FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Universidade de Coimbra

Técnicas de Instrumentação e Controlo

Turma:PL1

Respiratory Rate Measurement
Final Report

Disa Palma - 2017255933
Beatriz Negromonte Batista dos Santos - 2017109753

Contents

1 Introduction note

During the execution of the project, we made several changes that made the specification documentation sent earlier not equivalent to the final project. Therefore, we proceed to change the specification part and add in this final report the updated version. The final project goal is still the same: create an instrument to detect the respiratory rate through temperature change.

2 General Description

2.1 Name of the instrument and description

Our instrument will be a "Respiratory rate detector", composed by a temperature sensor MCP9808 and an Arduino MKR1000.

Respiratory diseases are a major cause of death in the world and the pandemic just accentuated the need for low cost respiratory monitoring equipment. The measurement of the breath rate gives important information about human vital signs, such as prediction of cardiac problems or mortality rate, but can also provide a track for the progression of respiratory diseases. Therefore, as breathing is an important indicator of the physiological conditions, we will monitor the entire process of a breathing cycle, composed by the inhalation of oxygen and exhalation of carbon dioxide. [2][9]

2.2 General block diagram

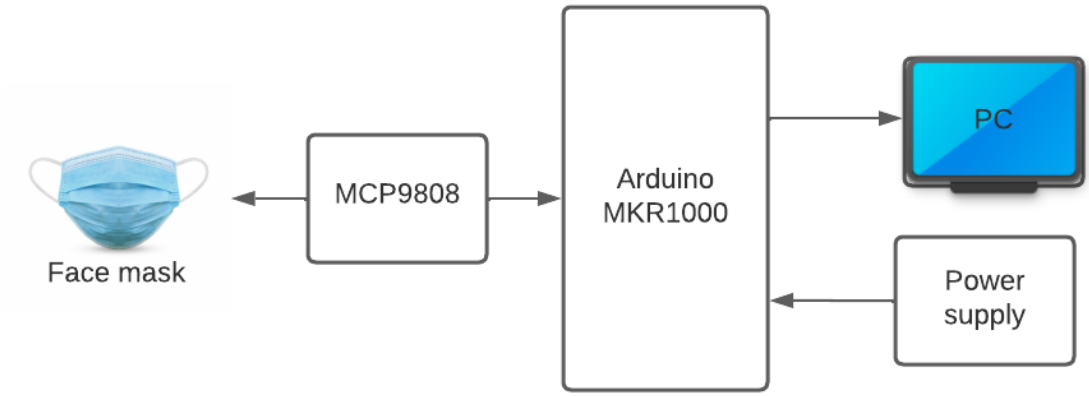

Image 1 - General block diagram representing the planned equipment

2.3 Use: why, where and when will it be used

Breath sensors are important to detect irregularities in the respiratory frequency related to respiratory diseases or to follow the evolution of one. During a pandemic, something like this becomes essential and could make a difference in places where there is a shortage of similar equipment. This breath rate measurer can be used both at the hospital or at home, and it can also be an alternative to hospitals that seek low-cost options. [2]

2.4 User: who will use it

All the patients that are suffering from respiratory diseases or related, their caretakers or health operators will use it regularly. Alternatively, students and researchers can also take benefit from the low cost of the instrument and use it for studies or to get practical knowledge.

2.5 Interaction: how will it be used

We will construct this breath sensor by detecting changes of temperature in the respiratory cycle, as the exhaled air has more heat than the inhaled one, giving a temperature difference. The temperature will be detected with the use of a mask with a temperature sensor incorporated. The temperature difference will then be transduced to a time varying electrical signal - the respiratory frequency. [5]

2.6 Conditions related with the use of the instrument

This is not an invasive instrument so there are not a lot of conditions associated. However, we need to have in consideration that breath rate varies with age and is also affected by the state of the patient, with a successful measurement when the patient is relaxed. In the table, is possible to see the variation of the respiratory rate according to the age[3]:

Age range	Breaths/minute
0-1	30-60
1-3	24-40
3-6	22-34
6-12	18-30
12-18	15-21
over 18	12-20

2.7 Conditions related with the operator of the instrument

The operator of the instrument needs to adjust the mask properly and confirm that all the equipment is correctly connected.

3 Requirements

3.1 Sensor

The sensor chosen for the purpose is the MCP9808 digital temperature sensor, that has a sensitivity of ±0.25°C over the sensor's -40°C to +125°C range and precision of +0.0625°C. This sensor is a good choice as it doesn't require any calibration or trimming. The sensor is ideal for sophisticated, multi-zone, temperature-monitoring applications, such as general purpose, industrial application and consumer electronics, as example. Although it's practical indication does not mention bio signals monitoring, we have tested for it as it was the available sensor we had and it was possible to detect different temperatures from inhale and exhale, making possible to identify the peaks.

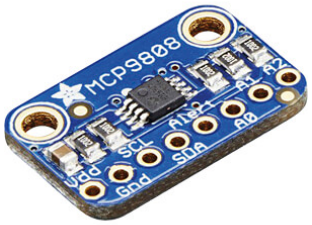

Image 2 - Sensor image, obtained from the specification sheet

3.2 Signal Conditioning

Signal conditioning is related to the measurement of the signal with correction and efficiency so it can be read in subsequent stages of the device. We are going to consider it in terms of amplification, filtering and isolation. Amplifiers increase the input amplitude for low voltage and the resolution of measurement. In filtering, the unwanted noise is rejected in a frequency range. Last, isolation is related to the perturbations a signal can have along the electrical path from the sensor to the circuit.

As for isolation, there can be some noise from the sensor to the Arduino, so we will have to make sure that the patient is in an environment where it won't have any external interference and the wires are isolated with a cable.

3.3 Data Acquisition

According to the MCP0808 specifications, this sensor outputs an analog voltage that is proportional to absolute temperature. The conversion is done with an internal ADC is used to convert the analog voltage to a digital word, as the specification sheet mentions: "The digital word is loaded to a 16-bit read-only Ambient Temperature register (TA) that contains 13-bit temperature data in two's complement format."

Regarding the storage of the data, it will be stored in a file on the laptop through the practice, as the Arduino has a limited memory and does not keep the information.

3.4 Data Processing

While measuring the temperature when a person inhales and exhales, the values will be shown through a graph where we will extract the peaks associated with high temperatures, denoting an exhalation. Through that, it will be possible to calculate the breath rate.

3.5 Display

To display the values of respiratory rate, we will use the computer. We previously though about using a display connected to the Arduino, however, through the way the code was produced in a first moment, we could not measure the respiratory rate at exact real. A future improvement would be using the instrument independently from the computer and show the results in a display, what would also require a battery.

3.6 Power Supply

Regarding power supply to Arduino, the board can operate with enough power from the USB connection to the computer. However, the Arduino can only operate in the range of 0 V and 1.1 V since that's the internal reference to avoid fluctuations on the values or errors in the interpretation of the sensor signals.

4 Preliminary Specification

4.1 Detailed Block Diagram

The sensor consists of a band-gap-type temperature sensor, user programmable registers and a 2-wire SMBus/I^2C protocol compatible serial interface. The schema shows the register structure block diagram contained in MCP9808 datasheet:

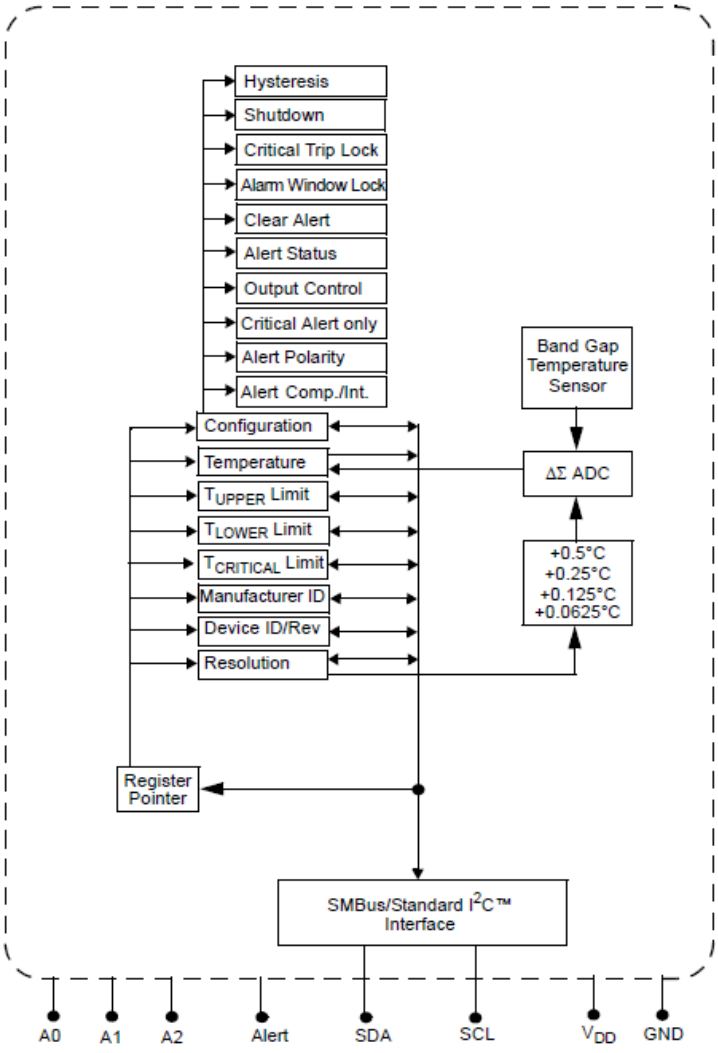

Image 3 - Detail block diagram

4.2 Bill of material

The materials needed for the instrument are the MCP9808 sensor, a face mask and an Arduino kit, which contains an Arduino and all the electronics components as well, and a computer.

Material	Price (€)
MCP9808 sensor	5.90
Arduino kit	31.00
Face mask	10.00
Elastic	0.10
Total	47.00

The face mask used for this project was produced through a 3D printer 'Ultimaker'. However, the user does not need to have a 3D printer to produce this instrument, it's possible to use a 3D printing service. We estimated the value above from the information and simulation contained on [4], estimating from the time - approximately 6 hours and 25 minutes - that it took to print the mask and the support for the sensor.

4.3 Task planning

The tasks needed to accomplish this project and create a functional respiratory monitor are:

1 - Buy all the material mentioned in 3.2 and correctly connect all the needed parts, as mentioned in 3.1.

2 - Write the code to detect the output from the MCP9808 sensor, measuring it continuously in a properly defined time frame, that will be stored together with the data.

3 - Analyze the signal received on point 2. In this step, we will also need to define the parameters that we will be looking for, such as which temperature will be perceived as exhale and which one will be as inhale.

4 - Calculate the breath rate through the information acquired in point 3.

5 - Display the breath rate and the time of the measurement in the computer.

4.4 Risk Identification

The risk associated with this device might be the incorrect diagnostic of the patient by a false positive or false negative detection of a respiratory problem. An error might occur as well with the sensibility of the sensor.

5 Prototype description

The mask body was obtained from an online model [6]. We tried both M and L sizes for the mask body wide, both are appropriate for the respiratory measurement. For the sensor's support, we

measured the mask and produced the support at InventorAutodesk:

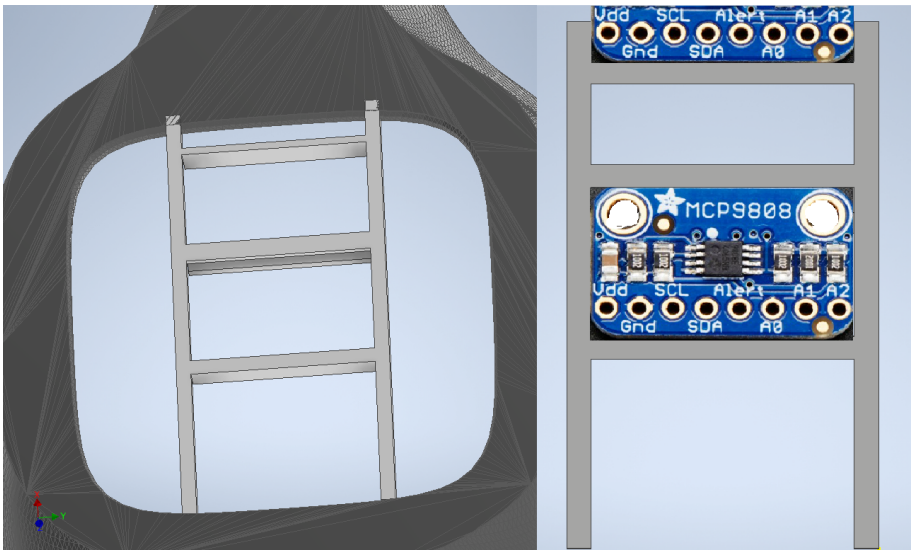

Image 4 - Sensor support designed in Inventor

With this support structure, we intend to measure the temperature through the through the mouth, even though in a finished mask it should also have the option of measuring through the nose with the sensor being adjusted according to the position desired.

The design was made to allow the heat to escape from the mask. We considered designing an structure with further support to adhere the mask, however, we postponed this due to our assumption that it would influence the detection of temperature: the mask would cover the face in a way it would keep the heat inside the mask, therefore, bringing temperature up and making the measurement not accurate after a certain time. However, we made a trial with the support in a base made from the 3D print that got good results, similar to the ones we had with the support without the base around to cover the mask entrance. Also, there are other types of support that may be considered for the purpose of better stabilization of the sensor on the desired positions.

Besides that, there's a need to add a support for the cables that connect the sensor to the Arduino, as the support created is not resistent enought for the weight of them.

The mask has the following structure:

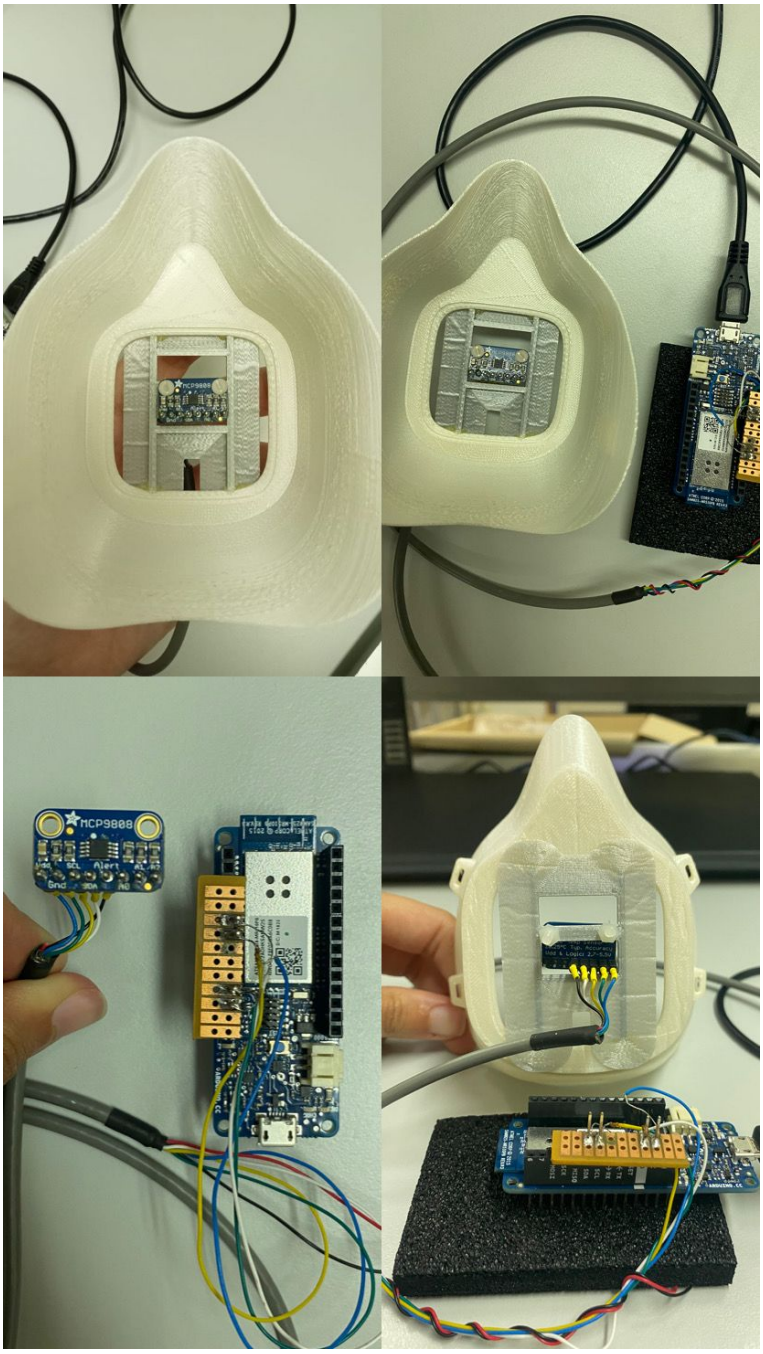

Image 5 - Mask prototype.

An improvement needed here is an isolation for the cables and the Arduino, as we were not able to provide a proper one due to time constraints.

The Arduino and the sensor are connected as per the image bellow:

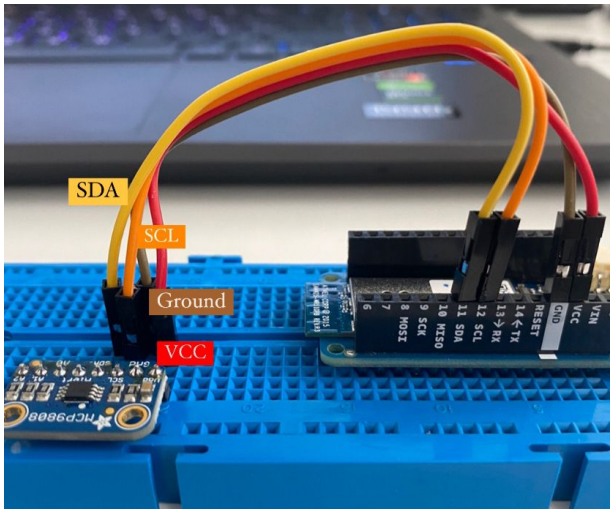

Image 6 - Cable connections between the sensor and Arduino.

For reference for the connections made in image 5, we add the image 6 above to simplify the understanding. The red cable is used for VCC (first pin on the MCP9808), the brown cable is used for ground, the orange is for SCL (12 on Arduino) and the yellow one is for the SDA (11 on Arduino). The alert, A0, A1 and A2 pins from the sensor are not used.

6 Code documentation

The code has the following sequence:

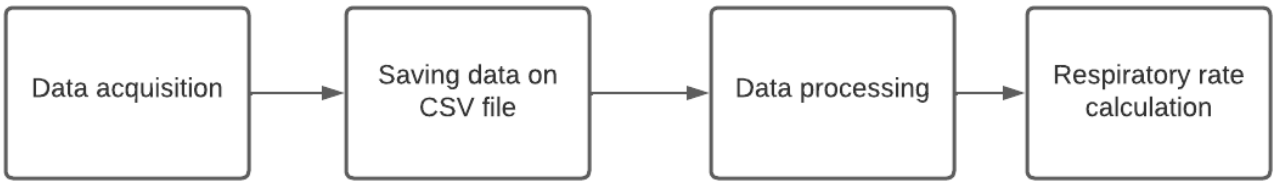

Image 7 - Code Sequence

6.1 Data processing algorithm

For data acquisition, we've created an Arduino algorithm, where the code begins by creating the MCP9808 object with the Adafruit_MCP9808[11], an open source code library created to obtain data from MCP9808 sensor. In the loop, the temperature will be read in Celsius degree and will be printed. Later, we proceed to save the CSV file on ArduSpreadsheet folder. In this case, the user will control the data acquisition on the ArduSpreadsheet tab, on the tools tab - there are the options

to start, stop and erase the data acquisition.

For the Arduino code, we have the following:

```
#include "Adafruit_MCP9808.h"
int delay_value = 100;
// Create the MCP9808 temperature sensor object
Adafruit_MCP9808 tempsensor = Adafruit_MCP9808();

void setup() {
  Serial.begin(9600);
  Serial.println("MCP9808 demo by Robojax");

  // Make sure the sensor is found, you can also pass in a different i2c
  // address with tempsensor.begin(0x19) for example
  if (!tempsensor.begin()) {
    Serial.println("Couldn't find MCP9808!");
    while (1);
  }
}


void loop() {
  float c = tempsensor.readTempC();
  Serial.println(c);
  delay(delay_value);

}// loop end
```

Image 8 - Arduino code for data acquisition

For the data analysis, we developed 2 codes, one where there is a need to manually save and import the CSV file after data acquisition from Arduino to Python (file named 'export_csv_code.py'), and a second code where it's possible to get the data from the Arduino and analyze it automatically on Python (file named 'real_time.py').

For the first one, where there is the need to save and import the CSV file, on Python, when the code starts, there will be the need to upload the CSV file directly to it:

```
# choose file to upload
uploaded = files.upload()

"""##Functions"""

def import_file(uploaded, fonte):
  ### uploaded is the archive chosen and fonte is str type where it was measured from: mouth or nose
  warnings.filterwarnings('ignore')
  # import/read file and add column name:
  a = str(uploaded.keys())
  a = a[12:]
  for i in range(len(a)):
    if a[i]=='_':
      break
    else:
      b = a[:i+1]

  file_name = b+'_'+fonte+'.csv'
  df = pd.read_csv(io.BytesIO(uploaded[file_name]))
  df = df.set_axis(['Time','Temp'],axis='columns')
  df = df.drop_duplicates(subset='Time', keep ='last', ignore_index = True)
  df['Count']=df['Time']
  df['T(ms)']=df['Time']

  # Calculates the minutes and seconds:
  temp= []
  time_ms = []

  for index in df.index:
    # replace the date+time for just the time
    df['Time'] = df['Time'].replace(df['Time'][index],df['Time'][index][11:])
    time = df['Time'][index]
    if index == 0:
      # sets the first measurement as 00:00
      start = datetime.strptime(time,'%H:%M:%S.%f')
      df['Count'][index] = '00:00.000'
      df['T(ms)'][index] = '0'
    else:
      pt = datetime.strptime(time,'%H:%M:%S.%f')
      diff = pt-start # calculate the difference between the start and time of measurement
      df['Count'][index] = str(diff)[2:-3]
      df['T(ms)'][index] = str(diff.total_seconds()*1000) # calculates the ms
    temp.append(df['Temp'][index])
    time_ms.append(df['T(ms)'][index])

  # 1st plot:
  df.plot(x = 'T(ms)', y='Temp', kind = 'line', figsize = (20,10), title = 'Temperature (C) x time (Milliseconds)',
          legend = 'Temperature',xlabel = 'Time (milliseconds)', ylabel = 'Temperature (C)')
  plt.show()

  return df
```

Image 9 - import_file function in Python

An observation about the file is that, in this case, it needs to have the name with an underline and the font that was measured from - nose or mouth -, such as "test_mouth".

For the second option, we developed a code that uses the Serial module on Python to get the data that is printed on the Arduino code, creates a CSV file and analyzes it after:

```
14
15  arduino_port = "COM6" #serial port of Arduino
16  baud = 9600 #arduino uno runs at 9600 baud
17  fileName="test1.csv" #name of the CSV file generated
18  ser = serial.Serial(arduino_port, baud)
19
20  if not ser.isOpen():
21      ser.open()
22  print('com6 is open', ser.isOpen())
23
24
25  time_seconds = 45 # for how many seconds data acquisition happens
26  now = datetime.now()
27  print(now)
28  later = now + timedelta(seconds = time_seconds)
29  current_time = now.strftime("%H:%M:%S.%f")
30  print(later)
31
32  while now <= later:
33      now = datetime.now()
34      current_time = now.strftime("%H:%M:%S.%f")
35      print("Current Time =", current_time)
36      getData=str(ser.readline())
37      data=getData[0:][2:-5]
38      print(data)
39
40      file = open(fileName, "a")
41      file.write(str(now) + ',' + data + "\n") #write data with a newline
42
43  print("Data collection complete!")
44  ser.close()
45  file.close()
```

Image 10 - Python code for real time acquisition from Arduino

In this case, the user needs to control for how long the code will run on the while loop with the "time_seconds" variable mentioned on line 25.

After this, both codes process and analyze the data in the same way, returning the peaks plot and the respiratory rate. For the data processing, we use the following libraries in Python: datetime, scipy.signal, pandas and matplotlib. The 'import_file' will read the CSV and transform into a dataframe, with the following columns: Time, Temp, Count and T(ms). The first one is the date and time that the data was acquired, it will be transformed into just time through a for cycle, and will later be transformed in the count for minutes in the Count column and in milliseconds, in the T(ms) column. The Temp column is the temperature read in Celsius degree. This function will return the dataframe with all those changes mentioned above, and a display of the data on Time vs

Temperature:

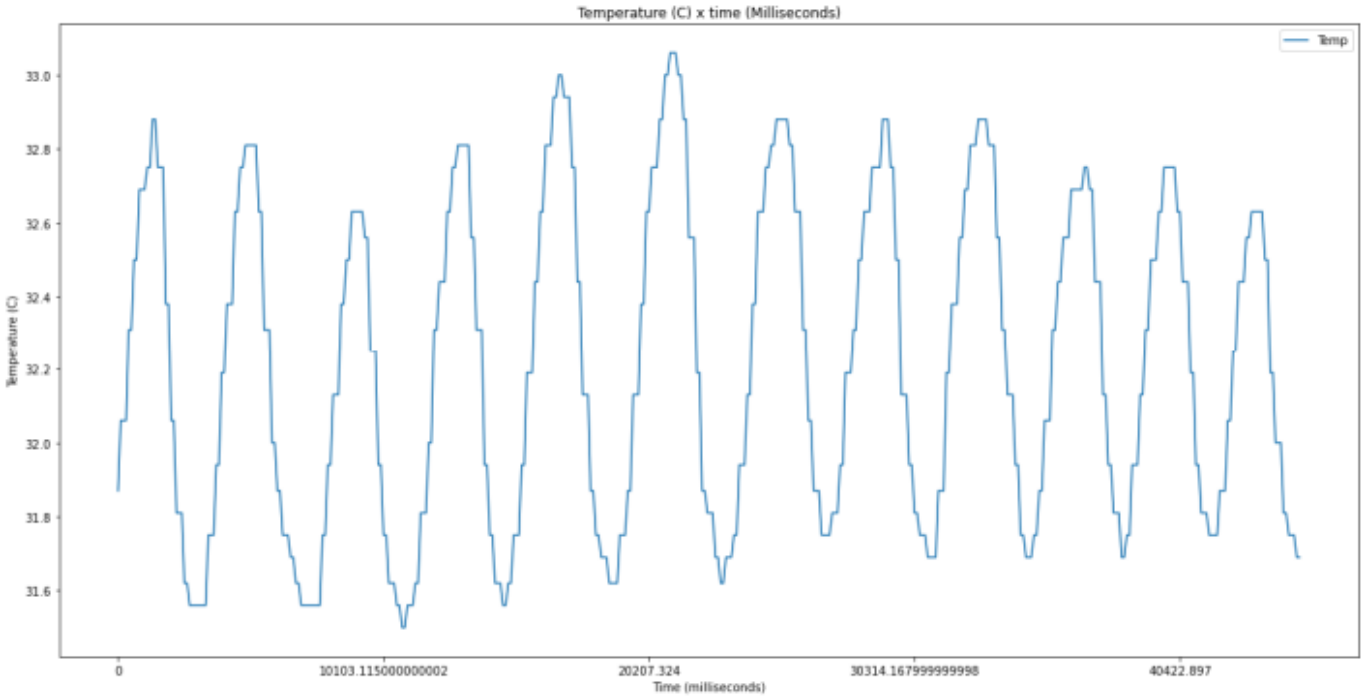

Image 11 - Temperature (^{o}C) x time (ms) graph from mouth, with 100 ms delay

The user is provided a question for analyzing just a certain part of the data or not, based on the time. The choice to add this option is made as the sensor still take some time to accurately sense the temperature, so there might be error in detecting the temperature peaks early or in the very end. If the choice is 'No' for choosing where to analyze, the peaks will be detected after 20 seconds still, as we estimated this was the average value it would take to "calibrate" the sensor and acquire accurate values.

```python
def plot_peaks(df2,cut,mode):
  # this function cuts the time for the default (-20s) or for the chosen one and plots the peaks. Also returns the respiratory rate
  if cut=='Y':
    start = 10
    end = 4
    # defines start and end from users choices:
    while start>=end:
      start = float(input('Start from (s): '))*1000
      end = float(input('End at (s): '))*1000

    # cuts for the time frame chosen:
    for i in range(len(df2)):
      time = float(df2['T(ms)'][i])
      if start>time:
        df2 = df2.drop(i)
      elif end<time:
        df2 = df2.drop(i)

  elif cut=='N':
    start = 20000
    # cuts for default:
    for i in range(len(df2)):
      time = float(df2['T(ms)'][i])
      if start>time:
        df2 = df2.drop(i)
  df2 = df2.reset_index(drop=True)

  # if it was measured by nose, the prominence considered is 0.2, if it was for mouth, is 0.5
  # this is due to the difference between measurement type
  if mode=='nose':
    peaks, properties = find_peaks(df2['Temp'], prominence = 0.2)
  else:
    peaks, properties = find_peaks(df2['Temp'], prominence = 0.5)

  # calculates the respiratory rate by counting the peaks calculated above, multiplying by 60 (seconds)
  # and dividing by the diference between the start and end time
  first = float(df2['T(ms)'][0])
  last = df2.iloc[-1]
  seconds =(float(last['T(ms)'])-first)/1000
  x = len(peaks)*60/seconds
  print('A contagem resulta em ', round(x),'respirações por minuto')

  df2.plot(x = 'T(ms)', y='Temp', kind = 'line', figsize = (20,10), title = 'Temperature (C) x time (Milliseconds)',
    legend = 'Temperature',xlabel = 'Time (milliseconds)', ylabel = 'Temperature (C)')
  plt.plot(peaks, df2['Temp'][peaks], ".")
  plt.show()
```

Image 12 - plot_peaks function on python

After this, there will be another plot for the Time (ms) vs Temperature (C) detecting the peaks and the respiratory rate calculated for the chosen timeframe. This was made through the find_peaks function, with prominence - detects the peaks considering the height of the point, compared to the other points around it - as the parameter's choice for detecting the peaks, with 0.2 value if the measurement was from nose, and 0.5 value if it was made from the mouth:

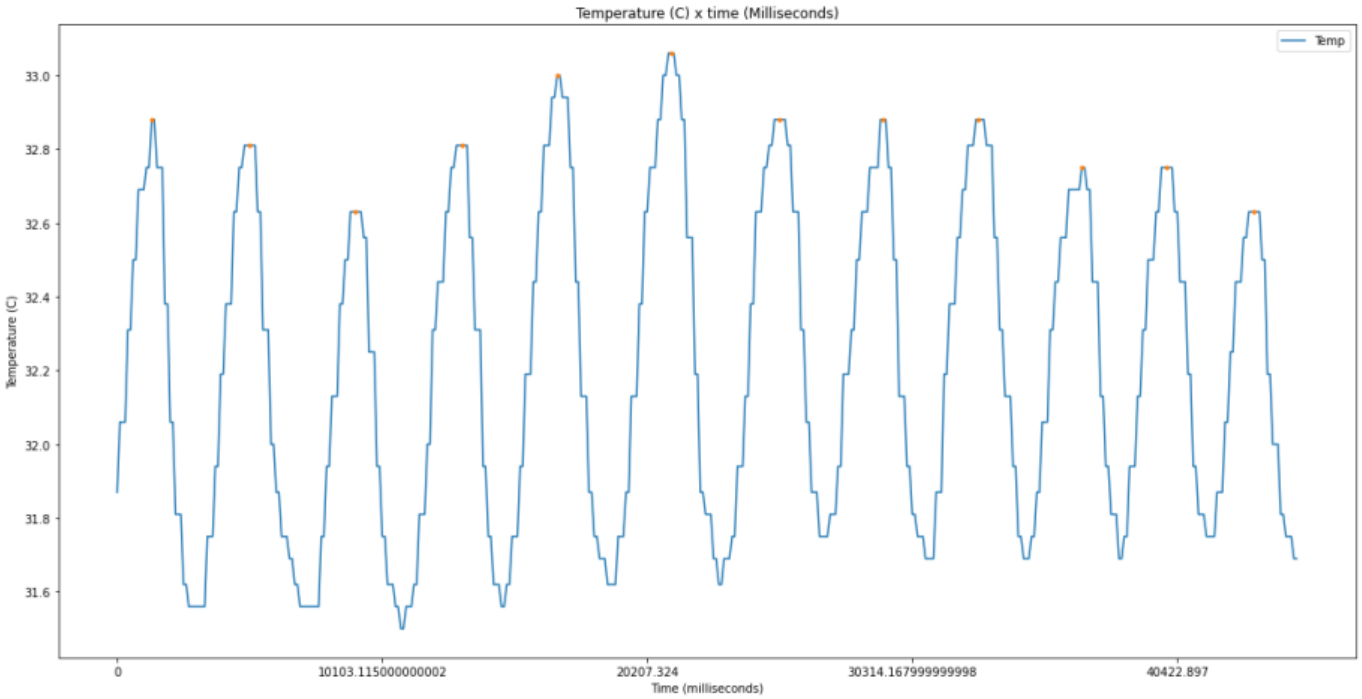

Image 13 - peaks plot and count of respiratory rate

We notice that if the temperature is measured from the nose, there is a difference in the height of the peaks compared with the measurements from the mouth. The peaks have small heights compared to the ones from the mouth:

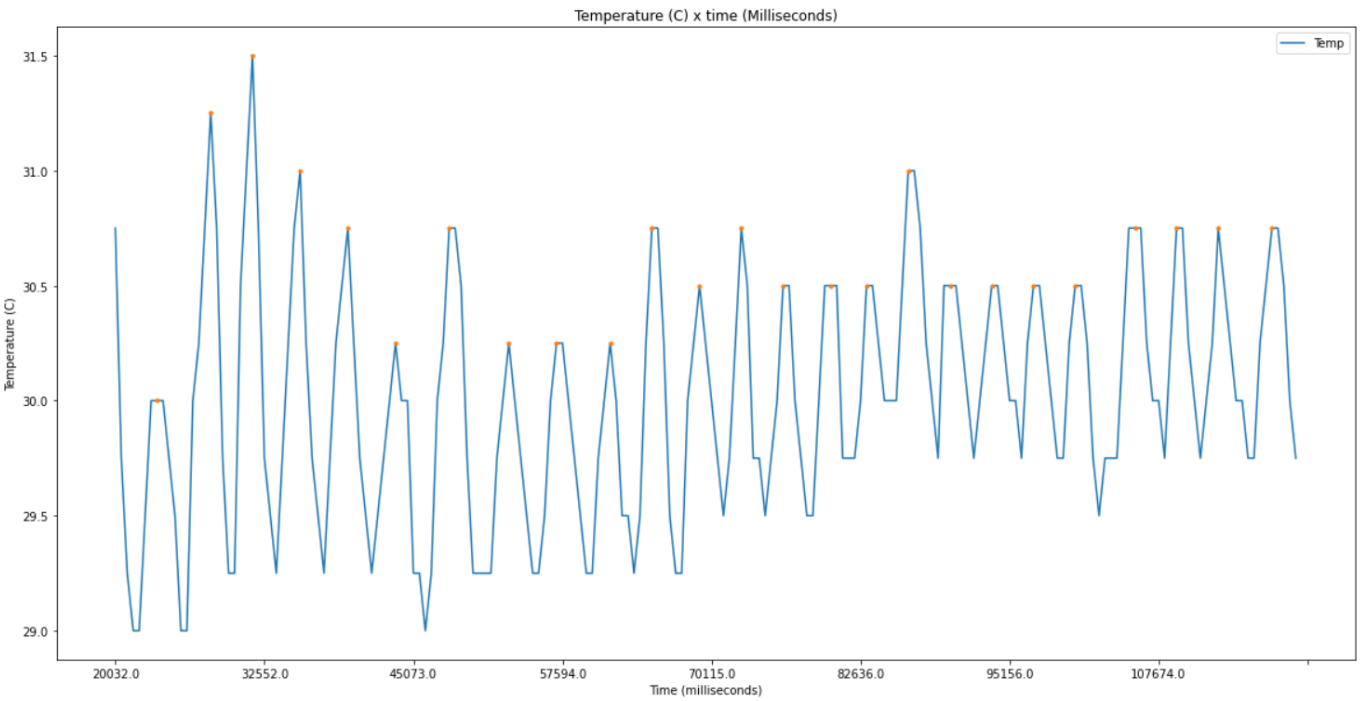

Image 14 - Peaks plot from the mouth

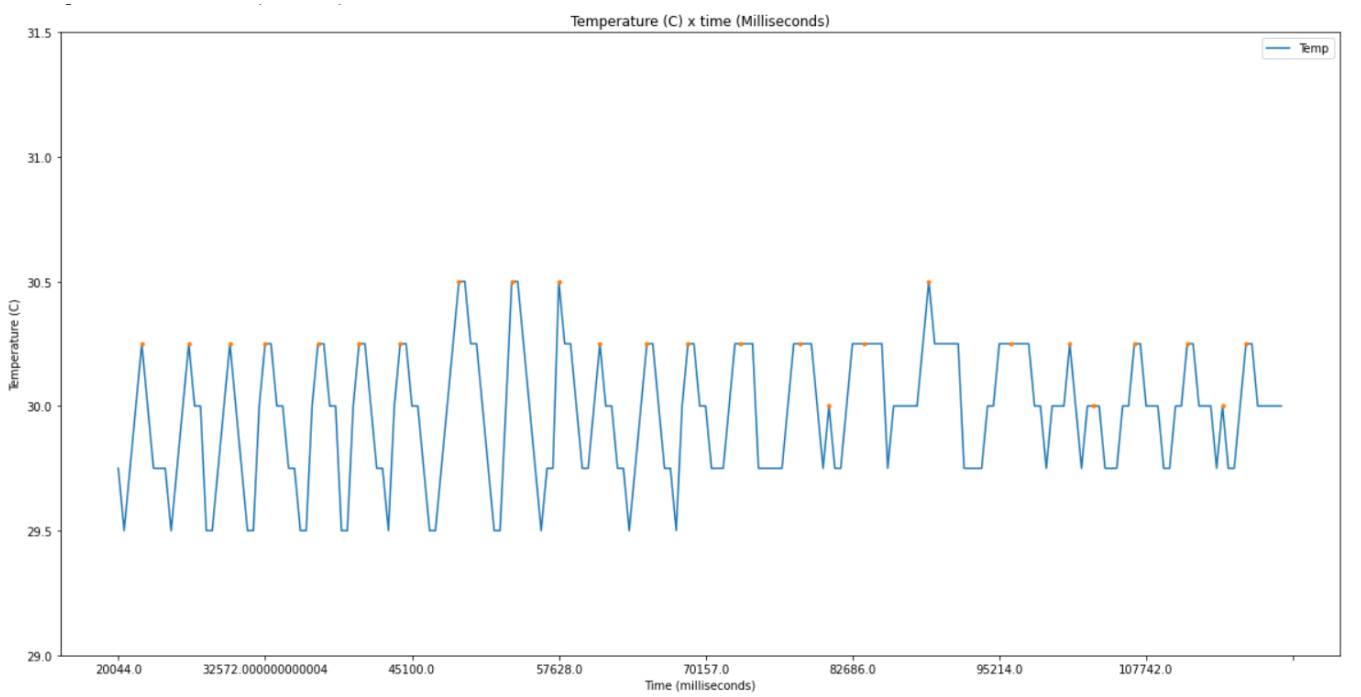

Image 15 - Peaks plot from the nose

However, both plots returned a measure of 15 breathes/minute, in this 500 ms delay between

each acquisition, so we consider that even though the use of the sensor from the nose appears to have little sensibility to the temperature change, there is still an accuracy on the prediction of the value as the prominence value difference permits to detect the peaks for the nose.

7 Improvements

Besides all the improvements already mentioned through the report, we add that there are still many others that can be made, although we have improved some parts already since the first report delivery.

Beginning from the mask, initially the support created was still vulnerable, as it's not very resistant because it only had four points of contact - on the top and the bottom of the mask. So we decided to increase the area of contact on the mask. Although this change decreased the area for ventilation, we found that it did not interfere significantly with the temperature detected. Another change that could be made is to compare two mask types, the one presented above and another one were the only opening in front is for fitting the sensor. Although we considered that the temperature could be different and change the results - as mentioned in section 5 - a test could be performed to compare the results and create a different structure, more stable, for the sensor.

The sensor, as mentioned in 3.1, does not contain practical indication for bio signals data acquisition. If needed one with clear specification for this purpose, either due to project requirement or compliance, a bio sensor generally available is PLUX temperature sensor [1], which would increase the cost of the instrument, as the MCP9808 sensor costs 5.80€ and the temperature (TMP) sensor costs 30.75€. Initially, the code only analyzed the data acquisition detected through Arduino and saved to a CSV file and imported to Python code. However we changed the code to become a real time data analysis, where we improved the code to automatically import data from Arduino to Python, such as [8], where it's possible to create an Arduino Data Logger with Python and the serial module. Even though we changed this, there was no improvement made from the data analysis perspective, where we maintained the code simple without any considerations to noise on data acquisition. Therefore, the data analysis could be improved.

References

[1] Arduino data logger (csv) with sensors and python. `https://www.pluxbiosignals.com/products/temperature-tmp-sensor`. [Online; accessed 20-July-2022].

[2] F.Q. AL-Khalidi, R. Saatchi, D. Burke, H. Elphick, and S. Tan. Respiration rate monitoring methods: A review. *Pediatric Pulmonology*, 46(6):523–529, 2011.

[3] Susannah Fleming, Matthew Thompson, Richard Stevens, Carl Heneghan, Annette Plüddemann, Ian Maconochie, Lionel Tarassenko, and David Mant. Normal ranges of heart rate and

respiratory rate in children from birth to 18 years of age: a systematic review of observational studies. *The Lancet*, 377(9770):1011–1018, 2011.

[4] Bohyun Kim. How to price 3d printing service fees. `https://acrl.ala.org/techconnect/post/how-to-price-3d-printing-service-fees/`. [Online; accessed 22-June-2022].

[5] A.; VYAS Vyas S.; Kodgirwar V. Kodgirwar, S.; Chandrachood. Design of signal conditioning circuit for biomedical sensors and battery monitoring circuit for a portable communication system. *American Journal of Engineering Research*, 5(6):100–107, 2016.

[6] lafactoria3d. Covid-19 mask v2 (fast print, no support, filter required). `https://www.thingiverse.com/thing:4225667/files`. [Online; accessed 21-June-2022].

[7] Indrek Luuk. Logging arduino serial output to csv/excel (windows/mac/linux). `https://circuitjournal.com/arduino-serial-to-spreadsheet`. [Online; accessed 03-June-2022].

[8] Liz Miller. Arduino data logger (csv) with sensors and python. `https://www.learnrobotics.org/blog/arduino-data-logger-csv/`. [Online; accessed 24-June-2022].

[9] Bilel Neji, Ndricim Ferko, Raymond Ghandour, Abdullah S. Karar, and Houssam Arbess. Microfabricated rtd based sensor for breathing analysis and monitoring. *Sensors*, 21(1), 2021.

[10] Ian Smith, John Mackay, Nahla Fahrid, and Don Krucheck. Respiratory rate measurement: a comparison of methods. *British Journal of Healthcare Assistants*, 5(1):18–23, 2011.

[11] Kevin Townsend and Limor Fried. Adafruit mcp9808 library. `https://github.com/adafruit/Adafruit_MCP9808_Library`. [Online; accessed 03-June-2022].