistor2(/* PIN */      A2,
                        /* RESISTOR */ 22170L,
                        /* NTC_T1 */   355000L,
                        /* T1 (°C) */   0.0, // 273,15 °K
                        /* NTC_T2 */   157500L,
                        /* T2 (°C) */   14.0, // 287,15 °K
                        /* NTC_T3 */   79300L,
                        /* T3 (°C) */   28.0, // 301,15 °K
                        /* NTC_T4 */   58300L,
                        /* T4 (°C) */   35.0, // 308,15 °K
                        /* Vref */      5.072);

```

When the coefficients are unknowns, It can use the above Constructor. The library calculate all the coefficients. It can use although getTemp... and fastTemp... Functions. It must measure the thermistor resistance at three or four different temperatures. The temperatures should be evenly spaced and at least 10 degrees apart for better results.

When it is used the Constructors above the library calc **A, B, C, D, BETA** and **NTC_25C** parameters. So, it can use any function to get the temperature. When use three pair of values (Resistance of NTC, Temperature °C), **A, B** and **D** parameters are estimated. As you can get with [Thermistor Calculator V1.1](#). But if it use four pairs of them (Resistance of NTC, Temperature °C), **A, B, C** and **D** parameters are estimated.

Unknow coefficients whith 3 measures of Resistance and Temperature

$$\frac{1}{T_1} = A + B * \ln(R_1) + D * \ln^3(R_1)$$

$$\frac{1}{T_2} = A + B * \ln(R_2) + D * \ln^3(R_2)$$

$$\frac{1}{T_3} = A + B * \ln(R_3) + D * \ln^3(R_3)$$

$$\begin{bmatrix} 1 & \ln R_1 & \ln^3 R_1 \\ 1 & \ln R_2 & \ln^3 R_2 \\ 1 & \ln R_3 & \ln^3 R_3 \end{bmatrix} \begin{bmatrix} A \\ B \\ D \end{bmatrix} = \begin{bmatrix} \frac{1}{T_1} \\ \frac{1}{T_2} \\ \frac{1}{T_3} \end{bmatrix}$$

$$L_1 = \ln R_1, \quad L_2 = \ln R_2, \quad L_3 = \ln R_3$$

$$Y_1 = \frac{1}{T_1}, \quad Y_2 = \frac{1}{T_2}, \quad Y_3 = \frac{1}{T_3}$$

$$\gamma_2 = \frac{Y_2 - Y_1}{L_2 - L_1}, \quad \gamma_3 = \frac{Y_3 - Y_1}{L_3 - L_1}$$

$$\Rightarrow D = \frac{\gamma_3 - \gamma_2}{(L_3 - L_2)(L_1 + L_2 + L_3)}$$

$$\Rightarrow B = \gamma_2 - D (L_1^2 + L_1 L_2 + L_2^2)$$

$$\Rightarrow A = Y_1 - L_1 (B + D L_1^2)$$

You can get **A**, **B** and **D** parameters with [Thermistor Calculator V1.1](#) or solving the system of equations above. For this, there are a utility that can help to solve the system ([Resolución de ecuaciones lineales.](#))

Unknow coefficients with 4 measures of Resistance and Temperature

$$\frac{1}{T_1} = A + B * \ln (R_1) + C * \ln^2 (R_1) + D * \ln^3 (R_1)$$

$$\frac{1}{T_2} = A + B * \ln (R_2) + C * \ln^2 (R_2) + D * \ln^3 (R_2)$$

$$\frac{1}{T_3} = A + B * \ln (R_3) + C * \ln^2 (R_3) + D * \ln^3 (R_3)$$

$$\frac{1}{T_4} = A + B * \ln (R_4) + C * \ln^2 (R_4) + D * \ln^3 (R_4)$$

$$\begin{bmatrix} 1 & \ln R_1 & \ln^2 R_1 & \ln^3 R_1 \\ 1 & \ln R_2 & \ln^2 R_2 & \ln^3 R_2 \\ 1 & \ln R_3 & \ln^2 R_3 & \ln^3 R_3 \\ 1 & \ln R_4 & \ln^2 R_4 & \ln^3 R_4 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = \begin{bmatrix} \frac{1}{T_1} \\ \frac{1}{T_2} \\ \frac{1}{T_3} \\ \frac{1}{T_4} \end{bmatrix}$$

$$L_1 = \ln R_1, \quad L_2 = \ln R_2, \quad L_3 = \ln R_3, \quad L_4 = \ln R_4$$

$$Y_1 = \frac{1}{T_1}, \quad Y_2 = \frac{1}{T_2}, \quad Y_3 = \frac{1}{T_3}, \quad Y_4 = \frac{1}{T_4}$$

$$\begin{aligned} DSA_1 &= (L_3^2 - L_1^2) * (L_2 - L_1) - (L_2^2 - L_1^2) * (L_3 - L_1), \\ DSA_2 &= (L_4^3 - L_1^3) * (L_2 - L_1) - (L_2^3 - L_1^3) * (L_4 - L_1), \\ DSB_1 &= (L_4^2 - L_1^2) * (L_2 - L_1) - (L_2^2 - L_1^2) * (L_4 - L_1), \\ DSB_2 &= (L_3^3 - L_1^3) * (L_2 - L_1) - (L_2^3 - L_1^3) * (L_3 - L_1), \\ DY_1 &= (Y_3 - Y_1) * (L_2 - L_1) - (Y_2 - Y_1) * (L_3 - L_1), \\ DY_2 &= (Y_4 - Y_1) * (L_2 - L_1) - (Y_2 - Y_1) * (L_4 - L_1), \\ DS &= (DSA_1 * DSA_2) - (DSB_1 * DSB_2), \\ DC &= (DY_1 * DSA_2) - (DY_2 * DSB_2), \\ DD &= (DY_2 * DSA_1) - (DY_1 * DSB_1) \end{aligned}$$

$$\Rightarrow D = \frac{DD}{DS}, \quad C = \frac{DC}{DS}$$

$$\begin{aligned} Z_1 &= Y_1 - C * L_1^2 - D * L_1^3, \\ Z_2 &= Y_2 - C * L_2^2 - D * L_2^3 \end{aligned}$$

$$\Rightarrow B = \frac{Z_2 - Z_1}{L_2 - L_1}, \quad A = \frac{(Z_1 * L_2) - (Z_2 * L_1)}{L_2 - L_1}$$

To get A, B,C and D parameters there are a utility that can help to solve the system ([Resolución de ecuaciones lineales.](#))

License

This file is part of ThermistorNTC Library.

ThermistorNTC Library is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

ThermistorNTC Library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with ThermistorNTC Library. If not, see <https://www.gnu.org/licenses/>.

License GPLv3

Authors

Copyright © 2021 Francisco Rafael Reyes Carmona. Contact me:
rafael.reyes.carmona@gmail.com

This project began as a fork of SteinhartHart work by [Andreas Tacke](#). If you want to know more about this work, visit the [Github page](#).

Credits

Thermometer icon at the beginning is from [Flaticon.es](#) designed by [Those Icons](#) and licensed by [free license](#).