**Supplementary material**

1. **Software**

**Code 1:** Arduino.ino

|  |
| --- |
| 1. **double** setTemp; 2. **double** input; 3. **double** voltageControl; 5. // Set the pins for the heating and cooling elements 6. **int** heatingPads = 4; 7. **int** peltiers = 5; 9. // Set histeresis in degrees 10. **double** histeresis =0.1; 12. // String from python tkinter is initialize as an empty string 13. String receivedString=""; 15. // the setup routine runs once when you press reset: 16. **void** setup() { 17. // initialize serial communication at 9600 bits per second: 18. Serial.begin(9600); 19. // The default set temperature for the OpenTCC is 15 degrees which 20. // is overwritten when the Python GUI starts. 21. setTemp = 15; 22. // These two lines of code set the pins required for the actuator 23. // as output pins. 24. pinMode(4, OUTPUT); 25. pinMode(5, OUTPUT); 26. } 28. // the loop routine runs over and over again forever: 29. **void** loop() { 30. //------------------------------------------ 31. **int** average = 0; 32. // The for loop serves to average the signals from the Pt100 33. // and improve the signal-to-noise ratio 34. **for** (**int** i=0; i < 100; i++) { 35. average = average + analogRead(A0); 36. delay(2); 37. } 39. average = average/100; 40. // Convert the analog reading (which goes from 0 - 1023) 41. // to a voltage (0 - 5V): 42. // The Arduino ADC produces bits that correspond to voltage values 43. // according to the maximum voltage of the Arduino (5 V) 44. // However, in this case we use mV 45. **double** voltage1 = average \* (5.0 / 1023.0) \* 1000; 46. // The map function maps a low mV value to a temperature and 47. // high mV value to a temperature 48. **double** temperature = map(voltage1\*100, 36657, 87488, 3320, 2040); 49. // Temperatures used for the map function 33.2 and 20.4 50. **double** tempC = round(temperature/10); 51. **double** temperature1 = tempC/10; 53. //------------------------------------------ 54. **int** environment = 0; 56. **for** (**int** i=0; i < 30; i++) { 57. environment = environment + analogRead(A1); 58. delay(2); 59. } 61. environment= environment/30; 62. **double** voltage2 = environment \* (5.0 / 1023.0); 63. **double** temperature2 = (voltage2-1.375)/0.0225; 64. // AD22100 Mathematical relation 66. //----------------------------------------- 68. input = temperature2; 70. // Send data by serial for plotting 71. // The tabs help to keep the serial output ordered for posterior handling 72. // by the Python script 73. Serial.print(voltageControl); 74. Serial.print("\t"); 76. Serial.print(temperature2); 77. Serial.print("\t"); 79. Serial.print(setTemp); 80. Serial.print("\t"); 82. Serial.print(temperature1); 83. Serial.print("\t"); 85. Serial.print(millis()); 86. Serial.print("\t"); 87. // The if statement compare the temperature obtained from the AD22100 88. // to the set temperature depending on the hysteresis value 89. // if is lower than the set temperature, a high output is set for the 90. // heating pads while the Peltiers are kept low 91. // A similar if statement is performed in case the cooling is required 92. // Finally, if the temperature is within the hysteresis, the OpenTCC 93. // does not open any of the N-channel MOSFETs 94. **if** (temperature2 < setTemp - 2\*histeresis){ 95. digitalWrite(heatingPads, HIGH); 96. digitalWrite(peltiers, LOW); 97. Serial.print("heating"); 98. } **else** **if** (temperature2 > setTemp + 2\*histeresis){ 99. digitalWrite(heatingPads, LOW); 100. digitalWrite(peltiers, HIGH); 101. Serial.print("cooling"); 102. } **else** { 103. digitalWrite(heatingPads, LOW); 104. digitalWrite(peltiers, LOW); 105. Serial.print("histeresis"); 106. } 108. Serial.println(); 109. delay(1000); 110. // This function calls a function to read any serial input 111. // which comes from the Python GUI 112. getPythonTemp0(); 113. } 115. // New serial read parsing function. Forced to change to allow python3 write to 116. // serial command in PySerial\_loop 118. **void** getPythonTemp() { 119. **static** **char** buffer[32]; 120. **static** **size\_t** pos; 121. **if** (Serial.available()) { 122. **char** c = Serial.read(); 123. **if** (c == '\n') {  // on end of line, parse the number 124. buffer[pos] = '\0'; 125. **float** value = atof(buffer); 126. Serial.print("received: "); 127. Serial.println(value); 128. setTemp = value; 129. pos = 0; 130. } **else** **if** (pos < **sizeof** buffer - 1) {  // otherwise, buffer it 131. buffer[pos++] = c; 132. } 133. } 134. } 136. // Old serial read function python2.7 write 137. **void** getPythonTemp0() 138. { 139. **while** (Serial.available()) 140. { 141. **double** input = Serial.parseFloat(); 142. setTemp = input; 143. **break**; 144. } 145. } |

**Code 2:** PythonGUI.py

|  |
| --- |
| 1. #!/bin/python 3. # This is a simple application written in Python and TKinter. 4. # Sending and receiving between Arduino OpenTCC and PC 5. # and saving a continuosly into a new file everyday. 6. # Necessary libraries for the Operative system, timestamp and GUI 7. **import** re 8. **import** os 9. **import** time 10. **from** Tkinter **import** \* 11. **import** serial 12. # Initialized serial data and output filtered data to an empty string 13. serial\_data = '' 14. filter\_data = '' 15. # Timing between the update of the serial information 16. update\_period = 15 17. serial\_object = None 18. gui = Tk() 19. # Initialized por and baud rate to empty string to allow to change them 20. # in case a different controller is used 21. current\_port = None 22. current\_baud = None  25. **def** connect(): 26. version\_ = button\_var.get() 27. **print** version\_ 28. **print**("attempting to connect") 29. **global** serial\_object 30. **global** current\_baud 31. **global** current\_port 32. current\_port = port\_entry.get() 33. current\_baud = baud\_entry.get() 35. **try**: 36. **if** version\_ == 2: 37. **try**: 38. serial\_object = serial.Serial('/dev/tty' + 39. str(current\_port), current\_baud, timeout =1) 41. **except**: 42. **print** "Cant Open Specified Port" 43. **elif** version\_ == 1: 44. serial\_object = serial.Serial('COM' + 45. str(current\_port), current\_baud) 47. **except** ValueError: 48. **print** "Enter Baud and Port" 49. **return**  52. **def** get\_data(serial\_object): 53. **try**: 54. serial\_data = serial\_object.readlines() 56. **except** Exception as e: 57. **raise**(e) 59. **if** len(serial\_data) == 0: 60. **raise** ValueError('Warning: misformatted data') 61. most\_recent = serial\_data[-1].strip('\n').strip('\r') 62. split\_data = most\_recent.split('\t') 63. #if len(split\_data) != 5: 64. #   raise ValueError('Warning: misformatted data') 65. **return**(split\_data, most\_recent)  68. # Line which saves the information from the serial communication to a file 69. # It adds the timestamp to each line to allow posterior analysis of the 70. # temperature curves 71. **def** save\_data(save\_dir, text, port, baud): 72. lineall=port + "\t" + baud + "\t" 73. lineall= lineall + time.strftime('%Y-%m-%d\t%H:%M:%S\t') + text 74. with open(os.path.join(save\_dir,time.strftime('%Y%m%d')), 'a') as f: 75. f.write(lineall + '\n')  78. **def** send(): 79. send\_data = data\_entry.get() 81. **if** **not** send\_data: 82. **print** "Sent Nothing" 84. serial\_object.write(send\_data) 85. serial\_object.flush() 86. connect()  89. **def** disconnect(): 90. **try**: 91. serial\_object.close() 93. **except** AttributeError: 94. **print** "Closed" 96. gui.quit()  99. # Main function that takes the serial object, check if a folder exist 100. # for the specific date and create it if necessary 101. **def** main(write): 102. **global** serial\_object 103. # set up outputs 104. cwd = os.path.dirname(os.path.realpath(\_\_file\_\_)) 105. dest\_dir = os.path.join(cwd, "ControlChamber") 106. **if** **not** os.path.exists(dest\_dir): 107. os.makedirs(dest\_dir) 109. # check if connection exists 110. **if** serial\_object **is** None: 111. **print**("no serial object") 112. **else**: 113. **if** **not** write: 114. **return** 115. # read in most recent data 116. **try**: 117. split\_data, serial\_data = get\_data(serial\_object) 118. **except** ValueError: 119. **print**("skipping") 120. **return** 121. save\_data(save\_dir=dest\_dir, 122. text=serial\_data, 123. port=current\_port, 124. baud=current\_baud) 125. text.insert(END, split\_data) 126. text.insert(END,"\n") 127. text.see(END) 128. **try**: 129. valueInside["text"] = split\_data[1] 130. valueSet["text"] = split\_data[2] 131. valueOutside["text"] = split\_data[3] 132. valuePWM["text"] = split\_data[0] 134. **except** : 135. **pass** 137. #TKINTER GUI PARAMETERS 139. gui.title("OpenTCC") 141. #frames 142. frame\_1 = Frame(height = 200, width = 390, bd = 3, relief = 'groove').place(x = 5, y = 5) 143. frame\_2 = Frame(height = 200, width = 390, bd = 3, relief = 'groove').place(x = 5, y = 200) 144. text = Text(width = 46, height = 5) 145. text.place(x = 10, y = 60) 147. #Labels 148. baud   = Label(text = "Baud").place(x = 180, y = 10) 149. port   = Label(text = "Port").place(x = 280, y = 10) 151. data1\_ = Label(text = "Chamber Temperature:").place(x = 15, y= 220) 152. data2\_ = Label(text = "Set Temperature:").place(x = 15, y= 250) 153. data3\_ = Label(text = "Outside Temperature:").place(x = 15, y= 280) 155. contact = Label(text = "c.sanchez2@nuigalway.ie").place(x = 150, y = 380) 157. #progress\_bars 158. valueInside = ttk.Label() 159. valueSet = ttk.Label() 160. valueOutside = ttk.Label() 161. valuePWM = ttk.Label()  164. #Entry 165. data\_entry = Entry() 166. data\_entry.place(x = 200, y = 340) 168. baud\_entry = Entry(width = 10) 169. baud\_entry.insert(END, '9600') 170. baud\_entry.place(x = 180, y = 25) 172. port\_entry = Entry(width = 10) 173. port\_entry.insert(END, 'ACM0') 174. port\_entry.place(x = 280, y = 25) 176. #radio button 177. button\_var = IntVar() 178. radio\_1 = Radiobutton(text = "Windows", variable = button\_var, value = 1).place(x = 80, y = 20) 179. radio\_2 = Radiobutton(text = "Linux", variable = button\_var, value = 2).place(x = 20, y = 20) 180. button\_var.set(2) 182. #button 183. button1 = Button(text = "New Temperature", command = send, width = 18).place(x = 15, y = 340) 184. connect\_button = Button(text = "Connect", command = connect) 185. connect\_button.place(x = 15, y = 160) 186. disconnect\_button = Button(text = "Disconnect", command = disconnect) 187. disconnect\_button.place(x =280, y = 160) 188. gui.geometry('400x400') 189. # draw once to start things off 190. #mainloop 191. text.place(x = 10, y = 60) 192. valueInside.place(x = 210, y = 220) 193. valueSet.place(x = 210, y = 250) 194. valueOutside.place(x = 210, y = 280) 196. #connect\_button.invoke() 198. **if** \_\_name\_\_ == "\_\_main\_\_": 199. oldtime = time.time() 200. new\_time = 0 201. **while** True: 202. #        print(serial.Serial('/dev/tty' + str("ACM0"), 9600 , timeout =5)) 203. gui.update() 204. newtime = time.time() 205. **if** newtime - oldtime >= update\_period: 206. main(write=True) 207. oldtime = newtime 208. button1\_flag = True 209. newtime = time.time() |

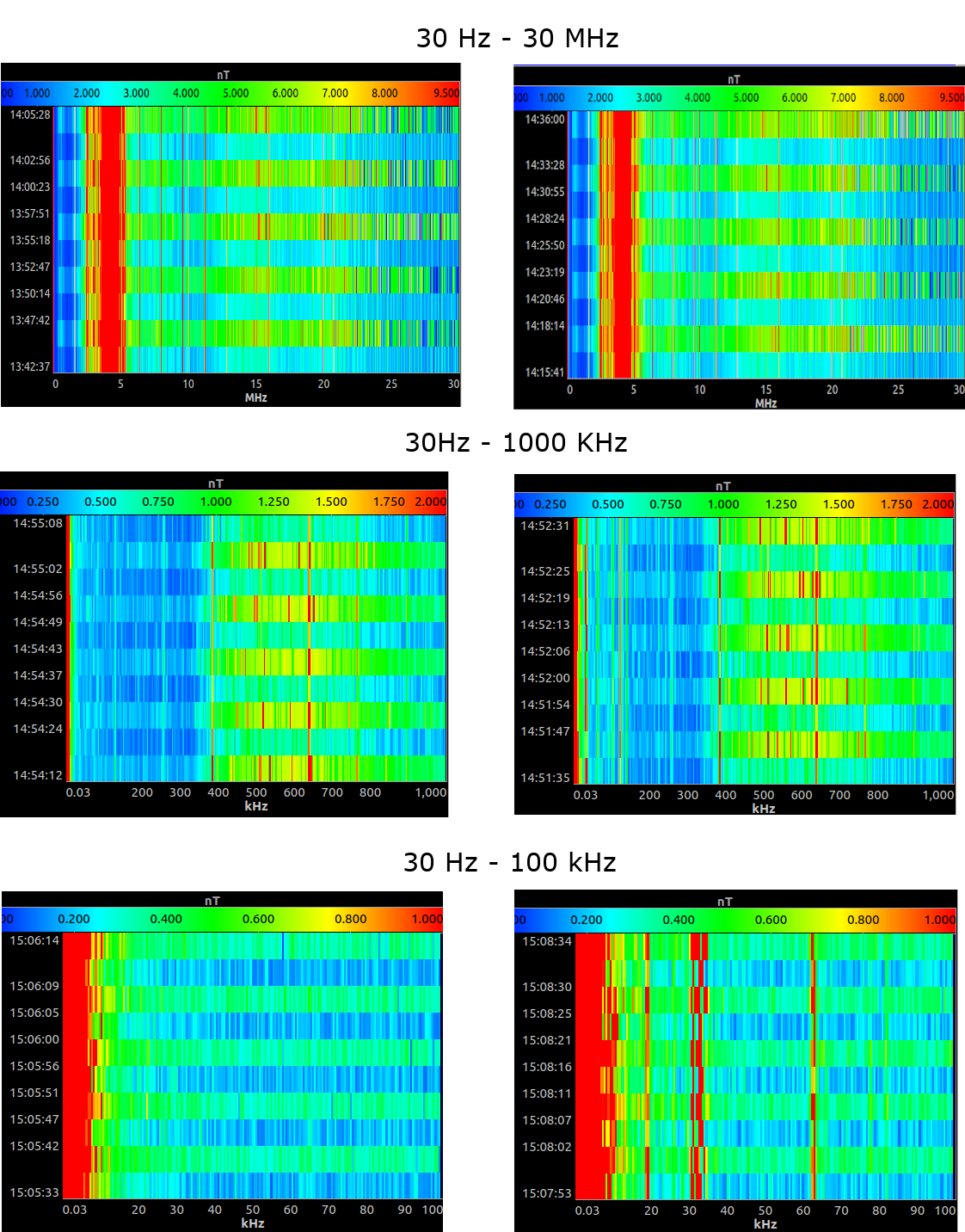
**Code 3:** Loop.py

|  |
| --- |
| 1. #!/bin/python 3. # Sending and receiving data from the OpenTCC Arduino. 4. # Loop to analyze time required to achieve 5. # set temperature for 20, 30, 40 and 55 6. **import** os 7. **import** time 8. **import** serial 9. **from** math **import** isclose 11. serial\_data = '' 12. filter\_data = '' 13. update\_period = 5 15. current\_port = "ACM1" 16. current\_baud = 9600 18. serial\_object = serial.Serial('/dev/tty' + str(current\_port), 19. current\_baud, timeout=1) 21. temp\_loop = [20, 30, 20, 40, 20, 55, 22. 20, 30, 40, 30, 55, 23. 30, 40, 55, 40] 25. cwd = os.path.dirname(os.path.realpath(\_\_file\_\_)) 26. save\_dir = os.path.join(cwd, "loop\_temperatures") 28. **if** **not** os.path.exists(save\_dir): 29. os.makedirs(save\_dir)  32. # Get data from serial object 33. **def** get\_data(serial\_object): 34. **try**: 35. serial\_data = serial\_object.readlines() 37. **except** Exception as e: 38. **raise**(e) 39. **if** len(serial\_data) == 0: 40. **raise** ValueError('Warning: misformatted data') 42. most\_recent = serial\_data[-1].decode().strip('\n').strip('\r') 43. split\_data = most\_recent.split('\t') 44. **print**(split\_data) 45. **return**(split\_data)  48. # Filter data to use only temperature set and temperature chamber 49. **def** filter\_data(split\_data): 50. **try**: 51. valueInside = split\_data[1] 52. valueSet = split\_data[2] 54. **return** (float(valueInside), float(valueSet)) 56. **except** Exception as e: 57. **print**(e) 58. **pass**  61. **def** send\_new\_temp(temp): 62. **print**("sending new temperature %i" % temp) 63. temp\_string = str(temp) + "\n" 64. serial\_object.write(temp\_string.encode()) 65. # serial\_object.flush() 66. # connect()  69. # Loop over temperatures jumps and 70. # set time on each temperature once is attained 71. **def** loop\_temp(): 72. valueInside, valueSet = filter\_data(get\_data(serial\_object)) 73. **print**(valueInside) 74. **print**(valueSet) 75. **for** temp **in** temp\_loop: 76. **print**(temp) 77. send\_new\_temp(temp) 78. **print**('current set temp: %i' % temp) 79. reached\_temp = False 80. got\_enough\_data = False 81. **while** (**not** reached\_temp) **or** (**not** got\_enough\_data): 82. **if** isclose(temp, valueInside, abs\_tol=0.4): 83. **if** **not** reached\_temp: 84. at\_temp\_time = time.time() 85. reached\_temp = True 86. **print**(time.time() - at\_temp\_time) 87. **if** (time.time() - at\_temp\_time) > 600: 88. # Indicates time to stay in the set temperature 89. **print**('time to go to the next temp!') 90. got\_enough\_data = True 91. valueInside, valueSet = filter\_data(get\_data(serial\_object)) 92. save\_data(save\_dir=save\_dir, 93. valueInside=valueInside, 94. valueSet=valueSet)  97. # Write to serial the data obtained from loop\_temp 98. **def** write\_serial(serial\_object, send\_data): 99. serial\_object.write(send\_data) 100. serial\_object.flush() 102. # Save data function 103. **def** save\_data(save\_dir, valueInside, valueSet): 104. # append timestamp 105. text = "{valueInside}\t{valueSet}".format(\*\*locals()) 106. lineall = time.strftime('%Y-%m-%d\t%H:%M:%S\t') + text 107. with open(os.path.join(save\_dir, time.strftime('%Y%m%d')), 'a') as f: 108. f.write(lineall + '\n') 110. **def** main(): 111. **global** serial\_object 112. **global** temp\_loop 113. **global** start\_time 114. start\_time = time.time() 115. # Check for data 116. #    valueInside, valueSet = filter\_data(get\_data(serial\_object)) 118. # check loop\_temp and write serial if 119. # attained temperature for more than 30 minutes 120. # repeat this loop 5 times 121. Times = 5 122. **for** i **in** range(Times): 123. **print**('Loop number %i' % i) 124. loop\_temp()  127. **if** \_\_name\_\_ == "\_\_main\_\_": 128. main() |

1. **EMI characterization**
   1. **ATX ON *vs* ATX OFF**

The measurement devices was placed at 10 cm from the ATX power supply (outside the chamber)

ATX OFF ATX ON

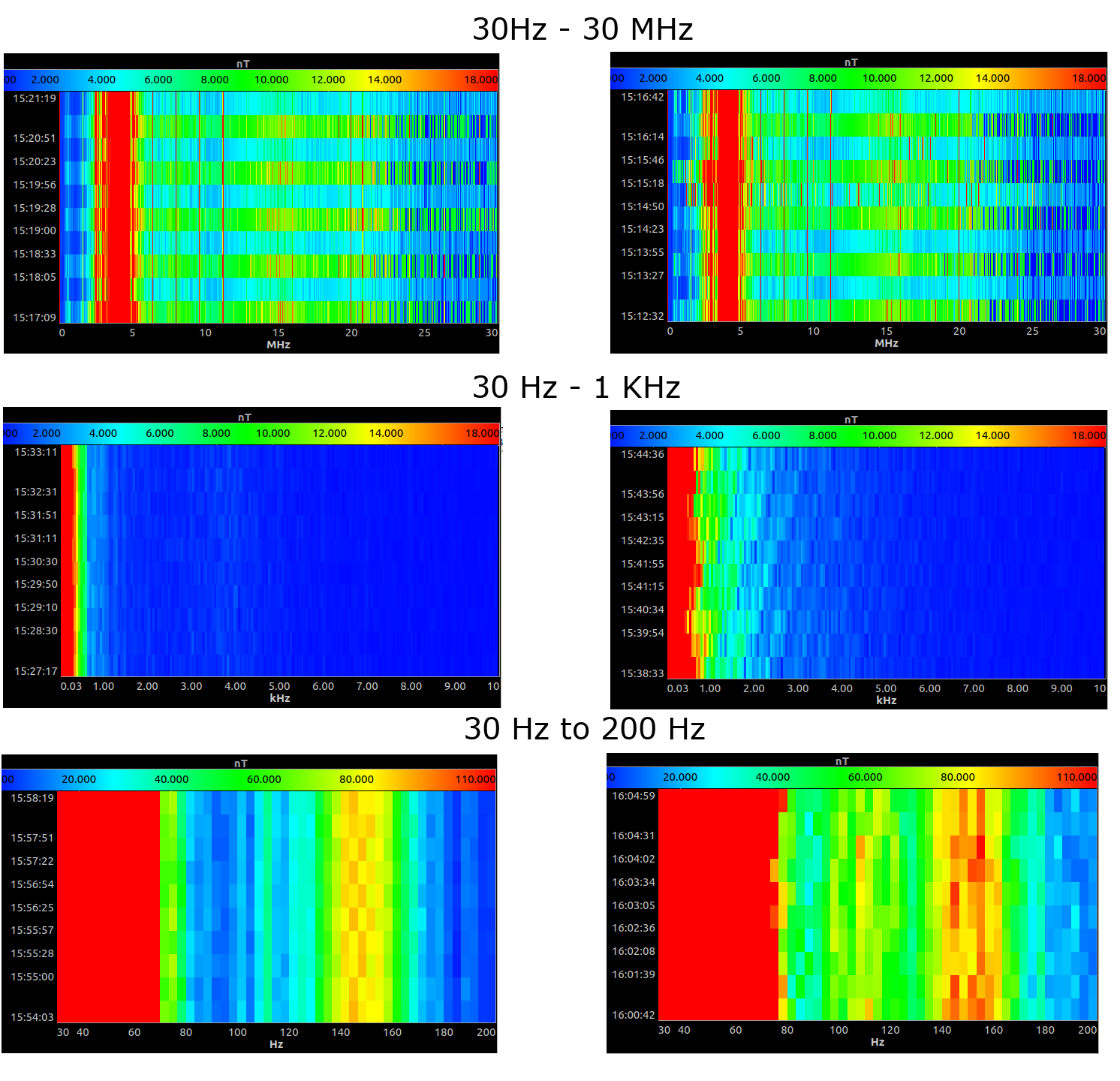


* 1. **Motors OFF *vs* ATX ON Motors ON**

The measurement devices was placed inside the chamber to detect possible radiation from wires inside the chamber corresponding to the BLDC motors. The measurement device was place inside the chamber equidistant from the walls.

Comparison only inside the chamber

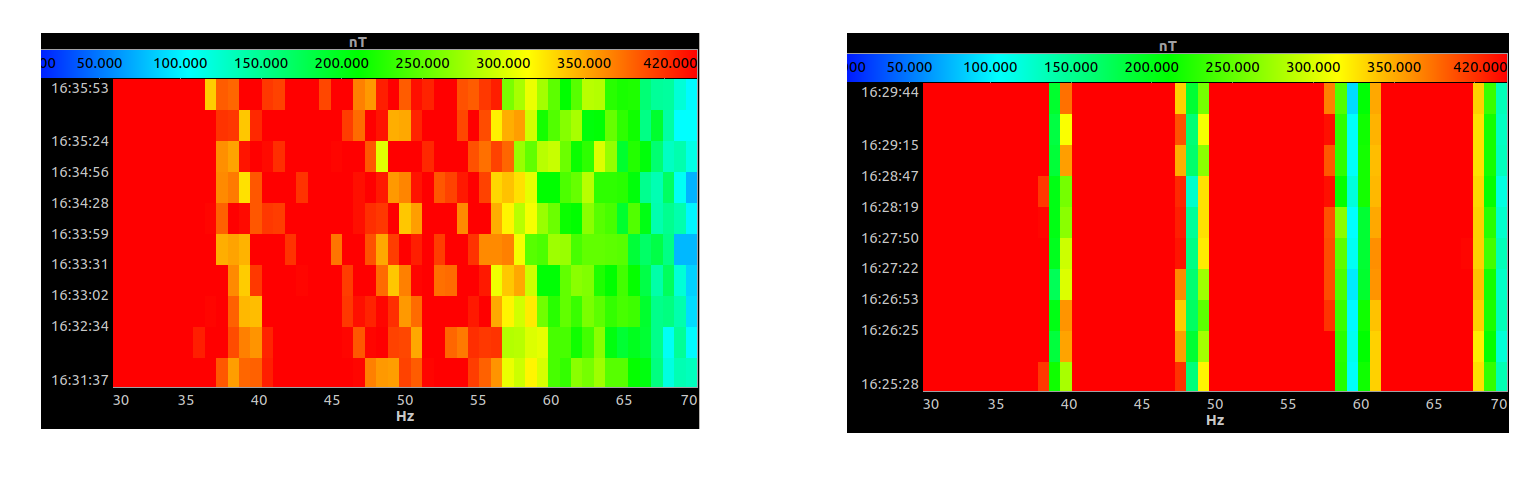
ATX ON Motors OFF ATX ON Motors On



Comparison between inside and outside

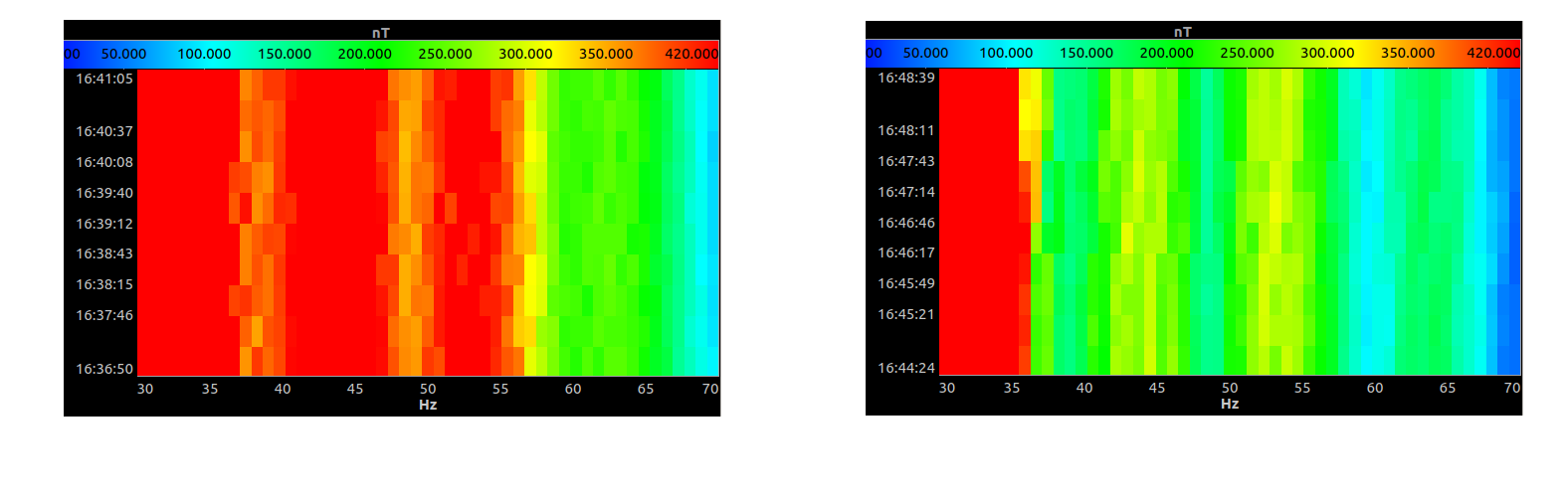
Motors ON

Inside Outside



Motors OFF

Inside Outside



1. **Temperature profiles with water volume**
   1. **Heating without additional volume vs heating with 6 L of water.**

**A close up of a map

Description automatically generated**

* 1. **Opening chamber without additional volume vs heating with 6 L of water**

