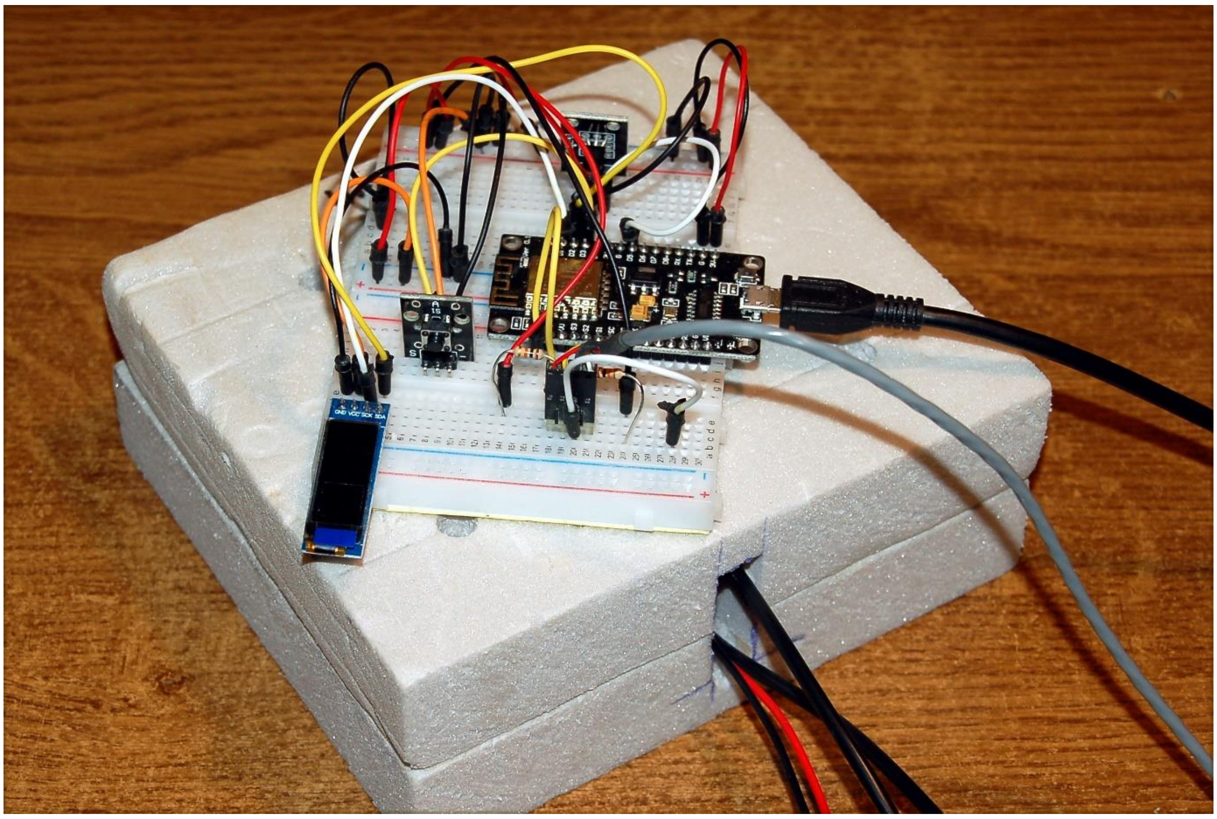


Abbildung 1: Wattmeter_Aufbau

They sneak into the room secretly and unrecognized. They can be welcome, but they can also appear superfluous. In summer we prefer to keep them outside, in winter we are happy to have a certain amount of them in the room. We are talking about calories, or to put it in the current physical language, joules (pronounced: "dschuul"). Welcome to the Energy Detectives Club. Today we are

On the trail of calories

A joule (= 4.16 Cal) is the unit of energy and can be expressed as a derived quantity, depending on the area of application, by other units, electrical 1J = 1VAs, mechanical 1J = 1Nm. Today we are going to make a simple circuit with which we can observe and measure the thermal energy on its way from here to there. Because the total energy in a room or a vessel is difficult to determine because too many parameters play a role. But on the other hand, we can record energy that is on the move, through radiation or thermal contact, quite well. In order for energy to be measurable for us, we need a device that we can insert into the path that the energy takes. For a better understanding one can take a look at electricity and make a comparison.

Electric charges (electrons) moving through a conductor are referred to as electric current. If they collect on a surface, we speak of their charge or charge Q for short. The total charge Q that sits on a body cannot be measured directly, but the current I and the time t can be measured while the charges are through hike up or down a ladder to the surface. We measure the charge currents with an ammeter, which we insert directly into the circuit, the path taken by the charges; it records the amount of charge Q that passes through the conductor cross-section A at the measuring point per unit of time.

$$\boxed{I = \frac{\text{Ladungsmenge } Q}{\text{Zeit } t}}$$

Abbildung 2: Definition Stromstärke

$$\boxed{[\,I\,] = \frac{1\ \mathrm{C}}{1\ \mathrm{s}} = 1\ \mathrm{A}}$$

Abbildung 3: Einheit Stromstärke

Today we will build a device to measure energy flows; it records the amount of energy in joules that passes through cross-section A of the measuring point per unit of time. This results in a new size. In general, the quotient of energy or work by time is defined as power P.

$$\boxed{P = \frac{\text{Wärmemenge } W_{th}}{\text{Zeit } t}}$$

Abbildung 4: Definition Wärmestrom - Wärmeleistung

$$\boxed{[\,P\,]= \frac{1\ \mathrm{J}}{1\ \mathrm{s}} = 1\ \text{Watt} = 1\mathrm{W}}$$

Abbildung 5: Einheit Wärmestrom – Wärmeleistung

And just as an ammeter is built into a circuit, we will inject our thermal watt meter into energy flows. If we then measure the time, we can calculate how many joules were emitted or absorbed by a body in total. Another analog would be the swimming pool, the amount of water, the supply line and the water meter.

What we need for the wattmeter

Die Hardware

1*	0,91 Zoll OLED I2C Display 128 x 32 Pixel
1*	NodeMCU Lua Amica Modul V2 ESP8266 ESP-12F
1	TEC1-12706 Thermoelektischer Wandler Das 5-er-Bundel ist sehr günstig in Hinblick auf den Folgebeitrag
1	KY-004 Taster Modul Sensor Taste 3er Bundle ist günstiger als 2 Einzeltaster
1	Mini Breadboard 400 Pin mit 4 Stromschienen für Jumper Kabel 5er-Bundel ist günstiger als Einzelstück oder 3er-Bundle
1	DS18B20 digitaler Temperatursensor TO92-55°C - +125°C oder besser gleich mit für den Folgebeitrag bestellen 1M Kabel DS18B20 digitaler Edelstahl Temperatursensor Temperaturfühler, wasserdicht
3	Widerstand 10kOhm
div.	Jumperkabel
1	USB-A-Stecker auf Micro-Stecker
Reste	von Stiftreihen für DS18B20 und Peltier-Element
1	Netzteil mit 5..15V / >=3A zum Beispiel den DC-DC-Wandler aus der vorigen Blogfolge

In addition to the hardware, some software is needed. On the one hand the MicroPython firmware for the ESP8266 in general and of course the programs to calibrate the controller as a wattmeter and then to measure heat flows in particular.

Our device will be able to record heat flows with a resolution of approx. 25mW. This could be improved by using a preamplifier or an ESP32.

Software

Used Software:

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft
micropython-font-to-py

Verwendete Firmware:

MicropythonFirmware
Bitte eine Stable-Version aussuchen

MicroPython-Programme

Module:
button.py zur Bedienung von Tasten
charset.py die Steuerklasse für größere Zeichensätze
geometer_24.py stellt den Zeichensatz geometer.ttf in Punkt24 auf dem OLED dar
oled.py, die Klasse OLED enthält Methoden zur bequemen Ansteuerung von OLED-Displays
ssd1306.py ist der Low-Level Treiber für das verwendete Display
Anwendungen:
eichen.py zur Ermittlung der Betriebsdaten des Peltierelements
wattmeter.py Messprogramm zur Erfassung von Wärmeströmen
_font2py.rar Paket zur Erzeugung von Pixelzeichensätzen aus TTF-Fonts

MicroPython - Language - Modules and Programs

You can find detailed instructions for installing Thonny here. There is also a description of how the Micropython firmware is burned onto the ESP chip

MicroPython is an interpreter language. The main difference to the Arduino IDE, where you always and exclusively flash entire programs, is that you only have to flash the MicroPython firmware once at the beginning on the ESP32 before the controller understands MicroPython instructions. You can use Thonny, µPyCraft or esptool.py for this. I have described the process for Thonny here.

As soon as the firmware is flashed, you can have a casual conversation with your controller, test individual commands and immediately see the answer without first having to compile and transfer an entire program. This is exactly what bothers me about the Arduino IDE. You simply save an enormous amount of time if you can do simple tests of the syntax and hardware through to trying out and refining functions and entire program parts via the command line before you knit a program out of it. For this purpose I also like to create small test programs over and over again. As a kind of macro, they combine recurring commands. From such program fragments, entire applications can develop.

Autostart

If the program is to start autonomously when the controller is switched on, copy the program text into a newly created blank file. Save this file under boot.py in the workspace and upload it to the ESP chip. The program starts automatically the next time it is reset or switched on.

Testing programs

Programs are started manually from the current editor window in the Thonny IDE using the F5 key. This is faster than clicking the start button or using the Run menu. Only the modules used in the program must be in the flash of the ESP8266.

In between, Arduino IDE again?

If you later want to use the controller together with the Arduino IDE again, simply flash the program in the usual way. However, the ESP32 / ESP8266 then forgot that it ever spoke MicroPython. Conversely, every Espressif chip that contains a compiled program from the Arduino IDE or the AT firmware or LUA or ... can easily be provided with the MicroPython firmware. The process is always as described here.

Structure of the circuit and the periphery

We mentioned earlier that our thermal wattmeter can be compared with an ammeter from the electrical sector. The way of working is also directly comparable. The ohmic resistance formula states that an electric current flows through a resistor when a voltage is applied. A digital ammeter uses the reverse of this theorem. A voltage drops across a resistor when a current flows through it.

$$\boxed{U = I \bullet R}$$

Abbildung 6: Spannung am Widerstand

This voltage is directly proportional to the current strength. The resistance in the circuit is therefore the current intensity sensor, which makes the current intensity measurable indirectly via the voltage. The measuring device has the task of converting the voltage at the sensor into a current value based on knowledge of the resistance value and displaying it as such.

Our heat flow meter works the same way, only with a different effect, a different sensor. We use the Seebeck effect of a thermoelectric converter, also known as the Peltier element. These components are primarily used to convert electrical energy into a heat flow via the Peltier effect. We use this in the third post for the beverage cooler. This Peltier effect can also be reversed, resulting in the Seebeck effect.

If a heat flow is conducted through a Peltier element, a voltage is generated at its electrical connections that is directly proportional to the temperature difference on the surfaces and thus also to the heat flow. We use this Seebeck effect for our heat flow meter. The physical unit of a heat flow results from heat work / time as heat output. It applies:

$$\boxed{\text{(I)} \quad P_{th} = k \bullet U_{th}}$$

Abbildung 7: Thermospannung

The idea for this came from a contribution by Werner B. Schneider and H. Dittmann, published in Volume 4, Paths in Physics Didactics, Erlangen 1998. There it is also described how the calibration process can take place, because first we have to get the k from the Determine equation (I) so that the ESP8266 can then convert the voltage into a heat flow and display it.

But first we need a circuit and a few more information about its environment. You can download the circuit diagram as a PDF file in DIN A4.

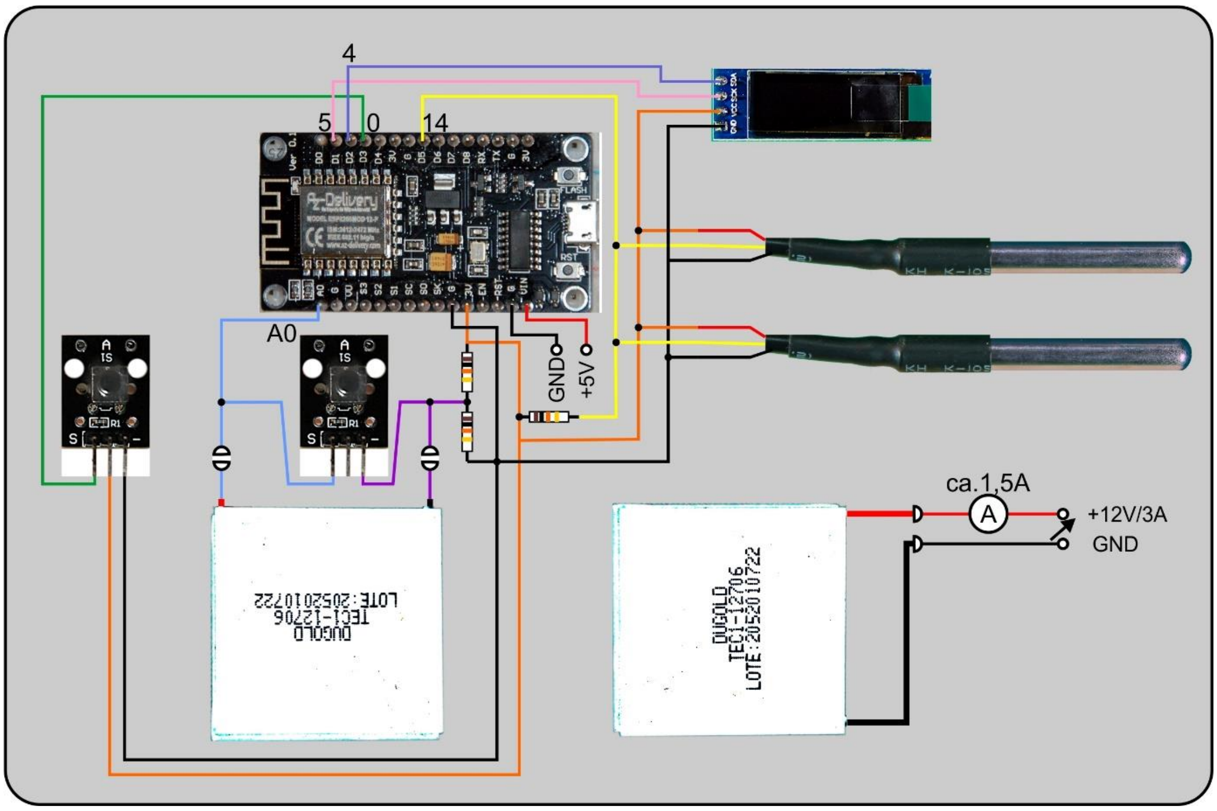


Abbildung 8: Wattmeter_Schematic

The heat conduction equation behind the Seebeck formula primarily uses the temperature difference between the two sides of the Peltier element instead of the thermal voltage.

$$\text{(II)}\quad Q = \lambda \bullet \frac{A}{l} \bullet t \bullet \Delta T$$

Abbildung 9: Wärmeleitungsgleichung

The thermal voltage itself is also directly proportional to the temperature difference.

$$\text{(III)}\quad U_{th} = \beta \bullet \Delta T$$

Abbildung 10:Thermospannung und Temperaturdifferenz

β and λ are material constants. With the area A and the thickness l of the Peltier element and by rearranging and combining the constants, equation (I) is finally obtained.

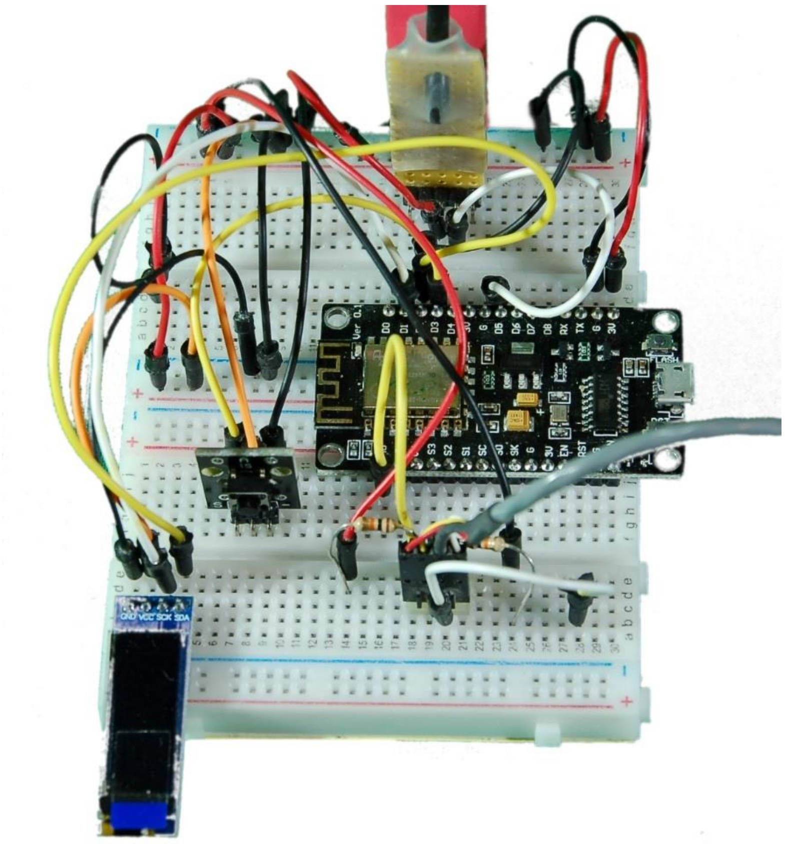


Abbildung 11: Wattmeter-Aufbau_calibrieren

In our setup, we measure the temperatures of two aluminum cuboids, they represent our heat reservoirs, which we bring to temperatures slightly above and below room temperature at the beginning of the calibration. This prevents excessive heat exchange with the environment, which would lead to measurement errors. In addition, we wrap the two bodies between which the Peltier element lies in foam padding. The dowels secure the cover and thus also the upper cuboid and the thermocouple against slipping.

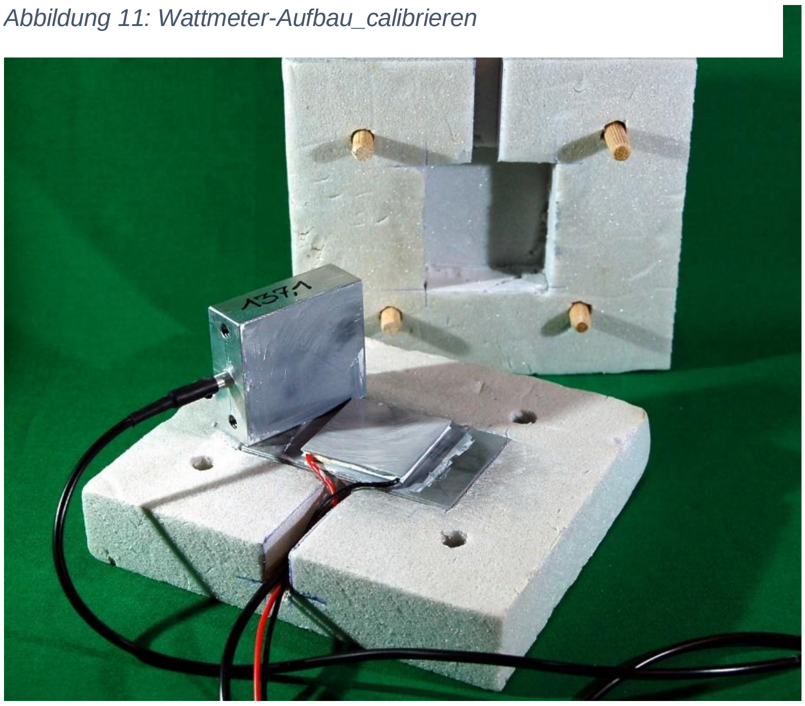


Abbildung 12: Peltierelement mit DS18B20-Fühlern

The insulating material is made of rigid foam. The depressions were made with a Proxxon and a cylinder milling cutter on the drill stand and the edges were reworked with a cutter. The aluminum cuboids were drilled so deep with a 6mm precision drill that the top of the cylinder with the DS18B20 comes to rest in the middle. A little silicone thermal paste then ensures good thermal contact between the sensor, aluminum block and Peltier element. The mass mhot (= 137.1g, upper cuboid) and mcold (= 172.8g, lower cuboid in the foam bed) as well as the c-value of aluminum cAlu = 0.896 J / (g K) must be known. Your mass values will be different from mine for sure. My cuboids are 20mm thick. You need to cover the element well.

The calibration of the thermocouple as a wattmeter sensor

Let us now start a series of measurements, the results of which provide us with the coveted proportionality factor k from formula (I) and the Seebeck coefficient α. We use the program eichen.py for this. All of the above modules must be in the flash of the ESP8266.

We determine the assignment of the DS18B20 in advance. The sensors are automatically recognized by the program. We start without calibration and check by touching a sensor with the hand whether the left value in the list in the terminal increases. This sensor is placed in the upper cuboid, the other in the lower one. This will make the assignment easier later. The upper cuboid is considered to be the warmer one for the later measurement.

```
# thermometer.py
# Author: J. Grzesina
# Rev: 1.0
# Stand: 2021-06-22
# ****************************************************************
from machine import Pin, I2C, ADC, Timer
from time import sleep, ticks_ms,sleep_ms
from onewire import OneWire
from ds18x20 import DS18X20
from oled import OLED
from button import BUTTON8266, BUTTONS

# Pintranslator fuer ESP8266-Boards
# LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                 SC SD

adc=ADC(0)

SCL=Pin(5)
SDA=Pin(4)
i2c=I2C(-1,SCL,SDA)
d=OLED(i2c,128,32)
d.clearAll()

ds_pin = Pin(14)     # D5@esp8266
ds = DS18X20(OneWire(ds_pin))
chips = ds.scan() # Liste von bytearrays
numberOfChips = len(chips)
print('Found DS devices: ')
for chip in chips:
    print(chip)

t0,t1=0,0
def los_messen(tim):
    global timerFlag
    timerFlag=True
u=0
```

```
u0=0
U=""
def getTh(n=5):
    s=0
    for i in range(n):
        s+=adc.read()
    s=s//n
    u=(int(s/1023*3.0*10000))/10  # Spannung in mV
    return u

taste = BUTTON8266(0,invert=True) # D3@esp8266
k=BUTTONS()

d.clearAll()
d.writeAt("THERMOSENSOREN",0,0,False)
d.writeAt("KALIBRIEREN",0,1,False)
d.writeAt(">>> Taste",0,2)

act=k.waitForTouch(taste,3)
if act:
    sleep(1)
    k.waitForTouch(taste,6)
    d.clearAll()
    d.writeAt("Sensor A + B in",0,0,False)
    d.writeAt("Wasser tauchen",0,1,False)
    d.writeAt(">>> Taste",0,2)
    sleep(1)
    k.waitForTouch(taste,6)
    d.clearAll()
    d.writeAt("Temperaturen",0,0,False)
    d.writeAt("konstant?",0,1,False)
    d.writeAt(">>> Taste",0,2)
    sleep(1)
    while taste.tpin.value()==1:
        ds.convert_temp()
        u0=getTh(20)
        sleep(1)
        t0 = (int((ds.read_temp(chips[0]))*100))/100
        t1 = (int((ds.read_temp(chips[1]))*100))/100
        print(t0,t1)
        d.clearAll()
        d.writeAt("A {}".format(t0),0,0,False)
        d.writeAt("B {}".format(t1),0,1,False)
        d.writeAt("U {}".format(u0),0,2)
        sleep(2)
    dt=t0-t1
    d.clearAll()
    d.writeAt("Calibrat. done",0,0,False)
    d.writeAt("Release key!".format(dt),0,1,False)
    d.writeAt("U0={} dT={}".format(u0,dt),0,2)
    sleep(3)
    f=open("calibration.txt","w")
    f.write(str(dt)+"\n")
    f.write(str(u0)+"\n")
    f.close()
    f=None
else:
    f=open("calibration.txt","r")
    print("Position",f.tell())
    dt=float(f.readline())
    u0=float(f.readline())
    f.close()
    f=None

print("dt = {}".format(dt))
print("u0 = {}".format(u0))

sleep(1)
T=Timer(0)
timerFlag= True
intervall=10000
T.init(mode=Timer.PERIODIC,period=intervall, \
       callback=los_messen)
firstRun=True
start=ticks_ms()
while taste.tpin.value() == 1:
    if timerFlag==True:
        timerFlag=False
        ds.convert_temp()
        zeit=ticks_ms()-start
        s=getTh(20)
        u=s-u0  # Spannung in mV
        uString=(str(u)).replace(".",",")
        sleep_ms(750)
        t0 =  (int((ds.read_temp(chips[0]))*100))/100
        t1 = dt+(int((ds.read_temp(chips[1]))*100))/100
        dT=t0-t1
        f="{:.2f}"
        dT = f.format(dT)
        t0 = f.format(t0)
        t1 = f.format(t1)
        dTString = dT.replace(".",",")
        t0String = t0.replace(".",",")
        t1String = t1.replace(".",",")
        f="{:>8};{:>6};{:>6};{:>6};{:>6}"

print(f.format(str(zeit),t0String,t1String,dTString,uString))
        d.clearAll()
        d.writeAt("A {}".format(t0),0,0,False)
        d.writeAt("B {}".format(t1),0,1,False)
        d.writeAt("U {}".format(u),0,2)
T.deinit()
d.clearAll()
d.writeAt("PROG DONE",0,0)
```

The program does not contain any exotic code parts. The time control takes place via a timer interrupt whose interrupt service routine (ISR) sets a flag. This flag tells the main program that a measurement is to be carried out. The IRQ runs in continuous mode until the program is aborted with the button on GPIO0 (cancel button).

At the start, the two aluminum cuboids should have the same temperature. So that both positive and negative voltages can be measured on the thermocouple, the negative connection is not connected to GND but to the middle of the voltage divider made up of two 10kΩ resistors. The center voltage must be determined so that the ESP8266 can subtract it from the measured value later during the measurements. We are only interested in the voltage at the Peltier element.

Nevertheless, despite sufficient waiting, the two DS18B20 sensors result in slightly different temperature values and fluctuations in the ADC values due to noise on the line. To mitigate this effect, the ADC scans the input 20 times and calculates an average value from it. This happens in the getTh () function.

A calibration can therefore be carried out before the measurements, which the display provides information about. The calibration data are written to a file that is read in during further measurement processes. The thermocouple does not have to be connected for calibration. But during the process, the button that is parallel to it must be pressed, or we short-circuit the path with a jumper cable. When the values in the display no longer change significantly, we press the button on GPIO0.

What happens next?

The thermocouple must now be disconnected from the ESP8266 in any case, otherwise the controller will die! The labeling of the Peltier element should point downwards. Then we set a value between 5V and 6V on the power supply unit and switch off the power supply to the converter. We connect the positive pole of the converter to the red connection of the thermocouple, the negative pole to the black one.

The eichen.py program is started. We wait until the temperatures of the aluminum blocks and the voltage are shown in the display and in the terminal. The temperature value of the lower block is noted.

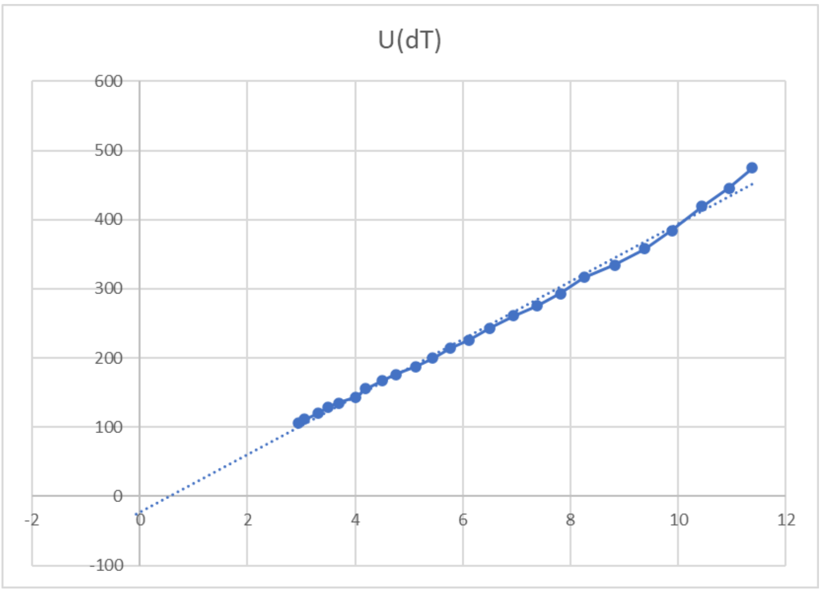
The power supply to the converter is now switched on. If the temperatures have risen and fallen by a good 5 degrees Celsius, we switch off the power supply and disconnect the connection between the converter and the thermocouple. We stop the program with the cancel button and restart it. The thermocouple is connected to the ESP8266. Every 10 seconds, line by line is output until the temperature of the cold cube has just reached the starting temperature again. The measurement is stopped.

Here is my table of measured values, which has been expanded to include the deltaT column, which shows the difference between the warm and cold cuboid. We use the values marked in color for the calculation.

Zeit in	oben	unten		
Millisekunden	Thot	Tcold	deltaT	U
130003	29,43	18,06	11,37	475,1

140003	29,25	18,31	10,94	445,7
150003	28,93	18,49	10,44	419,3
160003	28,62	18,74	9,88	384,1
170003	28,31	18,93	9,38	357,8
180003	28	19,18	8,82	334,3
190003	27,68	19,43	8,25	316,7
200003	27,43	19,62	7,81	293,2
210003	27,18	19,81	7,37	275,6
220003	26,93	19,99	6,94	261
230003	26,68	20,18	6,5	243,4
240003	26,43	20,31	6,12	225,8
250003	26,25	20,49	5,76	214,1
260003	26,06	20,62	5,44	199,4
270003	25,87	20,74	5,13	187,7
280003	25,68	20,93	4,75	175,9
290003	25,56	21,06	4,5	167,1
300003	25,37	21,18	4,19	155,4
310003	25,25	21,24	4,01	143,7
320003	25,06	21,37	3,69	134,9
330003	24,93	21,43	3,5	129
340003	24,87	21,56	3,31	120,2
350003	24,68	21,62	3,06	111,4
360003	24,62	21,68	2,94	105,6

Die Grafik zeigt einen linearen Zusammenhang auf, der eigentlich rückläufig zu betrachten ist.



Now has to be calculated.

The heat capacity (aka heat capacity) of the cuboids
C_{hot} = c_{Alu} • m_{hot} = 0,896 J/(gK) • 137,1g = 122,84 J/K

$C_{cold} = c_{Alu} \cdot m_{cold}$ = 0,896 J/(gK) • 172,8g = 154,83 J/K

Hier sind die in der Grafik markierten Zeilen zusammengefasst.

		oben	unten	
	t in s	T(hot) in °C	T(cold) in °C	U in mV
	130003	29,43	18,06	475,1
	360003	24,62	21,68	105,6
Differenz	230	4,81	3,62	369,5

The mean thermoelectric voltage
U_{mid} = (475,1 + 105,6)/2 mV= 290,35 mV = 0,290 V

The internal energy given off by the upper cuboid
$W_{th\_ab} = C_{hot}$ • (29,43 – 24,62) K = 122,84 J/K • 4,81 K = 590,87 J

The thermal output of the cuboid
$P_{th} = W_{th\_ab} / \Delta t$ = 590,87 J / 230s = 2,57W

and finally our k-value
$k = P_{th} / U_{mid}$ = 2,57 W / 0,290 V = 8,8 W/V

ΔT_{mid} = (11,37+2,94)/2 K = 7,16 K

$\beta = \Delta U / \Delta T_{mid}$ = 369,5mV / 7,16K = 51,64 mV/K

With 127 elements you have 254 contact points.

α = 51,64 mV / 254 = 203μV/K

This is within the manufacturer's specifications in the data sheet. Perfect!

But the most important thing for us is the k-value, because with it we can convert our structure into a wattmeter. For every volt of thermal voltage, 8.8 J are transported from the warm to the cold side per second. Our ESP8266 can resolve 3.0V / 1024 LSB = 2.93 mV / LSB. This corresponds to 8.800 W / V • 0.00293V / LSB = 25 mW / LSB. That means we measure in 25mW steps. As noted elsewhere, the resolution can be increased if you also use an operational preamplifier and / or an ESP32 instead of the ESP8266.

The wattmeter in use

There is now only one small step left to implement the thermal joule and watt meter. The thermocouple that has just been tested is removed and cleaned. The silicone paste must be removed thoroughly so that the element can be glued into a plexiglass holder as in the following photo. During the measurement process, fingers must not be touched, as otherwise uncontrolled heat flows would flow, which would falsify the result.

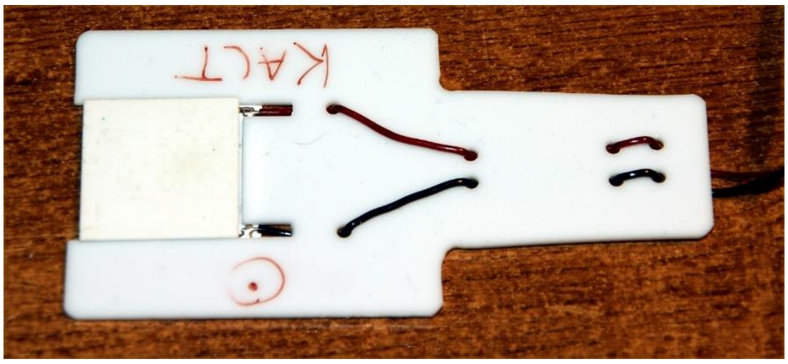


Abbildung 13: Wattmeter - Thermoelement als Fühler

If you hold one side against a smooth surface, the next program tells us how many joules flow from the warmer side to the colder side per second. The smaller line below provides information about the ambient temperature. Using the sensor area of 4 x 4 = 16 cm², you can then extrapolate to the entire tested area, for example windows, walls, oven, house doors ... The test on windows with and without roller shutters shows serious differences in the energy flow in summer and winter.

If the sensor, blackened on one side, is glued to a large heat sink with thermally conductive adhesive, which is built into a styrofoam block and provided with a collimator tube, then very sensitive heat radiation can be measured.

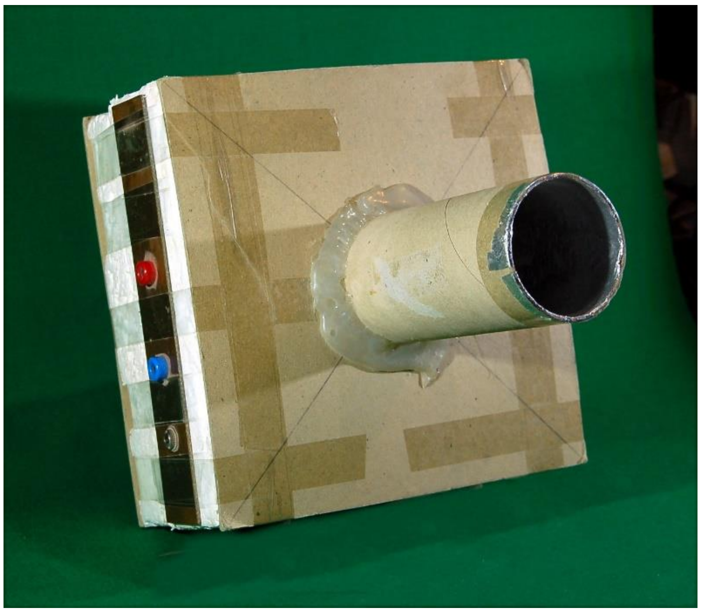


Abbildung 14: Strahlungssensor

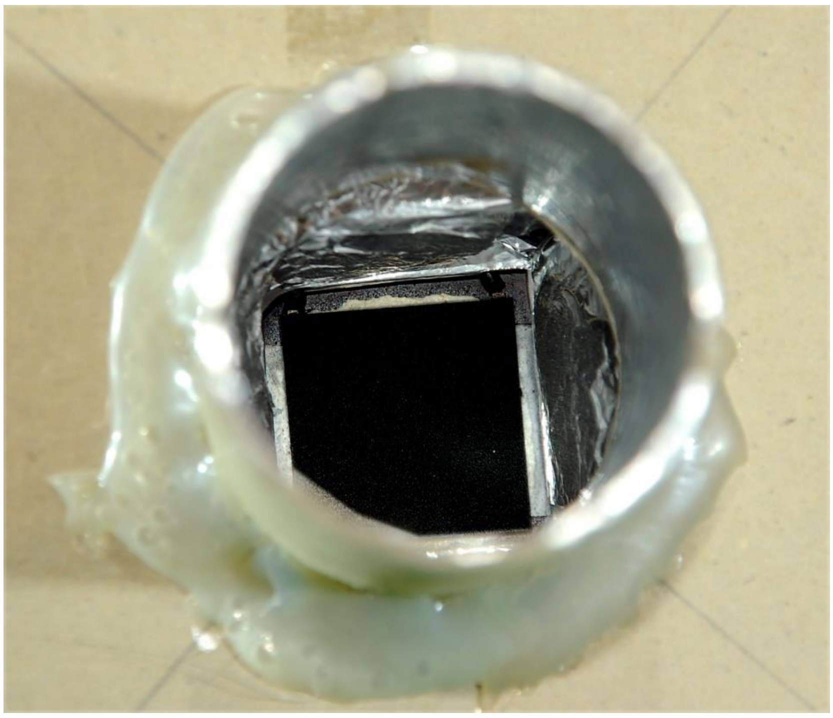


Abbildung 15: Strahlungssensor_Detail

But now to the program. For the watt and joule display we use a large character set with 24 points, in the way it was created in the last episode. If you want to create one yourself, you can use the font2py.rar package. This time, however, I packed the functions for the control into the CharSet class in the charset.py module. The character set is geometer_24.py.

A timer interrupt ensures a measured value update every 2 seconds. During this time, the watt and joule displays alternate. A normal line of text below shows the ambient temperature, which is scanned by a (bare) DS18B20. The circuit is, apart from the lack of one DS18B20, identical to the circuit for calibrating the Peltier element. As there, a calibration is offered in the wattmeter.py program, for which the sensor is best placed or hung for a while in still air, without the sensor surfaces coming into contact with other objects.

```
# wattmeter.py
# Author: J. Grzesina
# Rev: 1.0
# Stand: 2021-06-22
# *******************************************************
from machine import Pin, I2C, ADC, Timer
from time import sleep, ticks_ms,sleep_ms
from onewire import OneWire
from ds18x20 import DS18X20
from oled import OLED
from button import BUTTON8266, BUTTONS
import geometer_24 as zs
from charset import CharSet
import os,sys
```

```
# Pintranslator fuer ESP8266-Boards
# LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                 SC SD

adc=ADC(0)

SCL=Pin(5)
SDA=Pin(4)
i2c=I2C(-1,SCL,SDA)
d=OLED(i2c,128,32)
d.clearAll()
d.setYoffset(4)

ds_pin = Pin(14)     # D5@esp8266
ds = DS18X20(OneWire(ds_pin))
chips = ds.scan() # Liste von bytearrays
numberOfChips = len(chips)
print('Found DS devices: ')
for chip in chips:
    print(chip)

t0,t1=0,0
def los_messen(tim):
    global timerFlag
    timerFlag=True

u=0
u0=0
U=""
def getUth(n=5):
    s=0
    for i in range(n):
        s+=adc.read()
    s=s//n
    u=(int(s/1023*3.0*10000))/10  # Spannung in mV
    return u

def testCalibration():
    calibrated=("ucal.txt" in os.listdir())
    print (calibrated)
    d.clearAll()
    d.writeAt("THERMOSPANNUNG",0,0,False)
    d.writeAt("KALIBRIEREN",0,1,False)
    d.writeAt(">>> KEYS A + B",0,2)
    act=k.waitForTouch(taste,3)
    print(act)
    if calibrated and act==None:
        f=open("ucal.txt","r")
        u0=float(f.readline())
        t0=0
        f.close()
        f=None
        d.clearAll()
        d.writeAt("CALIBR. READ",0,0,False)
        d.writeAt("U0={} T={}".format(u0,t0),0,2)
        sleep(3)
    elif act==1:
        u0,t0=calibrateUth()
    else:
        u0=1543
        t0=9999
        d.clearAll()
        d.writeAt("CALIBR. SET",0,0,False)
        d.writeAt("U0={} T={}".format(u0,t0),0,2)
        sleep(3)
    return (u0,t0)


def calibrateUth():
    d.clearAll()
    d.writeAt("PRESS KEY B",0,0,False)
    d.writeAt("WAIT FOR",0,1,False)
    d.writeAt("RELEASE MSG.",0,2)
    sleep(3)
    d.clearAll()
    d.writeAt("CALIBRATING",0,0)
    ds.convert_temp()
    u0=getUth(20)
    sleep(3)
    t0 = (int((ds.read_temp(chips[0]))*100))/100
    print(t0,u0)
    d.clearAll()
    d.writeAt("CALIBR. DONE",0,0,False)
    d.writeAt("RELEASE KEYS!",0,1)
    sleep(3)
    d.clearAll()
    d.writeAt("U0={} mV".format(u0),0,0,False)
    d.writeAt("T={} *C".format(t0),0,2)
    sleep(3)
    f=open("ucal.txt","w")
    f.write(str(u0)+"\n")
    f.close()
    f=None
    return (u0,t0)

taste = BUTTON8266(0,invert=True) # D3@esp8266
k=BUTTONS()

cs=CharSet(zs, d)

d.clearAll()
d.writeAt("THERMISCHES",0,0,False)
d.writeAt("WATT-METER",0,1,False)
```

```
d.writeAt("JOULE-METER",0,2)
sleep(3)

U0,T=testCalibration()
print("T = {}°C".format(T))
print("U0 = {}V".format(U0))

sleep(2)
T=Timer(0)
timerFlag= True
intervall=2000
T.init(mode=Timer.PERIODIC,period=intervall, \
       callback=los_messen)
energy=0
while taste.tpin.value() == 1:
    if timerFlag==True:
        timerFlag=False
        ds.convert_temp()
        s=getUth(20)
        u=s-U0  # Spannung in mV
        power=int((u*8.8/1000+0.05)*100)/100
        powerString=("{:.2f}".format(power)).replace(".",",")
        energy=energy+int((power*intervall/1000+0.05)*\
                          100)/100
        energyString=("{:.2f}".format\
                       (energy)).replace(".",",")
        sleep_ms(1000)
        t0 = (int(((ds.read_temp(chips[0]))+0.05)*10))/10
        f="{:.1f}"
        t0 = f.format(t0)
        t0String = t0.replace(".",",")
        d.clearAll(False)
        cs.putValue(powerString,"W",0,0,False)
        d.writeAt(" T= {} *C".format(t0String),0,2)
        sleep(1)
        d.clearAll(False)
        cs.putValue(energyString,"J",0,0,False)
        d.writeAt(" T= {} *C".format(t0String),0,2)
T.deinit()
d.clearAll()
d.writeAt("PROG DONE",0,0)
```

With the help of this measuring device, we are well prepared when we build our refrigeration machine with the thermoelectric converters in the next episode. If the energy on the warm side of the machine is sufficiently dissipated, you can (theoretically) achieve a temperature reduction on the cold side of approx. 60K according to the data sheet. We will then test how things are really going. So long, have fun doing handicrafts and measuring.