DTMF Time-Compression Experiment with `resample_poly`

A Proof-of-Concept for Rapid Audio Transmission

Yassine TAMANI
`github.com/rabbyt3s`

December 21, 2024

Abstract

We present an experiment in which text is encoded as extended DTMF (Dual-Tone Multi-Frequency) audio and then compressed in time (speed-up) using polyphase decimation, before being restored (slow-down) through polyphase interpolation. By relying on `resample_poly` from `scipy.signal`, we implicitly apply lowpass filtering during both decimation and interpolation, reducing aliasing and preserving signal quality. This document provides a brief theoretical overview, an implementation outline, and potential applications for rapidly transmitting short bursts of audio data.

1 Introduction

The goal of this experiment is to:

1. **Encode** a textual phrase into extended DTMF tones,
2. **Compress** (accelerate) the resulting audio by a factor M,
3. **Restore** (slow down) it to recover the original duration,
4. **Decode** the restored audio to retrieve the text.

This serves both an **educational** function—demonstrating decimation/interpolation concepts from DSP—and offers a **proof-of-concept** for transmitting data in very short audio bursts.

2 Background

2.1 Extended DTMF

Traditional DTMF uses 8 frequencies to encode digits (0–9, *, #, A–D). Here, we expand to a larger set of low and high frequencies so we can cover additional characters (A–Z, 0–9, etc.). Each character corresponds to one "low" frequency plus one "high" frequency.

2.2 Decimation (Speed-Up)

Decimation by a factor M:

- Applies a lowpass filter to remove frequencies above the new Nyquist limit,
- Keeps every M-th sample,
- Reduces the sampling rate by factor M.

A naive method might simply pick every M-th sample, risking aliasing if high-frequency content is present. By using `resample_poly(signal, up=1, down=M)`, we get an internal polyphase filter which automatically minimizes aliasing.

2.3 Interpolation (Slow-Down)

To *slow down* by the same factor M:

1. We upsample the signal by M (conceptually, inserting $M - 1$ zeros between each sample),
2. We apply a reconstruction lowpass filter to fill in the gaps.

Again, `resample_poly(signal, up=M, down=1)` handles these steps internally, producing fewer imaging artifacts than naive zero-insertion alone.

2.4 Decoding

Once the audio has been restored to approximately its original duration, we decode via:

1. Bandpass filtering (e.g., 600–2000 Hz),
2. FFT-based detection of two prominent peaks (low freq and high freq),
3. Mapping these to the extended DTMF matrix to recover the intended character.

3 Implementation Outline

3.1 Main Components

- **Encoder** (`encode_phrase`): Generates a WAV file containing the DTMF sequence from a user-provided string.
- **Speed Transform**:
  - `speedup_signal`: Uses `resample_poly(..., up=1, down=M)` for decimation with built-in lowpass.
  - `slowdown_signal`: Uses `resample_poly(..., up=M, down=1)` for interpolation with built-in lowpass.
- **Decoder**:
  - Offline: Reads a WAV file, block-wise FFT to detect frequencies.
  - Live: A `LiveDecoder` class capturing audio from the microphone (`sounddevice`), applying FFT, and detecting characters in real time.

3.2 Example Workflow

1. **Encode** the text "HELLO WORLD" into a 3-second WAV.
2. **Speed Up** by $M = 10$, producing a 0.3-second WAV.
3. **Slow Down** that compressed file by $10\times$, returning it to ≈ 3 seconds.
4. **Decode** via FFT-based peak detection in 600–2000 Hz range.

4 Results

In practice, factors up to about $10\times$ still allow correct decoding of the tones, though minor repetitions or missed characters can occur. Going beyond $10\times$ may require more advanced time-stretch algorithms (e.g., phase vocoder, WSOLA) or the use of shorter DTMF tones initially.

5 Future Work

1. **Non-integer factors**: `resample_poly` supports rational (e.g., 3/2) factor changes too.
2. **Error handling**: Add checksums or repeated tones to mitigate distortions in harsh conditions.
3. **Post-filtering / Duplication Suppression**: Eliminate repeated character detections from overlapping blocks.
4. **More robust detection approach**: Possibly cross-correlation or matched filters instead of a simple peak-finding FFT.

6 Conclusion

By leveraging `scipy.signal.resample_poly` for decimation and interpolation, we incorporate an implicit lowpass filtering stage that reduces aliasing compared to naive approaches. This allows a short "burst" of DTMF data to be transmitted (at, say, 1/10 the normal duration) and later restored to near-original quality. While primarily a proof-of-concept, it demonstrates the fundamentals of multirate DSP and could be extended for specialized audio data transmission.

Code Availability

All relevant scripts can be found on GitHub at:

`https://github.com/rabbyt3s`

These include:

- A menu-based main script for encoding, accelerating/decelerating, and decoding,
- A live decoder class using `sounddevice`,
- The polyphase-based `speedup_signal` and `slowdown_signal` functions.

Author

Yassine TAMANI
GitHub: github.com/rabbyt3s

References

[1] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, 3rd edition, Prentice Hall, 2009.