

Efficient Combination of Multi-Track Audio Files with Speaker Segmentation for Transcription Services

Michael Baltazar Chambi

Information and Communication technology, Judiciary of Tanzania, Dar es salaam, Tanzania

ABSTRACT

This research focuses on a method for transcription by swiftly segmenting multi-track audio files based on speaker identity and combining them together to provide spoken information assigned to individual speakers. Using the FFmpeg package, the approach combines multiple audio tracks to increase transcription accuracy in multi-speaker scenarios. It also incorporates speaker identification based on file input order. Application of this solution highlights its significance for transcribing services, especially in legal and court-related systems.

Keywords: FFmpeg, Audio Combination, Science, Software Development, NodeJs, Information and Communication Technology.

1. INTRODUCTION

Transcription of multi-speaker audio recordings is increasingly critical in various fields, especially in legal proceedings, interviews, and meetings. Accurate transcription requires clear segmentation of audio data by speaker identity. Current methods may involve manual effort in identifying speaker segments, which can lead to errors and inefficiencies.

This paper proposes a systematic approach to automatically combining multiple audio files into a single output while ensuring that the resulting transcription correctly attributes each speaker's words to the appropriate speaker. The solution involves using the FFmpeg library for multi-track audio file combination and a JSON-based structure to retain metadata for speakers.

2. PROBLEM DEFINITION

In multi-speaker audio recordings, a transcription system needs to:

1. Combine multiple audio tracks into a single output.
2. Maintain an association between each speaker and their respective audio segment in the final transcript.
3. Ensure the process scales efficiently to multiple speakers while reducing the manual effort involved in segmenting and labeling audio data.

Existing solutions for audio combination, such as FFmpeg's default merge functionality, may result in audio overlap, which is unsuitable for transcription purposes. Furthermore, many systems do not handle the speaker identity efficiently, leading to inaccurate transcriptions that do not clearly distinguish between different speakers.

3. METHODOLOGY

3.1 System Architecture

The proposed solution comprises three core steps:

1. Audio File Combination: Using FFmpeg to concatenate multiple .wav files in a sequential manner to prevent overlapping audio.
2. Speaker Identification: Assigning each speaker's name based on the order of the input audio files and embedding these speaker identities into the transcription.
3. JSON-based Metadata: Storing speaker information in a structured JSON file, which is used during transcription to ensure correct speaker attribution.

3.2 Audio Combination using Ffmpeg

The FFmpeg command-line tool is widely used for processing audio and video files. However, when combining multiple audio tracks, the default amerge filter combines audio into multiple channels, which can result in overlapping content. To address this, the concat method of FFmpeg is used, which concatenates multiple audio files sequentially.

The command used is:

```
ffmpeg -f concat -safe 0 -i file_list.txt -c copy output.wav
```

Where file_list.txt contains the paths to the audio files to be combined, ensuring they are played sequentially without overlap

3.3 Transcription and Speaker Tagging

After combining the audio tracks, the transcription is generated either manually or through an automated transcription service. The transcript is then processed by mapping each segment of the transcript to a speaker based on the order of the input audio files.

For example, given the following audio files:

1. channel_1_output_1725535819220.wav
2. channel_2_output_1725535819220.wav

And the transcript:

1. "Chaneli ya kwanza tunaweza kurekodi."
2. "Chaneli ya pili tunaanza kurekodi."

The final output would be:

channel_1_output_1725535819220: Chaneli ya kwanza tunaweza kurekodi

channel_2_output_1725535819220: Chaneli ya pili tunaanza kurekodi

3.4 JSON Metadata

A JSON file is generated alongside the combined audio file. This file contains metadata that includes the court name (if applicable), the transcription parameters, and the list of speakers.

Example

```
{
  "court": "IJC DODOMA",
  "TranscriptionParams": {
    "MultiTrack": true,
    "SpeakersList": [
      "channel_1_output_1725535819220",
      "channel_2_output_1725535819220"
    ]
  }
}
```

```
}  
}
```

4. IMPLEMENTATION

The system is implemented using Node.js and the fluent-ffmpeg library. The server processes incoming requests to combine audio files, generates a transcription, and returns the transcription with appropriate speaker tags.

```
const fs = require('fs');  
const path = require('path');  
const ffmpeg = require('fluent-ffmpeg');  
  
// Function to combine audio files and prepare transcription  
exports.combineAndPrepareForTranscription = (req, res) => {  
  // Define the paths for audio files and output  
  const folderPath = path.join(__dirname, '../audio-recorder/controllers/recordings');  
  const finalFolderPath = path.join(__dirname, '../combined');  
  
  // Check if the recordings folder exists  
  if (!fs.existsSync(folderPath)) {  
    return res.status(500).json({ error: 'Recordings folder does not exist' });  
  }  
  
  // Read the audio files and combine them  
  const audioFiles = fs.readdirSync(folderPath)  
    .filter(file => file.endsWith('.wav'))  
    .map(file => path.join(folderPath, file))  
    .sort((a, b) => fs.statSync(a).mtime - fs.statSync(b).mtime);  
  
  if (audioFiles.length < 2) {  
    return res.status(400).json({ error: `You need at least two audio files to combine. Found ${audioFiles.length} file(s).` });  
  }  
  
  // Ensure output directory exists  
  if (!fs.existsSync(finalFolderPath)) {  
    fs.mkdirSync(finalFolderPath, { recursive: true });  
  }  
  
  const outputFilePath = path.join(finalFolderPath, `combined_sequential_output_${Date.now()}.wav`);  
  const jsonFilePath = path.join(finalFolderPath, `transcription_params_${Date.now()}.json`);  
  
  // Create a temporary file list for FFmpeg concat
```

```
const fileListPath = path.join(finalFolderPath, `file_list_${Date.now()}.txt`);
const fileListContent = audioFiles.map(file => `file '${file}'`).join('\n');
fs.writeFileSync(fileListPath, fileListContent, 'utf-8');

// Execute FFmpeg to combine the audio files
ffmpeg()
  .input(fileListPath)
  .inputOptions(['-f concat', '-safe 0'])
  .output(outputFilePath)
  .on('start', commandLine => console.log(`FFmpeg command: ${commandLine}`))
  .on('end', () => {
    console.log('Audio files combined sequentially.');
```

```
const speakersList = audioFiles.map(file => path.basename(file, '.wav'));

const jsonData = {
  court: "IJC DODOMA",
  TranscriptionParams: {
    MultiTrack: true,
    SpeakersList: speakersList
  }
};

fs.writeFileSync(jsonFilePath, JSON.stringify(jsonData, null, 2), 'utf-8');
res.json({
  message: 'Files combined successfully',
  outputFile: outputFilePath,
  jsonFile: jsonFilePath,
  TranscriptionParams: jsonData
});
})
.run();
};
```

5. RESULTS

The proposed system was tested on a series of multi-track audio files from a legal setting. The resulting transcription correctly identified and labeled each speaker, significantly improving the accuracy of the transcription. The use of FFmpeg's concat method ensured that no overlapping occurred, while the JSON metadata provided essential information for future automated transcription services.

CONCLUSION

This research presents a streamlined method for combining multi-track audio files and preparing them for accurate transcription. By leveraging FFmpeg's concatenation capabilities and JSON metadata, the

system enables efficient speaker identification and transcription in multi-speaker environments, especially in court and legal proceedings. Future work will explore automated integration with transcription services to further improve the efficiency and accuracy of the process.

REFERENCES

1. FFmpeg Documentation. <https://ffmpeg.org/documentation.html>.
2. Node.js Documentation. <https://nodejs.org/en/docs/>.
3. Fluent-FFmpeg Library. <https://github.com/fluent-ffmpeg/node-fluent-ffmpeg>.