Åbo Akademi

Multidimensional Sensing Techniques

Assignment N°2: Audio sensors

Cédric Léonard, 06/10/2021, student ID: 2100775

Introduction

This second Multidimensional Sensing Techniques assignment aims to get familiarized with audio sensors. After connecting a second microphone to our computer we will compute mean and variance of input volumes in order get new information and try to detect defective microphone. You can find additional informational on the following GitHub repository.

What you did?

Set up my work environment

After reading the assignment instructions, the first task was to set up a work environment in order to run the python script provided on Moodle. I chose to setup a virtual environment for its ease of use and in order to not generate trouble with python packages on my whole computer. The bash commands below correspond more or less accurately what I have done.

```
# Install a python virtual environment (virtualenv)
pip install virtualenv
# Create a new virtual environment
virtualenv --python /My/Path/To/Python/python.exe venv
# Activate this virtual environment
source ./venv/Scripts/activate

# Install every packages needed
pip install numpy
pip install keyboard
pip install PyQt5
# I had trouble with PyAudio, I had to go with pipwin
pip install pipwin
pipwin install PyAudio
```

After successfully running **audiosensors.py**, I used *WOW Mic* in order to connect my smartphone microphone to my computer. Following a tutorial I downloaded the *WOW Mic Client* on my computer and the *WOW Mic App* on my phone. As I chose to connect them by

WIFI I then, simply had to connect them to a same IP address and start recording data from my phone. Hence, I could see my phone microphone input signal in the window:

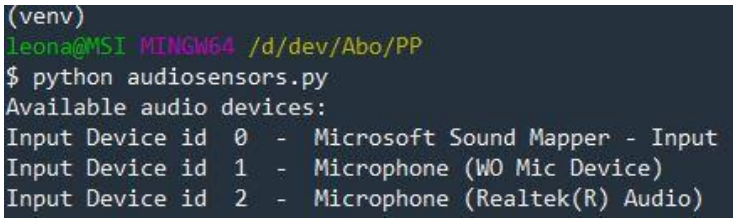

Figure N°1: Console screenshot showing the available audio devices

On Figure N°1 we can see all the available devices detected by **audiosensors.py**. For the whole assignment, we won't use the device N°0 as it appeared to be a Microsoft feature and simply outputs the same data than device N°1 which is my phone microphone. When computing means and variances, we will be careful to avoid considering this first device as it would be as giving twice its importance to my phone microphone.

We could also see that I'm running the script inside my virtual python environment as the `(venv)` line on top of the picture indicates (*venv* is the name of my environment).

How and why you did it?

Q1: Read and understand the code

The first instruction was to read and understand the code and to display the volume for each sensor. That was a simple task and I will not go into the detail. But it is important to note that there are 2 different functions, `log_sound()` and `MainThread()`. `log_sound()` is a function called in a thread and there is one thread for each audio device. Therefore, inside the `log_sound()` function, we can easily access each audio device feature and display data on the corresponding **Qt** label (on the GUI). Hence, `log_sound()` is the online place in the code where we can modify the text of each individual label device.

On the other hand, `MainThread()` is obviously the function whose job can be likened to the main function of our program. In `MainThread()` we can access labels for combined data, i.e. mean and variance.

Q2: Combined mean and variance

Then we were asked to compute the combined mean and variance of the current signal. As explained above, these computations will be done in the `MainThread()` function. We first want to access the current volume of each audio device. This is possible through the `buffer\[][]` global variable. This variable is filled by the multiple `log_sound()` threads and is global in order to access these values inside the main thread. To dive a bit more in details, the list `buffer[0]` contains every last 100 volumes of the device N°0 (This limitation of the buffer width is done in `MainThread()` with a simple for loop).

After understanding how we can access each data the job is simple: compute the mean and variance of the dataset consisting in every last value of each audio device, i.e. each

current volume. The code below corresponds to the calculations for the combined mean and variance.

```
# --- Mean computation ---
mean_value = 0
# For every audio devices, except ID N°0 (Microsoft stuff)
for i in range(1, nb_audio_devices):
   mean_value = mean_value + buffer[i][-1]
# After the sum of every devices volume, finish the computation by dividing by the number of terms
mean_value = mean_value / (nb_audio_devices-1)

# --- Variance computation ---
var_value = 0
# Sum of square distances between each data point to the mean
for i in range(1, nb_audio_devices):
   var_value = var_value + (buffer[i][-1] - mean_value)**2
var_value = var_value / (nb_audio_devices-1)
```

Once the calculations are done, we update the text of the mean and variance labels.

Q3: Circular buffer and individual means and variances

We then were asked to compute each audio device individual statistic over the last 100 samples. This was the main point of the assignment and has been realized inside the `log_sound()` function. As the different threads are running in parallel, the limitation of the buffer to the last 100 samples is effective even if we write the code in the `log_sound()` function.

The computations were realized with the use of **Numpy**. We first cast the list containing each sample (i.e. `buffer[index]`) into a `np.array` and then use the `.mean()` and `np.var()` functions. This is rather simple but effective and doing that in the `log_sound()` thread allows us to directly display each individual statistic on each audio device label.

Q4: Source defects detection

Last instruction was about detecting when one of the audio device was defective. This was quite a broad and hard question and the solution implemented is not really satisfying. We are trying to detect if a microphone volume stays low, even when the other devices detect more sound. I chose to implement this with 3 conditions:

- The individual mean of the device must be low (for the last 100 samples);
- The combined variance must be high, which means the current volume of the devices are broadly distributed;
- And finally, the distance between the individual mean and the combined mean must be great.

If the 3 conditions above are met the current device is considered defected and we update its label. I first tried to change the background color of the label in red but the

`label.setStyleSheet("background-color: red")` method is not thread safe. So, the solution adopted is to simply add some characters "----" before the label text.

The results of your work

During the different parts of the assignment I verified my calculations by printing the data used for the computations (in the combined mean label).

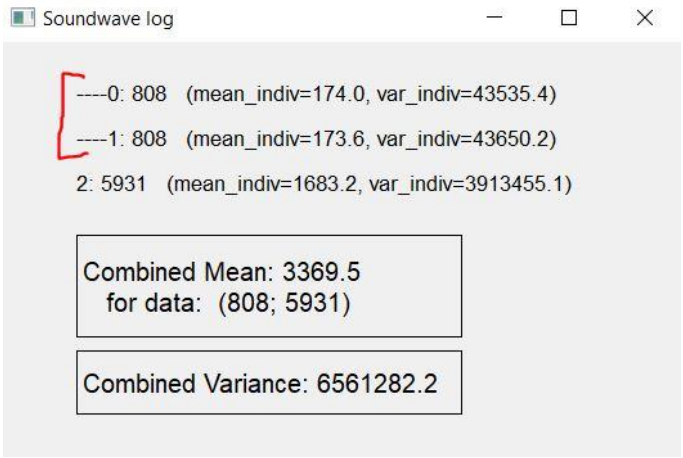

Figure N°2: GUI screenshot showing the detection of 2 defective microphones ID 0 and 1

The Figure N°2 shows every element implemented for this assignment. Devices N°0 and 1 outputs the same data and corresponds to my phone which was far away from my computer microphone (device N°2) under a pillow and therefore wanted "defective".

The 3 first labels show the 3 devices identified by **PyAudio**, their ID, their current volume and their individual statistics over the last 100 samples (mean and variance). Then the 2 labels with the thin black border display the combined information, as explained earlier, these combined statistics do not include the device N°0 (because it's a clone of device N°1). Moreover, the combined mean label also includes additional information in parenthesis: current volumes used to compute the combined mean and variance.

About the last assignment instruction, defect detection, Figure N°2 shows that device N°0 and N°1 are detected as defective. We can manually verify that: Their individual means are low *(< 500)*, the combined variance is huge *(> 50.000)* and the difference between their mean and the combined mean is significative: *(3369 - 174 > 1000)*.

How your solution could be improved/generalized

I said earlier that I displayed the current volumes of each device in the combined mean label. That may be useless but this is because my solution suffers a lack of synchronization between threads. I had trouble verifying my calculations because the computed combined mean in the `MainThread()` did not correspond to the actual displayed individual volume in the `log_sound()` threads. This is not the case of the screenshot (Figure N°2) but it sometimes happens and I'm not sure why.

If we want to do real time application with this solution, this should be improved with, for example, forced synchronization between threads or the globalization of the calculations in `MainThread()`.

What did you learn?

I discovered how we can create a GUI and threads in Python and it was really formative to go through different documentations and tutorials in order to understand the script. I also spend a bit of time trying to implement a virtual python environment for Windows which was also really interesting.

Did anything surprise you? / Did you find anything challenging? Why?

How hard it can be to try detecting defection. Of course, it depends on your environment, your devices, what you call defection, etc., but if you don't have a really concrete definition of what you "are looking for" it's really hard to implement a definition of defection. This is also quite hard to test.

Did you find anything satisfying? Why?

The overall assignment was quite satisfying, but screaming in front of your computer just to see numbers increase is funny and I spend some time trying to understand the delays and values of microphone outputs.