Supplement – Multichannel thermistor and BaRAGRABA 2.0 construction and calibration

This supplement describes the construction and calibration of the multichannel thermistor and data capture application. This process took place in consultation with an independent electrical engineer and the Wits School of Electrical and Information Engineering. Calibration of the device took place at the Wits School of Physiology using a standardised thermometer for reference temperature measurements.

Multichannel Thermistor

The circuit diagram (Figure 1) below describes the construction of the multichannel thermistor used for temperature measurement.

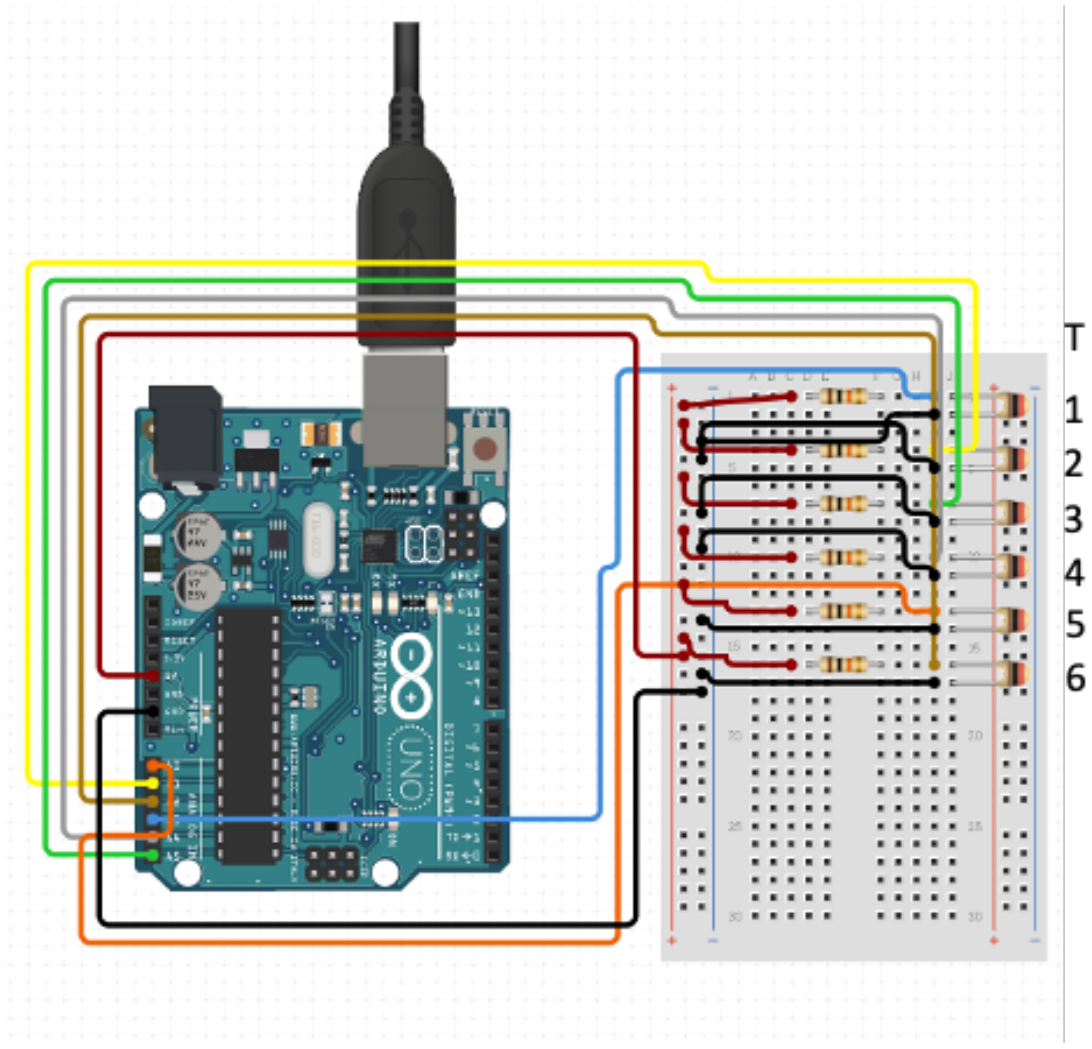


Figure 1 Circuit diagram

Table 1 lists the components used in the construction of the multichannel thermistor.

Table 1 Multichannel thermistor components

Component	**Manufacturer/Serial Number**	**Notes**
Microcontroller	Arduino Uno Arduino LLC Inc, 10 St. James Avenue, Boston Massachusetts, 02116, USA	https://www.arduino.cc/
Thermistors	10k ohm glass bead thermistors DHT0B103J3953SY Communica Electronics	6 thermistors used and labelled T1 – T6
Resistors	10k Ohm	

Table 2 contains the source code for the microcontroller. Note in line 14 of the microcontroller code the values c1, c2 and c3 correspond to the A,B and C coefficients required for the Steinhart-Hart equation. The code has been modified from the original open source version and is available at www.create.arduino.cc [1].

Table 2 Multichannel thermistor source code

```
1.    #include <OneWire.h>
2.    #include <DallasTemperature.h>
3.
4.    #define ONE_WIRE_BUS 5
5.
6.    OneWire oneWire(ONE_WIRE_BUS);
7.
8.    DallasTemperature sensors(&oneWire);
9.
10.   float Celcius=0;
11.   int sensorPins[] = {A0, A1, A2, A3, A4};
12.   float R1 = 10000;
13.   float logR2, R2, T;
14.   float c1 = 1.753159443e-03, c2 = 0.5683506275e-04, c3 = 18.16478028e-07;
15.
16.   void setup() {
17.     Serial.begin(9600);
18.     sensors.begin();
19.     Serial.println();
```

```
  Serial.println("t6,t5,t4,t3,t2,t1,");
}

float readSensor(int number) {
  float value;
  value = analogRead(number);
  R2 = R1 * (1023 / value - 1);
  logR2 = log(R2);
  T = (1.0 / (c1 + c2 * logR2 + c3 * logR2 * logR2 * logR2));
  T = T - 273.15;
  Serial.print(T);
  Serial.print(",");
  return value;

}

void loop() {
  sensors.requestTemperatures();
  Celcius=sensors.getTempCByIndex(0);
  Serial.print(Celcius);
  Serial.print(",");
  float val;
  for (int i = 4; i >= 0; i--) {
    val = readSensor(sensorPins[i]);

  }
  Serial.println();
  delay(500);
}
```

Figure 2 illustrates the modified 3-way Luer stopcock used to record inline fluid temperatures. The thermistor was mounted within the stopcock chamber to minimise interference with fluid flow.

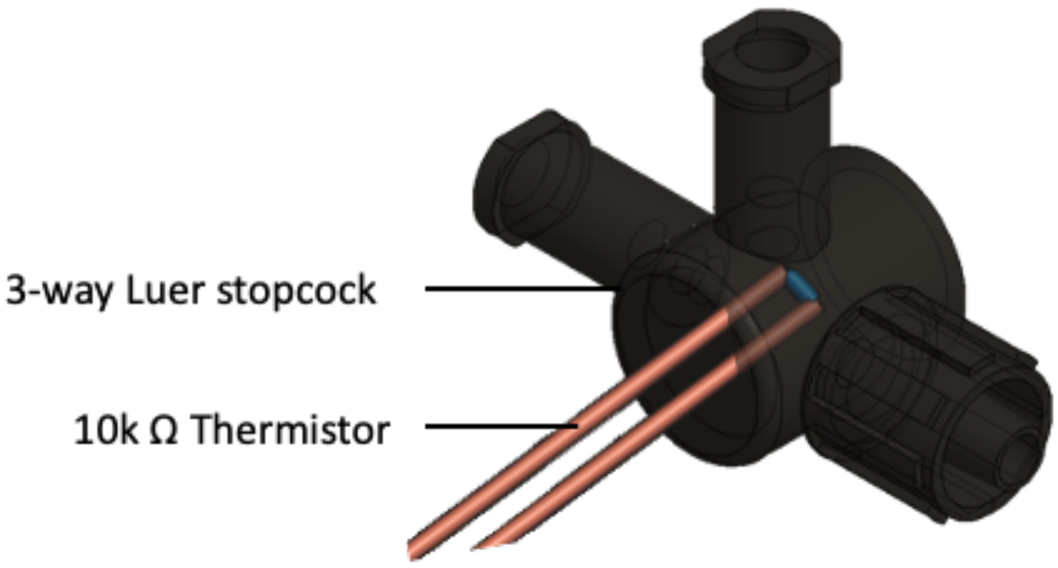


Figure 2 Modified Luer stopcock

BaRAGRABA 2.0

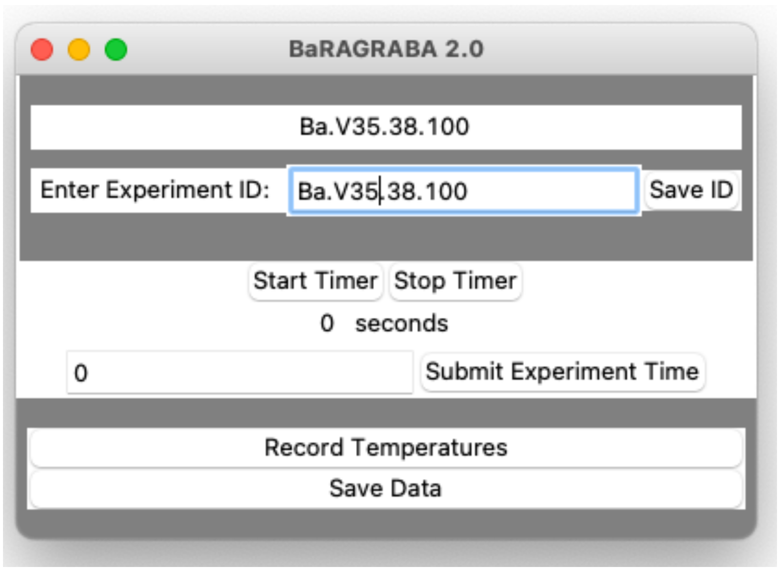


Figure 3 BaRAGRABA 2.0

BaRAGRABA 2.0 (Figure 3) was designed to read the data transferred from the multichannel thermistor and save it in an appropriate format for further management and interpretation. The application was coded on Python 3.8.2 (v3.8.2:7b3ab5921f, Feb 24 2020, 17:52:18). Table 4-3 contains the source code for this application.

Table 3 Source code for BaRAGRABA 2.0

```
import tkinter as tk
import csv
import time
import serial
root = tk.Tk()
root.geometry('400x250')
root.title('BaRAGRABA 2.0')
IDEnterFrame = tk.Frame(root, bg='grey', bd=6,pady=10)
IDEnterFrame.pack()
ExpTimFrame = tk.Frame(root)
ExpTimFrame.pack()
transFrame = tk.Frame(root)
transFrame.pack()
secondtestFrame = tk.Frame(root)
secondtestFrame.pack()
StartExpFrame = tk.Frame(root, bg='grey', bd=6,pady=10)
StartExpFrame.pack(fill=tk.X)
#######################ExpIDEnter##############################
# Enters the experiment Identifier. Needs to update the experiment ID so that a csv filename
# is the same as the experiment ID
en = tk.StringVar()
ent = tk.StringVar()
experimentID = ''
def subID():
global experimentID
ent.set(en.get())
experimentID = en.get()
ExperimentNr = tk.Label(IDEnterFrame, textvariable = ent).pack(fill=tk.X)
expLabel = tk.Label(IDEnterFrame, text='Enter Experiment ID: ').pack(side=tk.LEFT)
expEnter = tk.Entry(IDEnterFrame, textvariable=en).pack(side=tk.LEFT,pady=8)
expButt = tk.Button(IDEnterFrame, text='Save ID', command=subID).pack(side=tk.LEFT)
##########################ExpTime############################
# Experiment duration = airbubble transit time from T1-T5 + 5seconds
# The start timer will be pressed when the flow starts and stopped when
# bubble is distal to T5.
# This transit time will be used to calculate the number of items collected
# from the Arduino.
# items collected = transit time(seconds) * 2 (500ms sampling)+ 10 (5s * 2)
d = tk.StringVar()
d.set('0')
dd = 0
starttime = None
def start():
global starttime,dd
if starttime is not None:
d.set(f"Running")
print("timer is running press STOP")
else:
starttime = time.perf_counter()
print(f"TIMER RUNNING @ {starttime:.2f}")
def stop():
'''Stop Timer'''
global starttime,d,dd
if starttime is None:
d.set(f"Stopped")
print("Timer is stopped press start")
dd = (time.perf_counter() - starttime)
d.set(f"{(time.perf_counter() - starttime):.2f}")
print(f"TIMER STOPPED @ {time.perf_counter():.2f}")
starttime = None
print(f"Transit time is: {dd:.2f} seconds")
timStart = tk.Button(ExpTimFrame, text="Start Timer", command=start).pack(side=tk.LEFT)
timStop = tk.Button(ExpTimFrame, text='Stop Timer', command=stop).pack(side=tk.LEFT)
timTransit = tk.Label(transFrame, textvariable = d).pack(side = tk.LEFT)
SecLabel = tk.Label(transFrame, text='seconds').pack(side = tk.LEFT)
###################Enter Transit time for Second TEst##############
# enter prev experiment duration
# will save the time and assign it to variable dd
# dd used to calculate duration of arduino collection
ttime = tk.IntVar()
def SubTim():
global dd
```

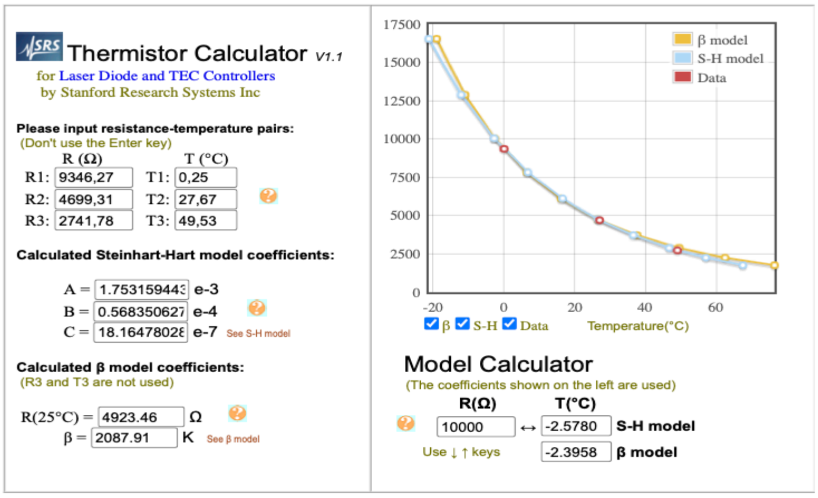
```
dd = ttime.get()
timEnter = tk.Entry(secondtestFrame, textvariable = ttime).pack(side = tk.LEFT)
timButton = tk.Button(secondtestFrame, text='Submit Experiment Time', command = SubTim).pack(side = tk.LEFT)
#################StartExp#############################
# Recording of temperature begins with this button.
# 2 functions are possible. Firstly RArduino() collects information
# from the Arduino. The number of items collected are calculated from
# the transit time as explained in ExpTime above
# The second function saves the data as a csv with filename entered in
# ExpIDEnter. It then clears the data[] list and the input fields on the BaRAGRABA app and
# prepares for a new experiment.
data = []
def RArduino():
global data, ser, experimentID, dd
e = ent.get()
experimentID = e
duration = (dd*(1/0.5)+10)
ser = serial.Serial('/dev/tty.usbmodem14101', 9600)

for i in range (int(duration)):
b = ser.readline()
serial_n = b.decode()
string = serial_n.rstrip()
data.append(string)
time.sleep(0)

ser.close()
print("FINISHED READING")
print(data)
print(len(data))
time.sleep(1)
print("Reminder: Review Data and Save")
def P2CSV():
global data, experimentID, d
print("Saving Data for experiment nr: ", experimentID)
with open('%s.csv' %(experimentID), 'w', newline='') as f:
for i in data:
writer = csv.writer(f)
writer.writerow([i])
data.clear()
strd = str(dd)
with open('testime', 'a', newline = '') as t:
t.write(f'{experimentID} -----  {strd} \n')
time.sleep(2)
print("DONE saving experiment nr: ", experimentID)
d.set(0)
en.set('')
ent.set('')
expStart = tk.Button(StartExpFrame, text='Record Temperatures', command=RArduino).pack(fill=tk.X)
saveExp = tk.Button(StartExpFrame, text='Save Data', command=P2CSV).pack(fill=tk.X)
root.update_idletasks()
root.mainloop(
```

Multichannel thermistor calibration

The multichannel thermistor was calibrated against the Bat-12 Microprobe Thermometer (Physitemp Instruments INC, 154 Huron Avenue, Clifton, NJ 07013, USA). The calibration took place at the Wits University School of Physiology teaching laboratory. The temperature, as measured by the BAT-12, and corresponding resistance of the thermistors were measured at three different temperatures. These temperatures were selected from the expected temperature range of the experiments and represent the lower (0 ºC) the mid-point (27 ºC) and the upper limit (50 ºC) of the temperature range. These temperature-resistance pairs were entered into the SRS Thermistor Calculator (Stanford Research Systems, available at https://www.thinksrs.com/downloads/programs/therm%20calc/ntccalibrator/ntccalculator.html). A, B and C coefficients calculated by the SRS Calculator were entered into the Steinhart-Hart equation in line 14 of the multichannel thermistor source code, and correlate with c1, c2 and c3. Figure 4-4 shows the results of this calibration



Thermistor Calculator v1.1
for Laser Diode and TEC Controllers
by Stanford Research Systems Inc

Please input resistance-temperature pairs:
(Don't use the Enter key)

	R (Ω)		T (°C)
R1:	9346,27	T1:	0,25
R2:	4699,31	T2:	27,67
R3:	2741,78	T3:	49,53

Calculated Steinhart-Hart model coefficients:

A = 1.75315944 e-3
B = 0.568350627 e-4
C = 18.1647802 e-7 See S-H model

Calculated β model coefficients:
(R3 and T3 are not used)

R(25°C) = 4923.46 Ω
β = 2087.91 K See β model

Model Calculator
(The coefficients shown on the left are used)

R(Ω)		T(°C)	
10000	↔	-2.5780	S-H model
Use ↓ ↑ keys		-2.3958	β model

Figure 4
Calibration calculation

References

1. Christoulakis I. Make an Arduino Temperature Sensor (thermistor tutorial) © LGPL: ARDUINO; 2020. Available from: https://create.arduino.cc/projecthub/iasonas-christoulakis/make-an-arduino-temperature-sensor-thermistor-tutorial-b26ed3?f=1#comments. [Accessed 20 April 2020]