

## 21. Temperature Sensors & Recursive Least Squares

### Introduction:

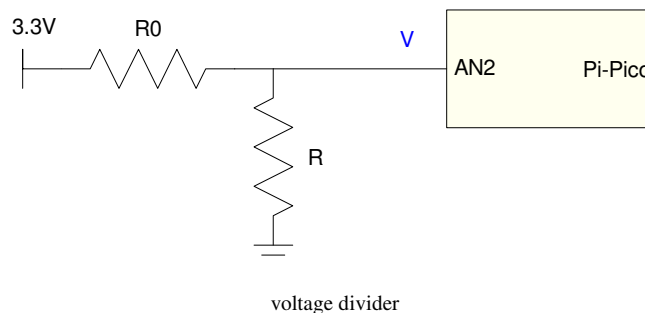
One of the things that is fairly easy to measure is temperature. This lecture looks at measuring temperature using

- Thermistor
- Thermal diode
- Digital temperature

Once we can measure temperature, the thermal time constant of a coffee cup is measured, using both data collection and Matlab as well as recursive least squares on a Pi-Pico

### Resistance:

One of the simplest things to measure is resistance. By using a voltage divider, resistance can be converted to voltage - which the A/D input can read directly.



For this circuit, the resistance - voltage relationship is

$$V = \left( \frac{R}{R+R_0} \right) 3.3V$$

or solving for R in terms of V:

$$R = \left( \frac{V}{3.3-V} \right) R_0$$

By measuring voltage, you can compute the corresponding resistance.

The resolution of this ohmmeter depends upon the resistor values. Assuming  $R_0 = R = 1k$ , the nominal voltage you read is 1.65V

$$V = \left( \frac{1000}{1000+1000} \right) 3.3V = 1.65V$$

The Pi-Pico has a 12-bit A/D, meaning the smallest change in voltage it can detect is 805.9uV

$$dV = \frac{3.3V}{4095} = 805.9\mu V$$

The change in resistance that produces that change in voltage is the resolution of the ohmmeter (the smallest change in resistance you can detect)

$$V = 1.65V + 805.9\mu V$$

$$R = \left( \frac{V}{3.3-V} \right) 1000 = 1000.9777\Omega$$

$$dR = 0.9777\Omega$$

The smallest change in resistance a Pi-Pico can detect is 0.977 Ohms at 1000 Ohms (0.0977%)

## Temperature Sensors: Thermistor

Once you can measure resistance, you can measure pretty much any sensor whose output is resistance. A thermistor is one such sensor.




Image shown is a representation only. Exact specifications should be obtained from the product data sheet.

B57891M0102J000	
DigiKey Part Number	495-2156-ND
Manufacturer	EPCOS - TDK Electronics
Manufacturer Product Number	B57891M0102J000
Description	THERMISTOR NTC 1KOHM 3930K DISC
Manufacturer Standard Lead Time	11 Weeks
Customer Reference	<input type="text"/>
Detailed Description	NTC Thermistor 1k Disc, 3.5mm Dia x 3.5mm W
Datasheet	<a href="#">Datasheet</a>
EDA/CAD Models	<a href="#">B57891M0102J000 Models</a>

1k thermistor from Digikey

A thermistor is essentially a piece of intrinsic silicon. At zero degrees Kelvin, all of the electrons in a silicon crystal are tied up in covalent bonds. This means there are no free electrons to carry current, and hence, infinite resistance.

As temperature goes up, more and more electrons escape their covalent bonds creating charge carriers (both holes and electrons - a topic covered in Electronics.) More charge carriers means less resistance.

In general, the resistance - temperature relationship for a thermistor is of the form:

$$R = \exp\left(a + \frac{b}{K} + \frac{c}{K^2} + \frac{d}{K^3} + \dots\right)$$

where K is the temperature in degrees Kelvin and {a, b, c, d, ...} are constants. If you go with a two-term model

$$R = \exp\left(a + \frac{b}{K}\right)$$

you can rewrite this as

$$R = R_{25} \cdot \exp\left(\frac{B}{T+273} - \frac{B}{298}\right)$$

where

- T is the temperature in degrees C ( $T + 273 = K$ ),

- R25 is the resistance at 25C, and
- B is a constant

If you look up the data sheets for a thermistor, you can find the B parameter

- Digikey Part Number: 495-2156-ND
- R25: 1k
- B25/100: 3930
- Dissipation Factor: 3.5 mW/K

This gives you the model for the thermistor:

$$R = 1000 \cdot \exp\left(\frac{3930}{T+273} - \frac{3930}{298}\right) \Omega$$

Add a voltage divider

$$V = \left(\frac{R}{R+1000}\right) 3.3V$$

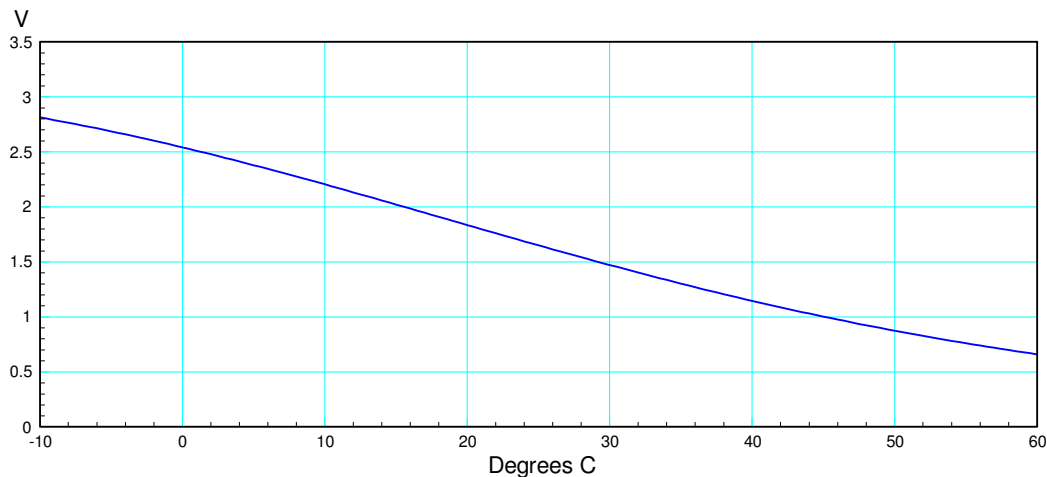
Solving backwards

$$R = \left(\frac{V}{3.3-V}\right) 1000\Omega$$

$$T = \left(\frac{3930}{\ln\left(\frac{R}{1000}\right) + \left(\frac{3930}{298}\right)}\right) - 273$$

Putting this together, as temperature changes, voltage changes - with the resulting relationship shown below for  $-10C < T < +60C$ . What this tells you is

- If you know the temperature, you know the voltage
- If you know the voltage, you know the temperature (it's a temperature sensor)



Temperature - Voltage relationship for a thermistor & voltage divider

The resolution is again the smallest change in temperature you can detect with the 12-bit A/D on the Pi-Pico. Assuming 25C

$$R = 1000\Omega$$

$$V = 1.65V$$

The smallest change in voltage you can detect is 805.9uV.

$$V = 1.65V + 805.9\mu V$$

When  $R = 1000$  Ohms, the smallest change in resistance you can detect is 0.977 Ohms

$$R = 1000.977\Omega$$

The corresponding temperature is

$$T = \left( \frac{3930}{\ln\left(\frac{R}{1000}\right) + \left(\frac{3930}{298}\right)} \right) - 273$$

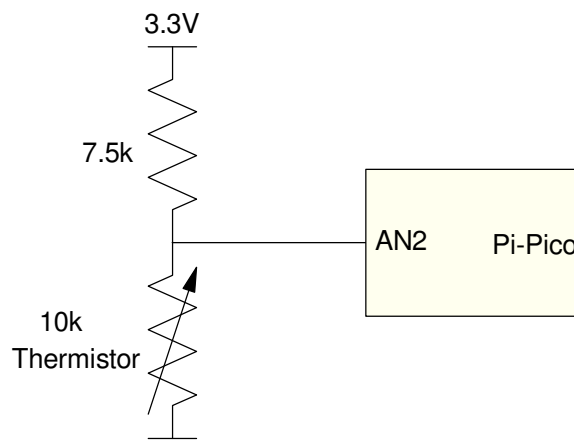
$$T = 24.9779C$$

The difference from 25C is the resolution in temperature:

$$dT = T - 25 = -0.02207C$$

With this setup, a Pi-Pico can measure temperature with a resolution of 0.022 degrees C.

As an example of using a thermistor, measure the temperature of an LED light bulb when turned on. The circuit is simply a voltage divider:



Hardware for measuring the temperature of an LED light using a thermistor

Software simply

- Uses timer interrupts to sample once per second
- Measures the voltage of AN2, and
- Computes the corresponding temperature

```
while(time < 300):
    while(flag == 0):
        pass
    flag = 0

    V = kV * a2d2.read_u16()
    R = V / (3.3-V) * 7500
    Temp = 3930 / ( log(R/10000) + (3930/298) ) - 273

    print(time, V, R, Temp)

    file1.write(str('{: 6.1f}'.format(time)) + " ")
    file1.write(str('{: 7.4f}'.format(Temp)) + " ")
    file1.write("\n")

    time += T
```

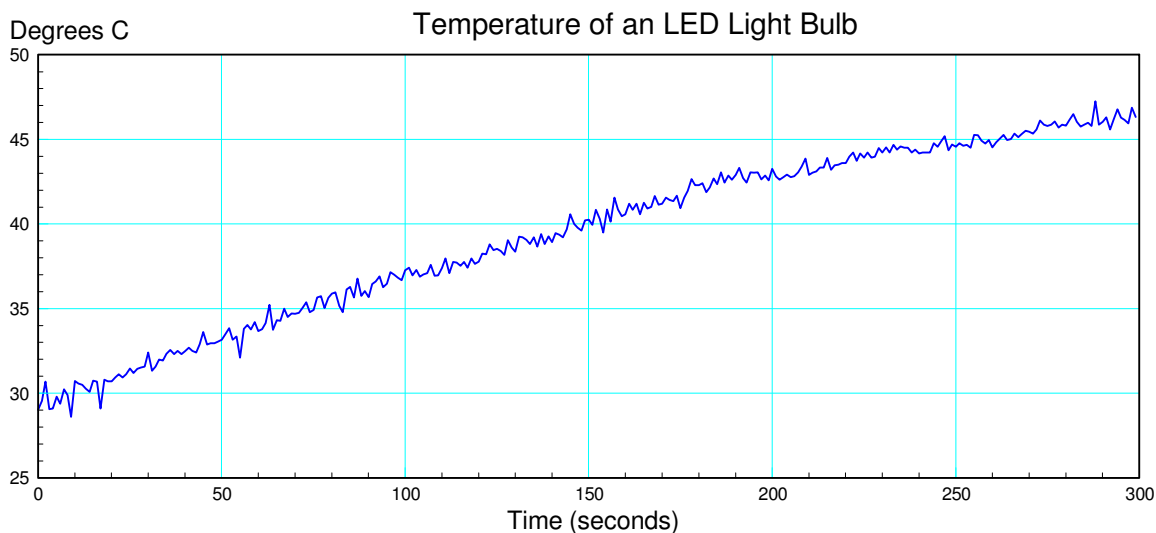
Main loop for 21\_Thermistor.py (complete code on Bison Academy)

The resulting temperature vs. time plot is presented in the following figure. Note from this figure:

- Temperature increases as a decaying exponential. This isn't too surprising: temperature follows the heat equation (coupled first-order differential equations).
- There is considerable noise in the data.


The noise is due to using long wires going from the Pico board to the sensor. These wires act as antennas - picking up stray RF interference. To reduce the noise,

- Twisted pair wires could be used (reduces the area between the wires, the +/- twist cancels most of the RF energy)
- Shielded twisted pair wires would reduce the RF interference even more, or
- Reducing the length of the leads would also help.



The temperature of an LED light bulb when turned on  
The noise on the signal is due to using long separate wires without shielding

## Temperature Sensor: TMP36



**TMP36GT9Z**

DigiKey Part Number: 505-TMP36GT9Z-ND

Manufacturer: Analog Devices Inc.

Manufacturer Product Number: TMP36GT9Z

Description: SENSOR ANALOG -40C-125C TO92-3

Manufacturer Standard Lead Time: 10 Weeks

Customer Reference:

Detailed Description: Temperature Sensor Analog, Local -40°C ~ 125°C 10mV/°C TO-92-3

Datasheet: [Datasheet](#)

EDA/CAD Models: [TMP36GT9Z Models](#)

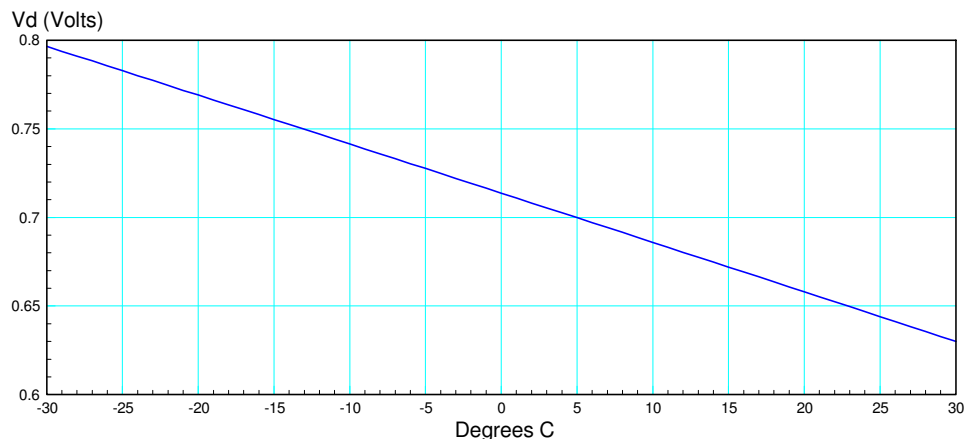
Image shown is a representation only. Exact specifications should be obtained from the product data sheet.

TMP36 Analog Temperature Sensor (Digikey)

Another way to measure temperature is to measure the voltage drop across a diode. From Electronics, the voltage drop across a diode is a function of temperature (ECE 320 lecture #5):

$$V_d = V_T \cdot \ln\left(\frac{N_A N_D}{n_i^2}\right)$$

If you plot the voltage drop across a diode ( $V_d$ ) vs. temperature, you get a line that's almost linear with respect to temperature



Voltage drop across a diode vs. temperature (ECE 320 lecture #5)

A TMP36 is simply a sensor which uses this characteristic of diodes to measure temperature.

From the data sheets for a TMP36 ([www.Digikey.com](http://www.Digikey.com)),

- The operating voltage range is 2.7V to 5.5V (i.e. 3.3V operation works)
- The output at 25C is 750mV (i.e. it's a silicon diode)
- The sensitivity is +10mV / degree C
- Linearity is within 0.5C over a range of -40C to +125C

*note: with diodes, the voltage drops with temperature. The TMP36 has additional circuitry so that the output increases by 10mV/C rather than drops.*

The output voltage should be:

-40C	25C	+125C
100mV	750mV	1.750V

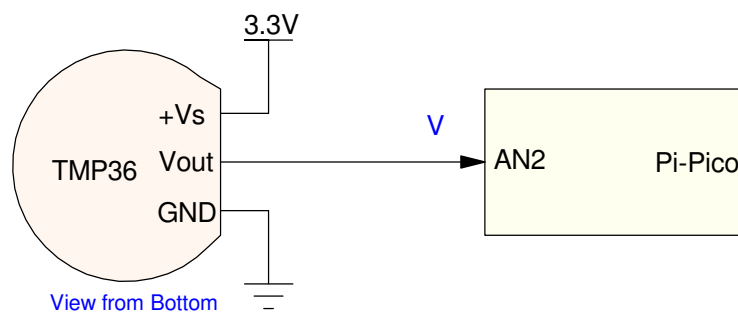
Based upon the voltage, you can then compute the temperature:

$$T = 100V - 50$$

To interface with a Pi-Pico, you can connect it directly to the A/D input. This gives a resolution of 0.08 degrees C:

A/D resolution = 805.9uV

$$\left( \frac{805.9\mu V}{10mV/C} \right) = 0.08059C \quad \text{resolution in degrees C}$$



Interface for a TMP36 temperature sensor to a Pi-Pico

With this sensor, you can measure things, such as the temperature of an incandescent light bulb when turned on. The code is almost identical to the previous code, only with a slight change in how temperature is computed:

```
while(time < 300):
    while(flag == 0):
        pass
    flag = 0

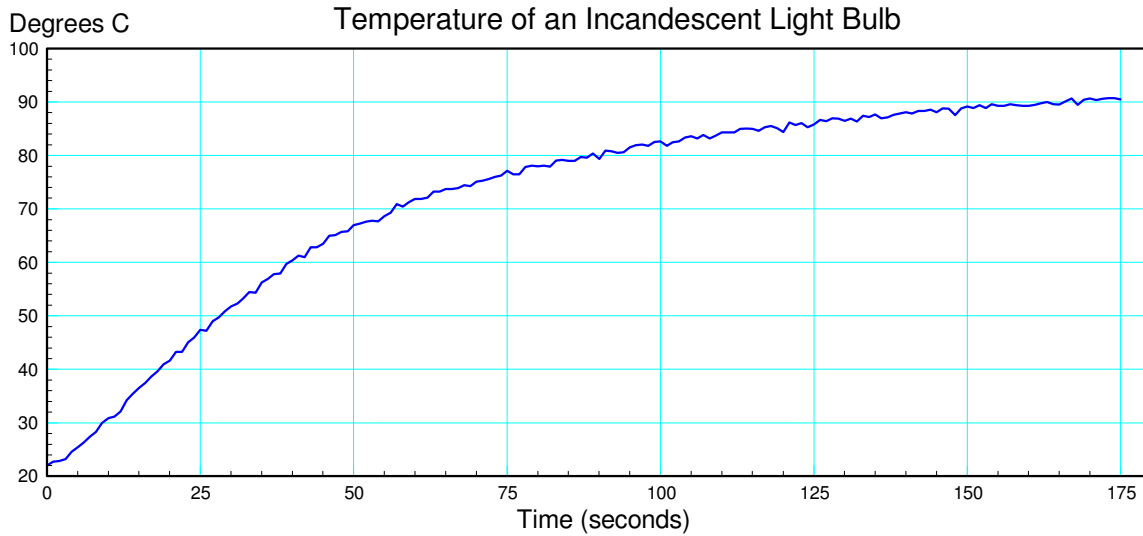
    V = kV * a2d2.read_u16()
    Temp = 100.0*V - 50

    print(time, V, Temp)

    file1.write(str('{: 6.1f}'.format(time)) + " ")
    file1.write(str('{: 7.4f}'.format(Temp)) + " ")
    file1.write("\n")

    time += T
```

The resulting measured temperature vs. time is as follows:



Temperature of an incandescent light bulb, measured with a TMP36 temperature sensor

Note from this figure:

- Incandescent light bulbs get a lot hotter than LED light bulbs
- There is still quite a bit of noise in the data

Again, three long (half-meter) wires were used to connect the sensor to the Pico board. These separate wires act as antennas, picking up stray RF signals. This noise could again be reduced by using shorter wires, using twisted-pair wires, or using shielding.

### DS18B20 Temperature Sensor



DS18B20 Temperature Sensor from Amazon

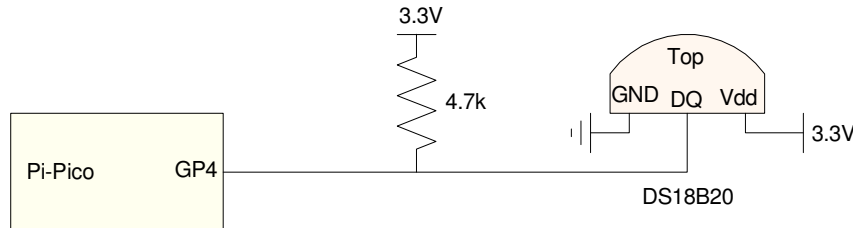
Yet another temperature sensor is the DS18B20. This temperature sensor has a built-in microprocessor along with a one-wire digital interface - making it fairly easy to interface to a microcontroller. With this sensor, you can measure temperature



- Over a range of -55C to +125C
- Using a power supply in the range of 2.5V to 5.5V
- With a resolution of 0.0625C (12-bit)

In addition, each DS18B20 has a unique 64-bit serial code. This means you can connect multiple DS18B20's together and read the temperature of each one separately.

In terms of hardware, just power, ground, and a signal from the Pi-Pico is needed:



Single-wire interface for a DS18B20 thermometer

In terms of software, 1's and 0's are sent and received by the Pi-Pico by switching GP4 between output (write) and input (read) with each bit lasting. To read a temperature,

- The Pi-Pico first sends a Read Time Slot command (0x44), followed by
- A Read Time Slot command
- Wait 750ms for a 12-bit reading, then
- Read temperature as a 16-bit sign-extended 2s compliment number.

Fortunately, the procedure for doing a temperature read has been placed into two libraries, ready for your use (onewire and ds18x20). To read the temperature, the following code works as a bare-bones temperature reading for a single sensor:

```
import onewire, ds18x20
from machine import Pin
from time import sleep, sleep_ms

ds_pin = Pin(4)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))

roms = ds_sensor.scan()
print('Found DS devices: ', roms)

while(1):
    ds_sensor.convert_temp()
    sleep_ms(750)
    Temp = ds_sensor.read_temp(roms[0])
    print(Temp)
```

Basic Python code for reading temperature

<https://randomnerdstutorials.com/raspberry-pi-pico-ds18b20-micropython>

To set the sampling rate to 1.00 second, a timer interrupt can be used:

```
import onewire, ds18x20
from machine import Pin, Timer
from time import sleep, sleep_ms

ds_pin = Pin(4)
ds_sensor = ds18x20.DS18X20(owewire.OneWire(ds_pin))

roms = ds_sensor.scan()
print('Found DS devices: ', roms)

flag = 1
T = 1

def tick(timer):
    global flag
    flag = 1

Time = Timer()
Time.init(freq=1/T, mode=Timer.PERIODIC, callback=tick)

sec = 0

while(sec < 1800):
    while(flag == 0):
        pass
    flag = 0
    sec += T

    ds_sensor.convert_temp()
    sleep_ms(750)
    Temp = ds_sensor.read_temp(roms[0])

    print(sec, Temp)
```

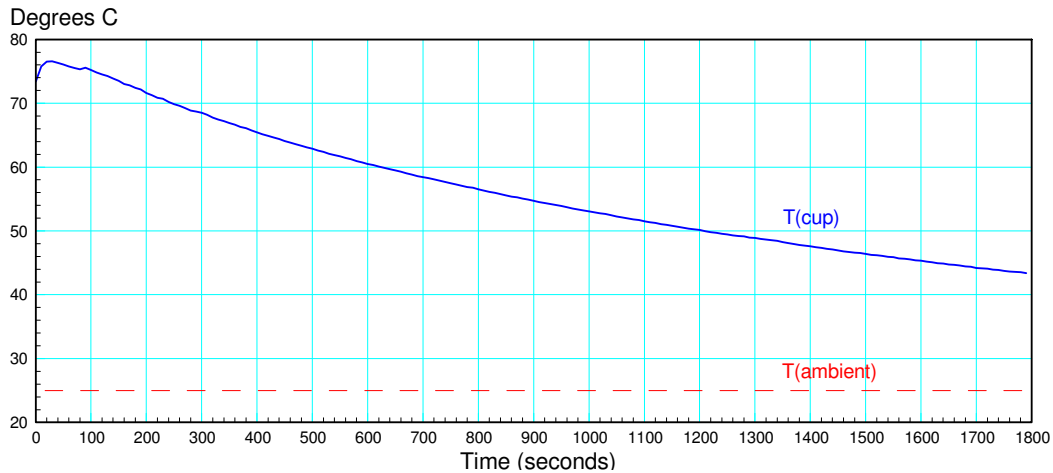
Timer interrupt sets the sampling rate to T seconds (one second here).

Note: The `sleep_ms(750)` command could be removed, making the code more efficient by swapping the `convert_temp()` and `read_temp()` commands. This results in using the 1 second wait (interrupt) to provide the 750+ms wait for the conversion to complete.

As a test, the temperature of a hot cup of water placed in a coffee cup is measured. Note

- The recorded temperature is very clean (very little noise),
- Initially, the temperature has some erratic behavior - probably due to the cup reaching equilibrium
- After 100 seconds, the temperature decays to room temperature ( $T_{amb}$ ) as

$$T = b^* \cdot \exp(at) + T_{amb}$$



Measured temperature of a hot cup of water using a DS18B20 temperature sensor

Note that the noise level with a DS18B20 sensor is *much* less than the previous sensors. This is primarily due to having very short leads: the temperature sensor and microcontroller are all on the same IC. Once the temperature is converted to a digital signal, noise is no longer as much of a problem. This is one of the reasons many sensors are going towards a digital interface.

## Least Squares

Once you can measure temperature, there are several things you can do. One thing is to measure the thermal time constant of your favorite coffee cup.

The thermal time constant can be found by modeling the temperature as

$$T = b^* \cdot e^{at} + T_{amb}$$

or equivalently

$$T - T_{amb} = \exp(at + b)$$

where {a, b} are constants. Taking the log of both sides results in an equation which is linear in t:

$$\ln(T - T_{amb}) = at + b$$

This can be solved using least squares. Placing this in matrix form:

$$\begin{bmatrix} \ln(T_0 - T_{amb}) \\ \ln(T_1 - T_{amb}) \\ \ln(T_2 - T_{amb}) \\ \vdots \end{bmatrix} = \begin{bmatrix} t_0 & 1 \\ t_1 & 1 \\ t_2 & 1 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

or

$$Y = BA$$

The least squares solution for {a, b} is

$$A = (B^T B)^{-1} B^T Y$$

In Matlab:

```
>> Data = [ <paste data from previous code >];
>> t = Data(:,1);
>> T = Data(:,2);
>> Tamb = 24.616;
>> B = [t, t.^0];
>> A = inv(B'*B)*B'*log(T - Tamb)

-3.1563e-004
 4.2630e+000

>> a = -1/A(1)

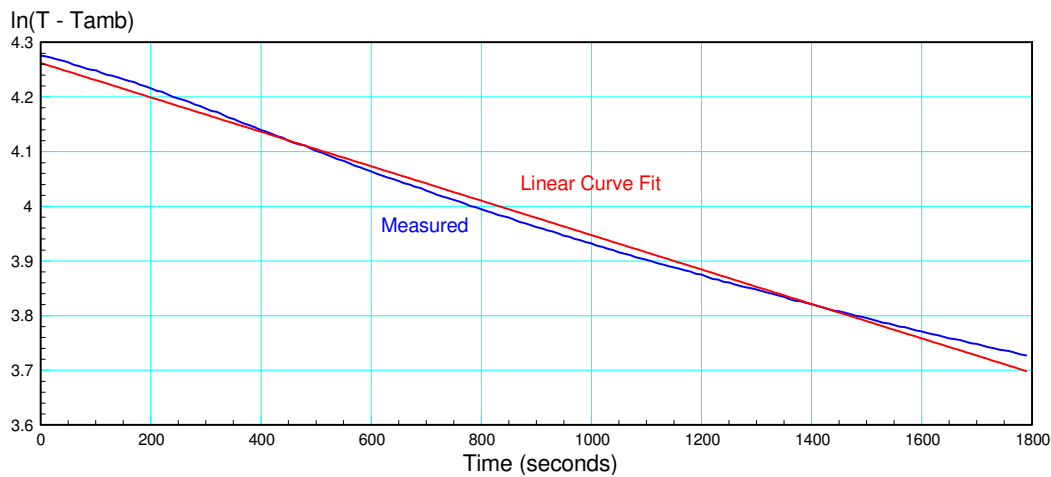
a = 3.1682e+003

>> b = A(2)

b = 4.2630e+000

>> plot(t,T,t,exp(-t/a+b) + Tamb);
```

The thermal time constant is 3168.2 seconds for this coffee cup.



Measured & Curve Fitted Data for the temperature of a coffee cup

You could bypass Matlab and do all of the calculations using Python - but two problems arise:

- With this method, there are a *lot* of computations to do every sample, and
- The initial measurements had errors

To solve these problems,

- Recursive least-squares will be used to simplify calculations (next section), and
- A forgetting factor will be added to weight more recent data more heavily (the following section)

## Recursive Least Squares

Assume you are trying to find a linear curve fit

$$y = ax + b$$

for a set of data. With least squares, you place your data in matrix form (assume four data points for now):

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_0 & 1 \\ x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

The least squares solution is:

$$\begin{bmatrix} a \\ b \end{bmatrix} = \left( \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 & 1 \\ x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

or

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{bmatrix}^{-1} \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix}$$

In Python, you only need to keep track of four terms to create B and Y

$$B = \begin{bmatrix} \sum x_i^2 & \sum x_i \\ \sum x_i & n \end{bmatrix}$$

$$Y = \begin{bmatrix} \sum x_i y_i \\ \sum y_i \end{bmatrix}$$

or equivalently, in a recursive manner:

$$B_i = B_{i-1} + \begin{bmatrix} x_i^2 & x_i \\ x_i & 1 \end{bmatrix}$$

$$Y_i = Y_{i-1} + \begin{bmatrix} x_i y_i \\ y_i \end{bmatrix}$$

Note that you don't need to keep track of the previous data: all you need are the net sums. From these, you can compute the constants, {a, b}

$$A = \begin{bmatrix} a \\ b \end{bmatrix} = B^{-1} Y$$

In Python, the main loop would look something like the following: (note: B is initially non-zero to avoid an error in the matrix inverse routine).

```

x = 0
Tamb = 19.38
B = [[0.01, 0], [0, 0.01]]
Y = [[0], [0]]
while(1):
    while(flag == 0):
        pass
    flag = 0

    ds_sensor.convert_temp()
    sleep_ms(750)
    Temp = ds_sensor.read_temp(roms[0])

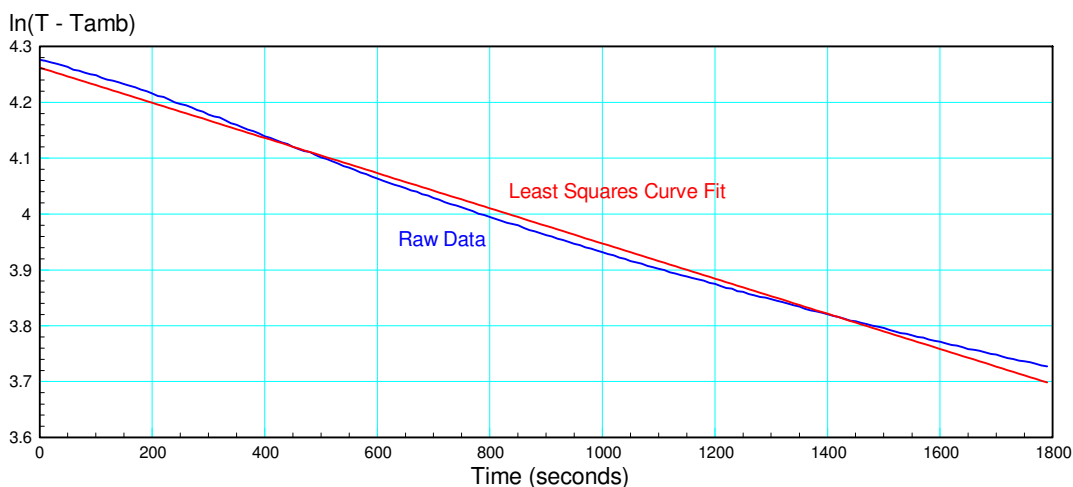
    x += 1
    y = log(Temp - Tamb)

    matrix.add(B, [[x*x, x], [x, 1]])
    matrix.add(Y, [[x*y], [y]])
    Bi = matrix.inv(B)
    A = matrix.mult(Bi, Y)
    a = A[0][0]
    b = A[1][0]
    print(x, a, b)

```

Recursive least squares algorithm to find the slope (1 / thermal time constant)

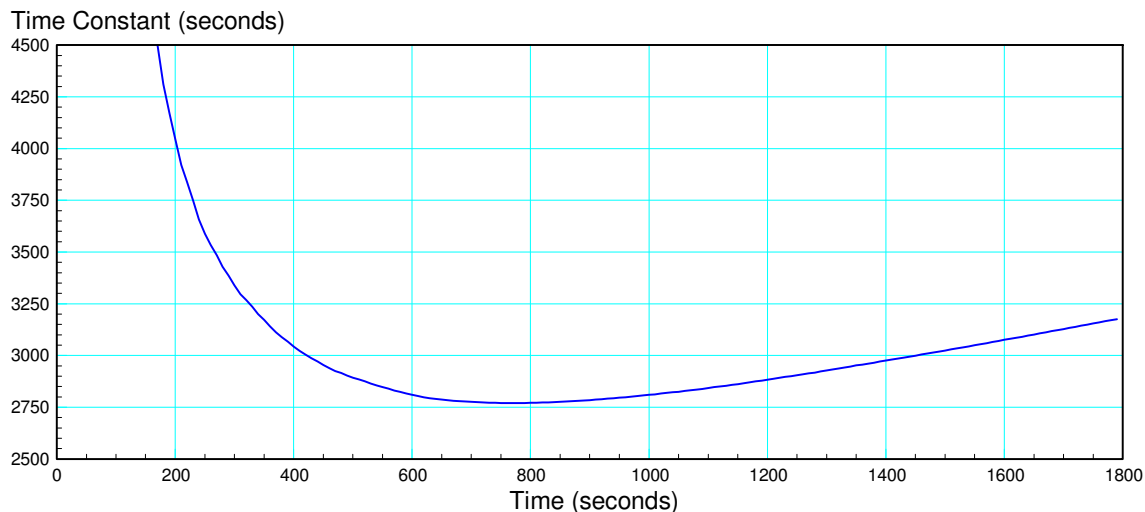
The resulting linear curve fit is as follows:



Least squares curve fit with a weighting of 1.000 for all of the data

Note the following:

- The data does not follow a line. This suggests the system is actually nonlinear (Evaporation adds additional heating at the start. Adding a lid would reduce this effect and make the system more linear.)
- As you collect more and more data, the slope changes, meaning the least-squares curve fit changes. This shows up in the estimate of the thermal time constant:  $1/a$



Estimate of the thermal time constant  $(-1/a)$  with a weighting of 1.000 for all of the data

## Recursive Least Squares with a Moving Window

One problem with the previous solution is that *all* of the data has an equal weighting. This can be good: more data means a better estimate. This can also be bad: the weightings on the current data drops as more and more data comes in. Likewise, if the system is time-varying, the changes will eventually be ignored. To alleviate this problem, more recent data can be weighted more heavily than older data.

One way to use more recent data is to limit the window over which the least-squares takes place. For example, if you only use the last 100 data points (last 100 seconds), you can see the changes in the slope / time constant.

In code

- Create a buffer which saves the last 100 data points
- Recompute B and Y using these last 100 data points each sample,
- From B and Y, recompute the least-squares curve fit each sample.

A circular stack saves time (so you don't have to constantly push data onto a stack each sample). This gives a more efficient way to compute B and A

$$B_i = B_{i-1} + \begin{bmatrix} x_i^2 & x_i \\ x_i & 1 \end{bmatrix} - \begin{bmatrix} x_{i-100}^2 & x_{i-100} \\ x_{i-100} & 1 \end{bmatrix}$$

$$Y_i = Y_{i-1} + \begin{bmatrix} x_i y_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_{i-100} y_{i-100} \\ y_{i-100} \end{bmatrix}$$

```

X = [0]*100
Y = [0]*100

Tamb = 19.38
B = [[0,0],[0,0]]
Y = [[0],[0]]
ptr = 0
time = 0
while(1):
    while(flag == 0):
        pass
    flag = 0
    ptr = (ptr + 1) % 100
    time += 1

    ds_sensor.convert_temp()
    sleep_ms(750)
    Temp = ds_sensor.read_temp(roms[0])

    x = X[ptr]
    Y = Y[ptr]

    matrix.subtract(B, [[x*x, x], [x, 1]])
    matrix.subtract(Y, [[x*y], [y]])

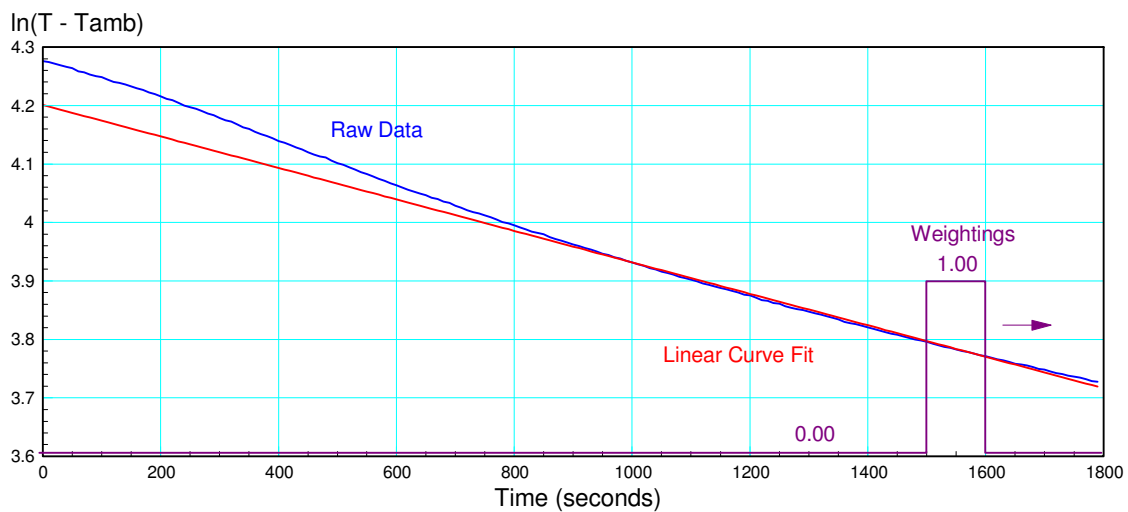
    X[ptr] = x = time
    Y[ptr] = y = log(Temp - Tamb)

    matrix.add(B, [[x*x, x], [x, 1]])
    matrix.add(Y, [[x*y], [y]])

    if(time >= 100):
        Bi = matrix.inv(B)
        A = matrix.mult(Bi, Y)
        a = A[0][0]
        b = A[1][0]
        print(x, a, b)

```

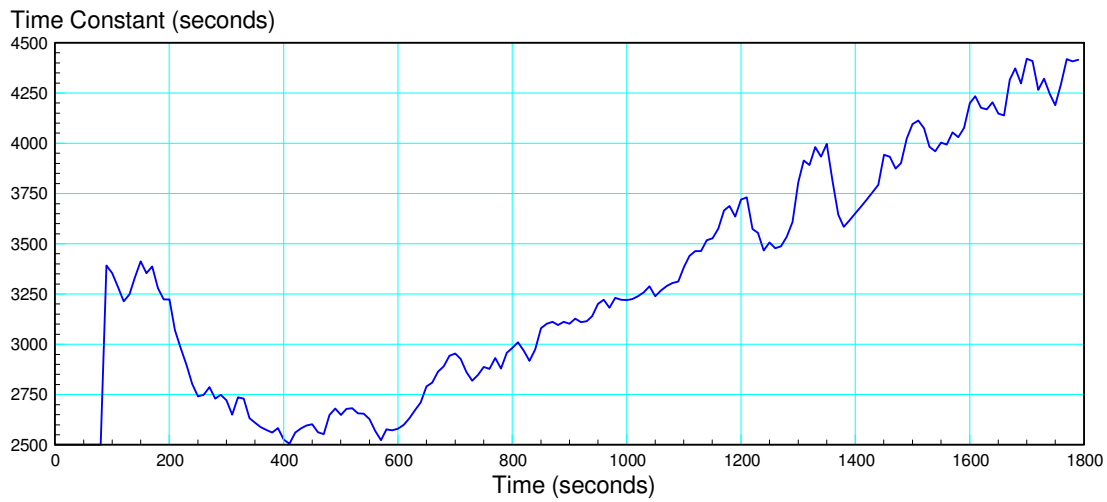
Python routine for using a moving window along with least squares curve fitting



In order to weight more recent data more heavily, only the last 100 data points can be used



The resulting estimate of the time constant is somewhat better: it tracks the changes in the thermal time constant. It's also somewhat worse: with less data you get more noise.



Least squares estimate of the thermal time constant using only 100 seconds worth of data (moving window)

## Recursive Least Squares with a Forgetting Factor

A similar scheme uses an exponential weighting factor:

$$\text{Weight}(k) = \alpha^k \quad 0 < \alpha < 1$$

where 'k' is how many samples in the past the data was collected and alpha is a weighting factor. This actually simplifies the code in that you no longer need a buffer storing all of the old data. All you need is the resulting B and Y matrices:

The long way to compute B and Y are:

$$B_k = \sum_{n=0}^k \alpha^{k-n} \begin{bmatrix} x_n^2 & x_n \\ x_n & 1 \end{bmatrix}$$

$$Y_k = \sum_{n=0}^k \alpha^{k-n} \begin{bmatrix} x_n y_n \\ y_n \end{bmatrix}$$

A shorter, recursive way to compute B and Y are

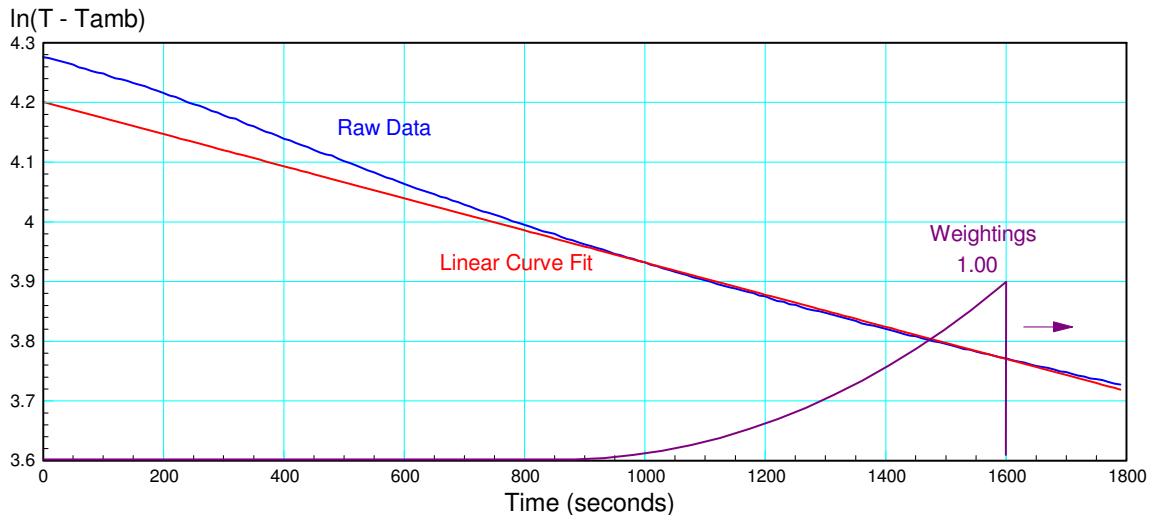
$$B_k = \alpha B_{k-1} + \begin{bmatrix} x_k^2 & x_k \\ x_k & 1 \end{bmatrix}$$

$$Y_k = \alpha Y_{k-1} + \begin{bmatrix} x_k y_k \\ y_k \end{bmatrix}$$

The constants are then

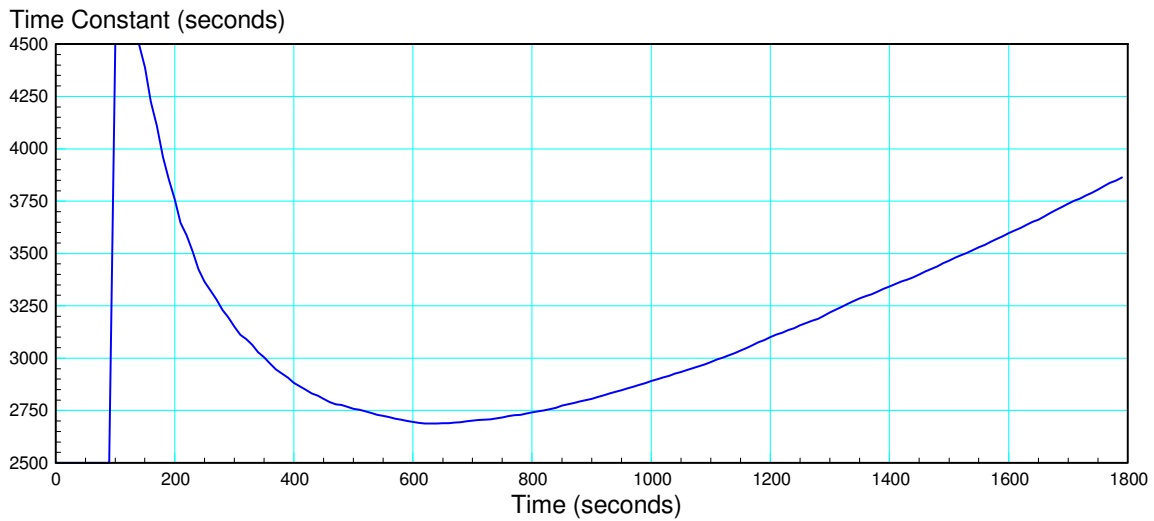
$$A = \begin{bmatrix} a \\ b \end{bmatrix} = B^{-1}Y$$

For example, making the forgetting factor 0.995 (old data is discarded by 0.5% per sample) results in data which matches the more recent data (the weighting of current data no longer goes to zero):



Least squares curve fit with exponential weighting and a forgetting factor of 0.995

and the estimate of the thermal time constant is less noisy (due to using more data in the curve fit)



Least squares estimate of the thermal time constant using exponential weighting and a forgetting factor of 0.995

---

**Summary:**

Temperature is pretty easy to measure.

- Thermistors and thermal diodes output an analog signal, which can be read by the 12-bit A/D on the Pi-Pico. This along with some computations allow you to determine the temperature.
- Digital temperature sensors allow you to read temperature without the need of an A/D conversion. These also have the advantage of returning temperature rather than a raw signal which needs to be converted to temperature.

Once you can measure temperature, you can do all sorts of things, like measure the thermal time constant of a coffee cup as well as other things.

**References**

## Pi-Pico and MicroPython

- [https://github.com/geekpi/pico\\_breakboard\\_kit](https://github.com/geekpi/pico_breakboard_kit)
- [https://micropython.org/download/RPI\\_PICO/](https://micropython.org/download/RPI_PICO/)
- <https://learn.pimoroni.com/article/getting-started-with-pico>
- <https://www.w3schools.com/python/default.asp>
- <https://docs.micropython.org/en/latest/pyboard/tutorial/index.html>
- <https://docs.micropython.org/en/latest/library/index.html>
- <https://www.fredscave.com/02-about.html>

## Pi-Pico Breadboard Kit

- <https://wiki.52pi.com/index.php?title=EP-0172>

## Other

- <https://docs.sunfounder.com/projects/sensorkit-v2-pi/en/latest/>
- <https://electrocredible.com/raspberry-pi-pico-external-interrupts-button-micropython/>
- <https://peppe8o.com/adding-external-modules-to-micropython-with-raspberry-pi-pico/>
- <https://randomnerdtutorials.com/projects-raspberry-pi-pico/>
- <https://randomnerdtutorials.com/projects-esp32-esp8266-micropython/>