FEATURES OF ANALYSIS OF MULTICHANNEL AUDIO SIGNALS

Tetiana Martyniuk  
Doctor of technical sciences, professor, professor department of computer engineering  
Vinnytsia National Technical University, UKRAINE

Maksym Mykytiuk  
Postgraduate student, department of computer engineering  
Vinnytsia National Technical University, UKRAINE

Mykola Zaitsev  
Postgraduate student, department of computer engineering  
Vinnytsia National Technical University, UKRAINE

Summary. The rapid growth of audio content has led to the need to use tools for analysis and quality control of audio signals using software and hardware and modules. The fastest-growing industry is software and programming languages. The Python programming language today has the most operational and visual capabilities for working with sound. When developing programs for computational signal analysis, it provides the optimal balance of high and low-level programming functions. Compared to Matlab or other similar solutions, Python is free and allows you to create standalone applications without the need for large, permanently installed files and a virtual environment.

Keywords: audio signals, python, pyAudioAnalysis, multichannel signal, analysis audio signal.

Introduction. The peculiarity of multi-channel signals is that they transmit audio information to the viewer not through one channel, but through several. This improves the quality of perception of sound information due to the impression that the sound comes from several sources [1].

Studying the peculiarities of the transmission of audio signals on many channels with the help of computer simulation is a very important task today. This is due to the difficulty of decoding the multi-channel analog signal and then its correct digital processing [2].

Among the well-known software tools that are effectively used for computer simulation of complex processes and signals, a special place is occupied by the Python programming language [3].

The purpose of this work is to analyze multichannel signals using Python software.

Formulation of the problem. There are two solutions to this problem. The first is the division of the signal into several channels in the first stage after its download. However, this can lead to inaccuracies and distortions in further work with it [1]. The second option is to download it in its current form and get it completely on the
dependency chart. Then, using the spectrogram, programmatically divide it, taking into account the sensitivity threshold [4,5].

The pyAudioAnalysis library provides characteristics that are usually unique compared to other related libraries [2,3]:

1. Analysis of the general characteristics and properties of the interconnected sound signal to form complete decisions on the classification and segmentation of sound.

2. To solve widely used problems of audio analysis, both the most modern and basic methods are used.

3. For some controlled tasks, pre-trained models are offered (for example, classification of language and music, classification of music genre and detection of events in the film).

4. All provided functionalities are written using clear and simple code so that conceptual algorithmic steps can be clearly presented in the context of the learning process.

*Practical implementation.* In the process of computer modelling, it is possible to qualitatively determine and analyze such parameters as the speed of sound signals, as well as their segmentation [1,5].

In fig. 1 shows the features of information presentation associated with the rate of sound flow, which contains automatic induction of beats, i.e., the task of determining the speed of musical beats over time is a very important task, especially in the case of programs for searching music information [4, 5]. In the considered library pyAudioAnalysis [2,3] the direct approach to the calculation of the rate is applied. It adopts a local maximum detection procedure applied to a set of short-term sequences of features. Also in fig. 2 shows an example of calculating the total histogram of time distances between successive local maxima and its maximum element, which corresponds to the most dominant time distance between successive strokes. The obtained value is used to calculate the impact rate per minute. In addition to the value of the stroke rate per minute, the ratio of the maximum value of the histogram to the total sum of the values of the histogram, which corresponds to the total "dominance" of the detected beating speed, is used as a sign [2].

Fig. 1. Example of a stroke histogram
The audio segmentation procedure focuses on splitting a continuous audio signal into segments of homogeneous content. The term "homogeneous" can be defined in different ways, so there is difficulty in providing a general definition. The library offers algorithmic solutions for two common subcategories of audio segmentation [6]:

The first type of segmentation contains algorithms that take a certain type of "prior" knowledge, such as a pre-trained classification scheme. For this type of segmentation, the library provides a common segmentation of a fixed size, which is shown in fig. 3 [6].

The second type of segmentation is either unsupervised or supervised. In both cases, prior knowledge of the audio content classes involved is not used. Typical examples are the elimination of silence, diarisation of speakers and sound reduction [6]. In fig. 3 shows the segmentation of the selected audio signal.

Multichannel implementation of stereo signals is performed through 2D arrays [6]. In the following example, the data_wav array has two columns, one for each channel. By default, the left channel is always the first and the second is the right channel. Similarly, you can develop solutions for more channels.

Code snippet [6]:

![Detection of local maxima to obtain impact indicators](image1)

![Example of segmentation](image2)
# Handling stereo signals

```python
fs_wav, data_wav = wavfile.read("data/stereo_example_small_8k.wav")
time_wav = np.arange(0, len(data_wav)) / fs_wav
plotly.offline.iplot({ "data": [go.Scatter(x=time_wav,
        y=data_wav[:, 0],
        name='left channel'),
        go.Scatter(x=time_wav,
        y=data_wav[:, 1],
        name='right channel')]
})
```

**Conclusion.** Thus, given the existing software capabilities of software analysis of the single-channel audio signal, it became possible to develop new solutions that can be applied when working with multi-channel systems.

Given the principles of working with a single-channel signal, you can improve the existing capabilities by making certain settings in the demo program. First of all, there is a need to install basic libraries for the software environment, and only then select the software modules to work with the type of signal to be studied.

**References:**


