REPAIR: A Web Application For Audio Quality Enhancement

Christopher Laguna, *Georgia Institute of Technology*

***Abstract*— REPAIR is a web application for audio quality enhancement targeted towards users uploading mobile phone video and audio recordings (which are often of poor audio quality) to the internet. Using REPAIR, users can declip audio, remove noise from audio, filter audio and normalize audio loudness using an interface designed for nonexpert users. This paper formally presents REPAIR, describing the design and implementation of the application.**

***Index Terms*—Audio Quality Enhancement, Web Application, Noise Removal, Declipping, Perceptual Equalizer, Mobile Phone Audio**

I. Introduction

As technology has advanced in the past few decades, the number of audio and video recordings recorded using mobile devices has dramatically increased. Uploading services such as YouTube and SoundCloud ease the process of publically sharing audio and video files. While this has generally increased our ability to document, share and spread experiences, unfortunately, many recordings taken from mobile phones suffer from poor quality audio. Recording artifacts such as clipping are often introduced, external noise sources might conceal the desired audio content, and simple processing such as loudness normalization and equalization might substantially improve the audio quality. Mobile phone audio quality is poor for a variety of reasons: the quality of mobile phone microphones is generally poor, users tend to record in loud and noisy environments such as musical concerts, and correct microphone placement is often overlooked or impossible (e.g., live concerts). Mobile phone users need a way to clean up the audio of recordings they take.

This paper presents REPAIR (Repair and Enhance Phone Audio on the InteRnet), a web application for audio quality enhancement. REPAIR includes the following audio processing modules: declipping, stationary noise removal, perceptually motivated equalization, and loudness normalization. The interface is designed for users without audio expertise. REPAIR is an open source project (see Section IX).

II. Research Statement

Our research goal is to create a web application which allows users to enhance the quality of audio recordings recorded using mobile phones. By providing users with tools for declipping, noise removal, equalization, and loudness normalization, we hope to improve the overall listening experience for users, especially in scenarios where audio recordings cannot be rerecorded, such as recordings of live music concerts.

III. Novelty

Current video uploading services do not provide an audio processing chain to content creators, allowing them to improve the audio quality of videos before uploading them to the internet. This project is an attempt to address this apparent whitespace in the industry. Additionally, the declipping algorithm is entirely novel and was published separately by the authors [3].

IV. Related Work

A. Audio Quality Enhancement in Uploading Services and Devices

Most video uploading services, including YouTube, normalize the loudness of the audio when a video is uploaded. Bandcamp requires users to upload lossless audio. Few, if any, uploading services expose any audio processing to users.

Real-time communication systems often have an audio processing chain that includes echo suppression and automatic gain control. However, this is done during recording time and is usually device specific.

Analogous image/video processing tools are widespread. Instagram allows users to apply filters to images before uploading them. YouTube has automatic stabilization tools, including the ability to detect unstable videos (YouTube goes as far as alerting users that their video is unstable and suggesting stabilizing their video). With image processing being so well integrated in the video uploading workflow, it is surprising that audio processing has not been given similar priority.

B. Declipping

Declipping audio is the process of removing the perceptual effect of clipping from clipped audio. Declipping is a two-step process: 1) The indices of the signal where clipping occurs must be detected (clipping detection), and 2) the signal values at these indices must be replaced with estimates that eliminate or reduce the perception of clipping without introducing other artifacts (sample replacement).

1) Clipping Detection

There is a surprisingly small amount of literature on clipping detection. One possible reason for this is that it is difficult to formally evaluate a clipping detection algorithm, as is discussed in the evaluation section.

There are two types of clipping: hard clipping and soft clipping. In hard clipping, any signal values above the clipping threshold are set to the clipping level. While easy to detect, this type of clipping is uncommon and usually occurs in the digital domain. Soft clipping occurs more often in practice. In soft clipping, signal values above the clipping threshold undergo nonlinearities that drive the signal value near the clipping threshold.

There are two main approaches to detecting clipping: histogram analysis [7] and time-domain analysis [9]. Histogram-based approaches rely on the fact that audio signals have predictable amplitude distributions: the distributions are high towards the mean and near-monotonically decrease towards both ends. Since hard and soft clipping both drive high amplitudes down near the clipping level, a clipped signal will have a probability distribution where amplitudes near the clipping level have an unnaturally high probability. This results in two 'bumps' in the histogram, one for positive clipping and one for negative clipping, near the positive and negative endpoints of the histogram. Histogram-based approaches work well in practice [7], although if the number of clipped samples is much smaller than the number of unclipped samples, it is likely that clipping will not be detected.

Time-domain approaches rely on the fact that the slope of the signal near the clipping level is relatively flat and/or that clipping results in sharp corners at the endpoints of each clipping interval. Since these approaches analyze the signal sample by sample, they tend to have fine time resolution. However, false positives can occur because there exist realistic signals that don't contain clipping but do exhibit the aforementioned characteristic properties of clipping (relatively sharp increase/decrease in slope and sections of flatness). Time-domain approaches work best when assumptions about the signal can be made such as the frequency content of the signal [9]. Such assumptions should not be made with music, which can contain any audible frequency.

2) Sample Replacement

Sample replacement is a large area of research that spans not only declipping but also many other applications including audio restoration and providing robustness against packet loss during real-time audio streaming [8]. Much research focuses on estimating a single 'burst' of unknown samples where there are many known samples on either side of the burst. Approaches include time-domain interpolation [2,14,22], frequency-domain interpolation [4,19,16], and sparse reconstruction [5,6].

The most common time-domain interpolation methods model the signal as an autoregressive process and use linear prediction to fill in the burst. Methods differ in how they guarantee that the prediction will be continuous on both sides of the burst. Janssen et. al. train a linear predictor on all known samples, and the estimates are formed by minimizing the estimation error over all samples in the burst [2]. Esquef et. al. train linear predictors on either side of the burst, and the results of forward and backward prediction are crossfaded over the burst [14]. Etter trains linear predictors on either side of the burst, but a single estimation is obtained by creating an objective function that takes both predictors into account [22]. These linear prediction results tend to work well with bursts that are shorter than 20ms [22]. However, the order of the autoregressive model must be known and is difficult to estimate. Training linear predictors of the incorrect order can result in unstable estimates.

Frequency-domain approaches create estimates for time-frequency bins instead of samples. Some approaches estimate separate autoregressive models for each sub-band of the signal [4,19]. Lagrange et al. handle signals containing vibrato by identifying partials on each side of the burst and training linear predictive models on the partials [16]. Some methods, including Lukin and Todd's approach, estimate tonal and nontonal components of the signal and treat them separately [19]. Frequency-domain methods are often used to estimate long bursts, where time-domain methods are insufficient. One common challenge with using frequency-domain approaches is that to get a large enough number of time-frequency bins to do proper interpolation requires a large number of known samples on either side of the burst. Locations of clipping are unpredictable, so these methods are difficult to apply to declipping.

Recent approaches to declipping use concepts from compressive sampling [5,6]. Clipped samples can be ignored and the signal can be interpreted as being sampled at non-uniform intervals. Then, the best sparse representation of a signal (in an accordingly sparse basis, such as the DCT) can be obtained using iterative optimization such as orthogonal matching pursuit [18]. These methods are relatively robust to different clipping scenarios, but unfortunately are computationally inefficient, usually having a complexity of $O(n^3)$, where n is the block size.

C. Noise Removal

Noise removal involves the suppression of noise in a signal. A signal is often modeled as the addition of a source signal and noise, where the noise is assumed to be stationary and uncorrelated to the source. Noise removal can be framed as the estimation of weights of an optimal linear filter given the spectral density of the noise. A noise removal algorithm includes noise estimation, the suppression rule (the method for estimating filter weights), and usually additional methods for reducing processing artifacts.

Noise estimation can be accomplished by first segmenting the signal into regions containing only noise vs. regions containing the source, and then averaging the spectra of the noise segments of the signal [20]. Depending on the application, segmentation can be done manually or automatically. For automatic noise segmentation, a classifier can be used. A simple classifier might classify by signal level thresholding [20].

These methods do not work for signals containing no noise-only segments (this happens when the source is never silent). If the source has a time-varying frequency, it can be assumed that at any instant, there exist frequency bins containing only noise. Then, taking the k-th quantile of the magnitude of a frequency bin across time will give an estimate of the noise magnitude at that frequency, where k is chosen depending on the amount of noise time-frequency bins compared to amount of source time-frequency bins [20].

The most common suppression rule minimizes the minimum mean squared error between the source signal and the output of the noise removal; this results in the Wiener filter [12]. It is known that processing a noisy input signal using a Wiener filter introduces artifacts such as musical noise. In general, there is a tradeoff between amount of noise suppression and amount of introduced artifacts. Ephraim and Malah propose a suppression rule that results in reduced artifacts [24]. Wolfe and Godsill introduce efficient alternatives to Ephraim and Malah's approach based on maximum a posteriori estimation of the original signal's magnitude and phase [11].

D. *Equalization*

Most non-expert users are more familiar with subjective timbre descriptors such as 'warm' or 'bright' than filter parameters such as cutoff frequencies and q values. Therefore, researchers have attempted to create interfaces for filters based on subjective terms [23,25]. Mecklenburg and Loviscach plotted subjective terms within a two dimensional space, and users modified the filter by selecting any point on the space [25]. Each subjective term has a corresponding frequency response based off of work by Katz [17]. When the user selects a point on the space, a desired frequency response is calculated by interpolating between the closest subjective term points. Cartwright and Prado also create a two dimensional perceptual space for users to interact with [23]. To construct this space, Cartwright and Prado first gathered guitar and drum recordings and created test audio files by filtering these recordings at various frequencies. Listeners then labeled pairs of files for similarity. A high dimensional space was derived from these similarity ratings using multidimensional scaling. Then, principal component analysis was used to reduce the dimensionality of the space to two dimensions.

V. System Description

First, the application workflow is presented from the perspective of the user. Second, the overall architecture of the system is described. Finally, the algorithms used in each audio processing unit are described.

A. *Workflow*

The user interface is shown in Figure 1. Upon loading the website, the user is prompted to upload audio by using a file chooser or dragging a file into the interface. The file is then normalized to -23 loudness units relative to full scale (LUFS) and displayed to the user in an interactive waveform viewer. The system displays two waveforms: the unprocessed input and the processed output. Before any processing occurs, the

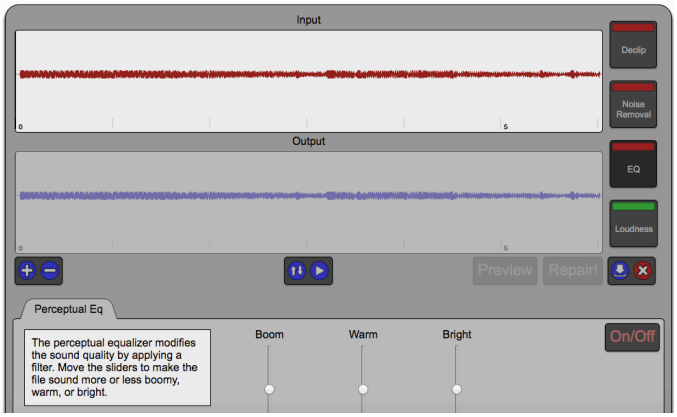

Figure 1: REPAIR user interface.

output is simply a copy of the input. Standard waveform interactions are supported: audio playback, pause and resume, playback of selected regions, zooming in and out, and toggling between the input and output in real time.

To the right of the waveforms are module buttons for declipping, noise removal, perceptual equalization, and loudness normalization. Each module button includes a status indicator: green indicates an active module that is included during processing, and red indicates an inactive module that is not included during processing. The user can click on the status indicator to toggle the status, or click the module button button to open a tab with settings for that module below the waveforms. To avoid loudness bias, loudness normalization cannot be deactivated.

Each settings tab includes a text description, a series of parameters, and an on/off button. The text description gives a brief overview of the module as well as an explanation for how to use parameters. The on/off button allows users to toggle the module between active and inactive states.

The declipping module contains no parameters.

The noise removal module prompts the user to select noise-only regions of audio from the input waveform. When the audio is loaded, sections of noise are automatically located by searching for segments of audio with a root mean square value below a threshold. This threshold is initially set to 10% of the peak amplitude in the waveform, but the user can modify the threshold by moving a slider. The user can edit regions by dragging and dropping the region endpoints, add regions by manually selecting a region and clicking a button, and delete regions by double clicking them.

The perceptual equalizer contains three sliders, each which provides independent control over a subjective timbral term: boom, warmth, and brightness. These sliders span values from -50 to 50. These values do not correspond to any meaningful unit but are useful for reference.

The loudness normalization module contains one slider for modifying the target LUFS. The slider spans 0 LUFS to -50 LUFS. If the target LUFS is high enough to cause clipping, the user is notified and the level is automatically set to the highest possible value that does not clip.

Below the waveform are two buttons: a preview button and a repair button. The preview button processes a 3.5 second portion of the audio and plays back the result to the user immediately after processing. This allows the user to listen to

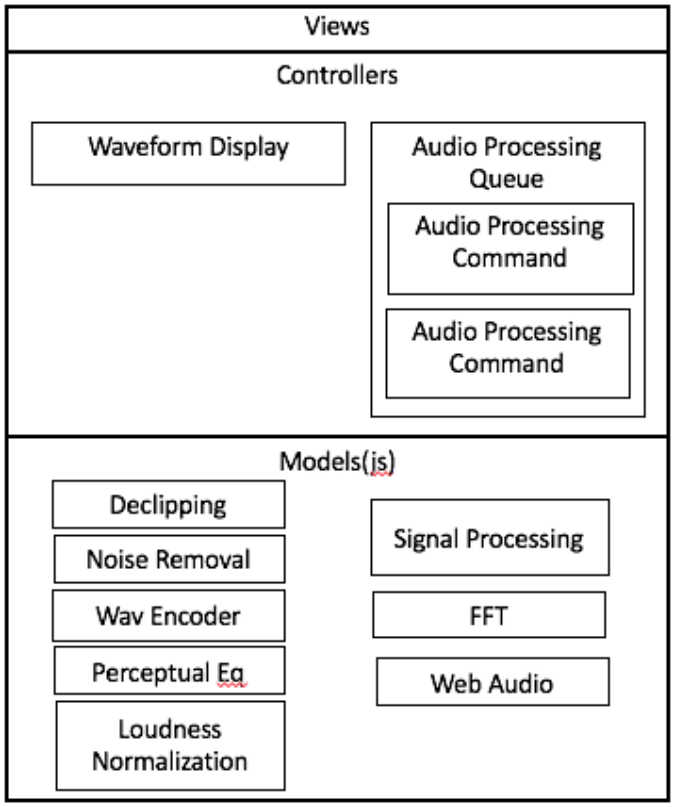

Figure 2: REPAIR system architecture.

the results without waiting for the entire audio to process. The section of audio with the largest root mean square value is chosen to preview. The repair button applies the processing to the entire audio, and upon completion updates the output waveform. For both the preview and the repair button, only active modules are used during audio processing.

After processing the audio, users can download the output as a .wav file by clicking on the download button.

B. System Architecture

Figure 2 shows a diagram of the system architecture. The system design follows a model view controller architecture. The views are implemented in HTML and CSS, and the models and controllers are implemented in JavaScript. The entire application runs in the browser.

The waveform interaction was implemented using WaveSurfer, a third party library for audio waveform display and interaction [26]. The façade design pattern (i.e., creating a simple API for managing a complex subsystem) was used to facilitate easy synchronization between the input and output waveforms. A façade class was created containing two WaveSurfer objects, one for the input and one for the output, and the façade class functions such as play() and pause() internally called corresponding WaveSurfer functions.

Along with the modules exposed to the user, a wav encoder was also implemented (i.e., the audio is wrapped with a wav header) to allow users to download the output. Additionally, shared signal processing code was placed into its own model. The perceptual equalizer was implemented using biquad filters from the Web Audio API [27], and the Web Evaluation Tool [28] was used to calculate LUFS loudness (following the loudness model from the EBU R 128 specification [10]). For frequency domain processing, the Fast Fourier Transform library provided by Project Nayuki was used [13].

Audio processing "command functions" were implemented to interface between the audio processing models and the controller. Following the command design pattern, each of these command functions can be black-boxed and treated in the same manner. Each command function processes audio on a separate thread (using web workers or Web Audio, depending on the module) and copies the result into the output audio buffer. When the user clicks the preview or repair button, the controller populates a queue of function pointers containing these command functions ordered as such: declipping, noise removal, equalization, and loudness normalization. To process audio, a command function is dequeued and run. After running each command function, if the queue is empty, processing is finished; otherwise, the next command function is dequeued and run. Declipping occurs first because clipped regions should be fixed before other processing. Equalization occurs after noise removal because filtering would effect the estimation of the noise spectrum. Loudness normalization occurs last because any processing will effect the calculation of loudness. Inactive modules were not placed on the stack.

C. Algorithm Description

1) Equalization

The boom, warmth, and brightness sliders each control the gain of separate Web Audio biquad filters. The boom slider controls a low-shelf filter with a cutoff frequency of 60 Hz. The warmth slider controls a peaking filter with a middle frequency of 300 Hz. The brightness slider controls a high-shelf filter with a cutoff frequency of 9000 Hz. These frequency values were chosen based on an online audio mixing tutorial by iZotope [15].

2) Noise Removal

The noise spectrum is estimated by taking the average spectrum of the noise segments of the signal. Wolfe and Godsill's maximum a posteriori suppression rule is used to calculate the filter weights [11]. The filter is applied in the frequency domain.

Measures are taken to reduce artifacts. First, the filter weights are smoothed across frequency by using a median filter. This prevents notching effects that could occur if one weight is much larger or smaller than the surrounding weights. Second, the filter weights are smoothed over time using a low pass filter. This reduces musical noise, which is caused when high frequency weights change rapidly. Third, the a priori SNR estimate is smoothed over time using a low pass filter. The a priori SNR estimate is an internal variable used in the suppression rule calculations that should not vary rapidly.

3) Declipping: Clipping Detection

The clipping detection algorithm is divided into two parts: clipping level detection and clipping interval detection. In clipping level detection, positive and negative clipping thresholds are found by looking for 'bumps' in the histogram of a predefined width. In clipping interval detection, sample-accurate locations of clipping are found by analyzing the

signal in the time-domain. Sample replacement uses a frequency-domain approach that focuses on fast processing, weak requirements on the number of samples on each side of the burst, and robustness to different types of signals.

a) Clipping Level Detection

First, the amplitude histogram is computed with 6000 equally spaced bins that span the amplitude range of the signal. To remove small, noisy fluctuations in the histogram, the histogram is smoothed by low-pass filtering it with a forward-backward exponential smoothing filter. Then, an adaptive threshold is computed by filtering the smoothed histogram with a forward-backward exponential filter with a significantly lower cut-off frequency. Note that the first pass of filtering aims to smooth the histogram, while the second pass aims to create a very slow-changing threshold. In order to locate the bumps in the histogram, a novelty function is computed by subtracting the threshold from the smoothed histogram. The novelty function will be above zero at the locations of the bumps. The novelty function might also be slightly above zero at other locations depending on the input signal. Then, all intervals of consecutive positive values occurring within the outermost 10% of the novelty function are found. These intervals are candidates which might correspond to the bumps. If the candidate does correspond to a bump, then it will have a large area compared to the other candidates. Therefore, a bump is detected when a candidate has an area that is 3 standard deviations above average. The clipping level is determined to be the amplitude corresponding to the innermost bin in the bump. If neither a positive nor a negative clipping level was found, then the algorithm reports that the signal contains no clipping. Thus, this method naturally handles cases where no clipping occurs. Note that histogram normalization is not required because the method only relies on relative values.

b) Clipping Interval Detection

Given the clipping levels, the clipping intervals are found by analyzing the signal in the time domain. Each local maximum above the clipping level is assigned a clipping interval. The clipping intervals are then extended according to criteria that ensure that the slope during clipping is near zero. The threshold for the the slope is determined by looking at the bump width, which corresponds to the amount of amplitude variation during clipping.

4) Declipping: Sample Replacement

An overview of the entire replacement process is shown in Figure 3. To remove clipping, the clipping intervals are first divided into short intervals and long intervals. The short intervals are interpolated in the time-domain using cubic spline interpolation. The signal is then split into a low-frequency band and a high-frequency band. Samples in the high-frequency band are replaced by linearly interpolating time-frequency bin magnitudes. The low-frequency band remains untouched. Finally, the low-frequency band and the processed high-frequency band are combined.

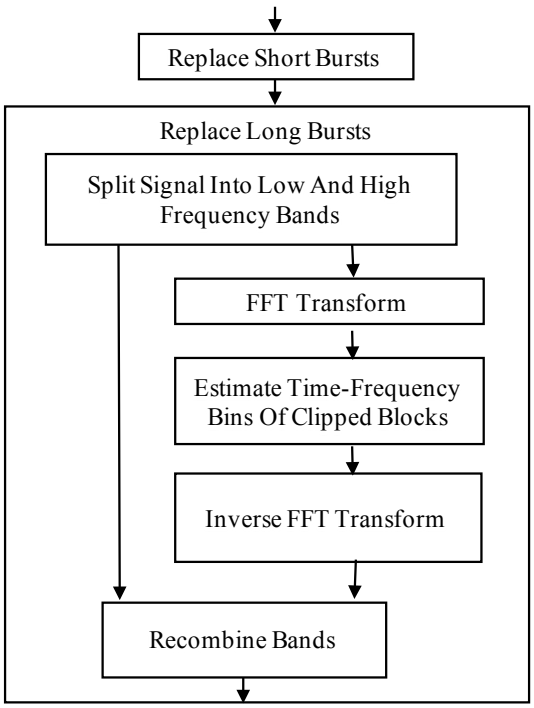

Figure 3: Sample replacement block diagram.

a) Replacing Short Bursts

Short intervals are replaced using cubic spline interpolation. The cubic spline interpolator is given at most 20 samples on either side of the clipping interval. If either side of the clipping interval contains less than 3 samples, no interpolation occurs.

b) Replacing Long Bursts

The distortion introduced by clipping mostly resides in high frequencies. Thus, the low frequencies are not processed. The signal is split into a low frequency band (0-100 Hz) and a high frequency band (100 Hz-Nyquist), and only the high frequency band is processed. After processing, the bands are combined by adding the processed high-frequency band to the low frequency band. It is important to note that filtering the signal modifies the locations where clipping occurs: the clipping intervals get smeared out and could be extended by up to the length of the filter's impulse response. With this in mind, an FIR filter with an impulse response length of 25 is used. The filter's cutoff frequency is 100 Hz, and the filter is designed to be as steep as possible given its order. The filter is applied forwards and backwards to ensure that the filter is zero phase. Because the filter is zero phase, the midpoints of the clipping intervals do not change. Since the support of the filter's impulse response spans from sample -25 to 25, the clipping intervals are extended by 25 samples on either side after filtering.

To replace long intervals in the high-frequency band, the signal is first blocked with a block size N = 512 and 75% overlap. The goal is to estimate the magnitudes and phases for each block containing any clipped samples. After obtaining the estimates, the time-domain signal can be reconstructed by inverse transforming all blocks and overlap-adding them together.

Clipping Detection Results					
Algorithm	Clipping Level	Histogram Block Length	Precision	Recall	F-measure
Histogram	95th Percentile	File	0. 937	0. 889	0. 911
Histogram	95th Percentile	3 Seconds	0.937	0.892	0.914
Histogram	90th Percentile	File	0.942	0.840	0.881
Histogram	90th Percentile	3 Seconds	0.947	0.880	0.912
Combined	95th Percentile	File	0.941	0.910	0.925
Combined	95th Percentile	3 Seconds	0.940	0.908	0.922
Combined	90th Percentile	File	0.950	0.902	0.925
Combined	90th Percentile	3 Seconds	0.945	0.862	0.894

Table 1: Clipping detection results.

Because clipping has a negligible perceptual effect on the phase content of the signal, the phase of the current clipped block is used as the phase estimate.

The magnitude of a clipped block is estimated by linearly interpolating between the magnitudes of the two closest reliable blocks. Reliable blocks are located by searching for the nearest interval of at least N / 4 consecutive non-clipped samples.

There are occasions when linear interpolation of magnitudes causes artifacts. Most notably, overestimation of magnitudes during blocks containing clipped transients can cause the transients to sound tonal, resulting in 'blips' during high hat and snare hits. To reduce this artifact, each estimated magnitude is upper bounded by the magnitude of the corresponding bin in the clipped block.

VI. Evaluation

We focus our evaluation on the declipping algorithm, since this is the most novel aspect of the project. We also informally evaluate the end to end system.

A. Clipping Detection

1) Methodology

Evaluating the detection of soft clipping is problematic because it is difficult to record soft clipping with corresponding sample-accurate ground truths of clipping indices. Existing research either simulates soft clipping in software [7] or does not include a formal evaluation of their algorithm [9]. To properly evaluate with soft clipped signals, a method would be required that is able to record clean audio data and corresponding soft clipped audio time-aligned and phase-aligned to near-sample accuracy in order to reliably annotate clipping interval locations.

For our evaluation, soft clipping is simulated by digitally hard clipping a signal (and saving the clipping locations), and then encoding the clipped signal using a lossy codec. When a hard clipped signal goes through a lossy encoder, the clipped sections of the signal are modified, as these segments of audio are the most difficult to encode (the frequency-domain is least sparse at sharp edges).

To evaluate, clipping detection is run on the resulting audio files and the precision, recall, and f-measure of the clipping indices are computed. The dataset published by Homburg et al. is used for evaluation [1]. This dataset contains 1886 10-second excerpts of songs from 9 musical genres. The audio files are encoded as MPEG-1 layer 3 files at 44.1 kHz/128 kb. This dataset is chosen because it covers a wide range of musical styles.

The evaluation is run with different amounts of clipping, different block sizes, and with different versions of the algorithm. The clipping levels should be chosen such that the amount of clipping is comparable between audio files. Clipping based on a percentage of the maximum amplitude does not normalize for amount of clipping because different signals have different amounts of dynamic variation. Instead, the clipping level is chosen based on a percentile of the amplitude distribution in the signal. This guarantees that the same number of samples clip in each audio file. Because the amount of perceptual clipping also depends on the frequency content of the signal, choosing the clipping level based on amplitude percentile does still not guarantee equal amounts of perceptual clipping. Preprocessing such as filtering the input by the inverse spectral envelope might be worth investigating, but was considered excessive for this evaluation.

Running the evaluation with different block sizes allows us to verify that the system can be used when the clipping level is time-variant. Comparisons are also conducted between clipping detection using the clipping interval detection (combined) vs. only using clipping level detection (histogram). When only using clipping level detection, clipping is detected at each sample with an amplitude above the clipping level.

2) Results

Results from the clipping detection evaluation are shown in Table 1. The algorithm performs with an f-measure near 0.91 in most cases.

In all experiments, precision (~0.94) is higher than recall (~0.90). One explanation for this is that the clipping level is chosen to be the innermost bin in the bump. It is possible that the amplitudes placed in bins on the inner half of the bump sometimes correspond to clipped regions of the signal and other times correspond to non-clipped regions of the signal. If this is the case, then there is an inherent tradeoff between

precision and recall for these regions of the signal. Here, precision is considered more important than recall because the goal is to replace all clipped regions of the signal, which can be done without knowledge of all non-clipped regions of the signal.

The following observations can be made by comparing results of different experiments. Firstly, experiments with different amounts of clipping have similar precision, recall, and f-measure. This indicates that the clipping level estimate is equally reliable regardless of the size of the bumps in the histogram. Secondly, precision, recall, and f-measure are similar between experiments using block-level histograms and experiments using file-level histograms. This indicates that the clipping detection algorithm can be used on time varying clipping. Thirdly, recall tends to improve when using clipping interval detection. This reinforces our claim that time-domain analysis can help find the accurate boundaries of clipping, which is especially useful for reducing false positives.

It is worth mentioning that lossy encoding might slightly alter the locations of clipping, which could impact the results of our evaluation.

B. Sample Replacement

1) Methodology

Objectively evaluating sample replacement is also problematic because the goal is to measure the perceptual difference between the clean signal and the reconstructed signal. Here, the evaluation is based off of the standard audio distortion measurement metric tonal harmonic distortion (THD), and some declipping listening examples are provided online [21].

The total harmonic distortion (THD) of a clipped sinusoidal before and after sample replacement are compared. The sinusoidal input is soft clipped (using the same method as in the clipping detection evaluation) at the 90th percentile and samples are replaced using the ground truth clipping locations. The signal is 2 seconds long and sampled at 44.1 kHz. The performance of replacing short intervals (cubic spline interpolation) and replacing long intervals (frequency magnitude interpolation) are tested separately. When testing long interval replacement, the input is also padded with one second of a non-clipped sinusoidal before and after the clipped region. Without this padding, there would be nowhere in the signal where a clean FFT is possible (note that a clipped sinusoidal would normally be declipped in the time domain, but here the goal is to evaluate only the frequency domain interpolation). The padding is removed before measuring the THD.

2) Results

THD results are shown in Figure 4. Both time domain and frequency domain methods are shown to improve the signal THD. The time domain reconstruction THD increases as frequency increases. This is because the number of samples in a single period decreases as frequency increases, and therefore a greater interpolation accuracy is necessary for higher frequencies to maintain a constant THD. This is an artifact of sampling; if the experiment is run with a high enough sample rate (e.g., 192 kHz), then the time domain reconstruction THD is constant with respect to frequency, remaining near -70 dB.

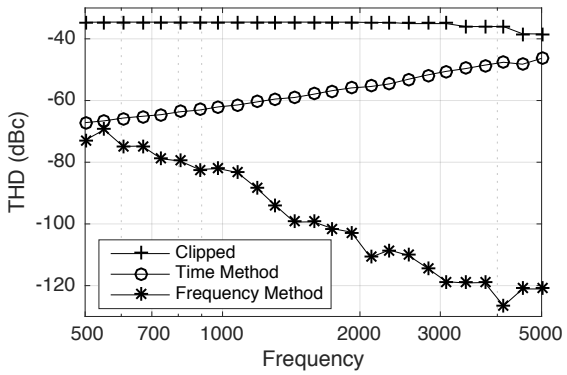

Figure 4: Total harmonic distortion of reconstruction on sinusoidal input.

The frequency domain reconstruction THD decreases as frequency increases. This is due to phase differences between the clipped and the original sinusoidal. In order to verify this, the original phase of the sinusoidal was used, and the resulting THD was relatively constant with respect to frequency (approximately -90 dB).

The frequency domain approach obtains a lower THD than the time domain approach across frequencies. However, the frequency domain approach required extra information (a clean signal before and after clipping) in order to function properly. This validates the two-stage approach to sample replacement.

Results for sinusoidals are not necessarily generalizable to high bandwidth signals. The next section briefly illustrates the algorithm performance on real world signals.

C. Declipping on Real World Examples

To illustrate the performance of our algorithm on a real world signal, the waveform and spectrogram of a clean, clipped, and declipped speech signal are visualized. Figure 5 shows the signal waveforms. The waveform of the declipped signal matches the general shape of the clean signal, although the reconstruction peaks tend to have a lower magnitude than the clean peaks. Figure 6 shows the signal spectrograms. From the spectrogram of the clipped signal, the high frequency distortion caused by clipping is clearly visible. It can be seen from the reconstruction spectrogram that this distortion is mostly removed; however, the higher partials of speech are occasionally missing.

D. System Computation Speed

To evaluate the computational efficiency of the audio processing, we ran an experiment using a 2012 MacBook Pro with a 2.9 GHz quad-core processor and Google Chrome. A 16 bit, 30 second stereo audio file sampled at 44.1 kHz was loaded into the web application. The audio file contained clipping. All modules were activated. The preview processing took just under 4 seconds. Processing the entire audio took just over 32 seconds: declipping took 16 seconds, noise removal took 16 seconds, and the perceptual equalizer and loudness normalization both ran in less than one second.

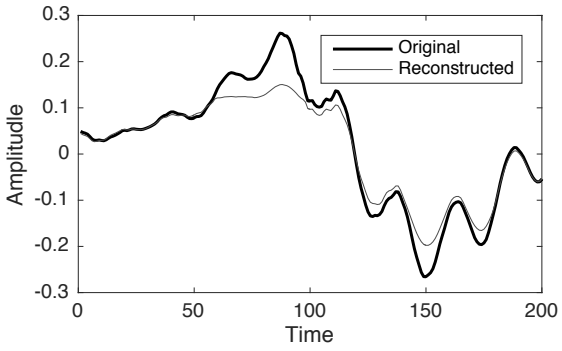

Figure 5: Clean and reconstructed speech signal waveforms.

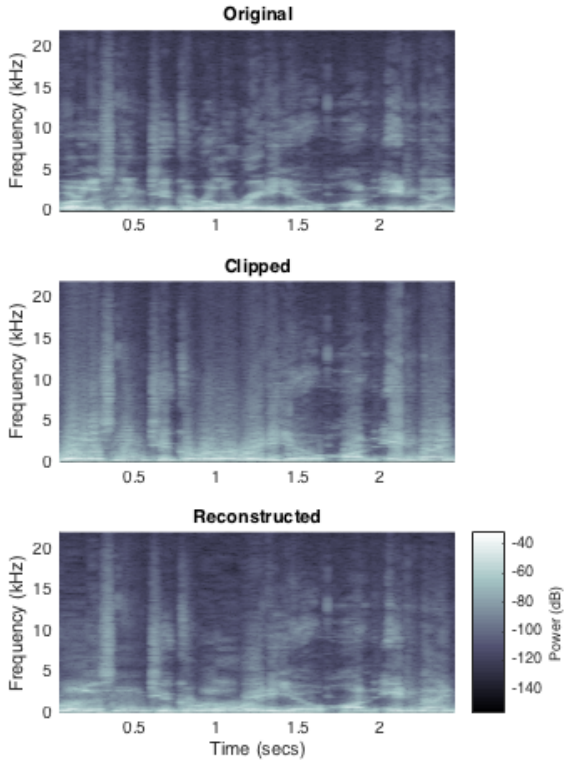

Figure 6: Clean, clipped, and reconstructed speech signal spectrograms.

VII. Discussion

It is important for audio processing to happen quickly so that users can receive timely audio feedback about their parameter adjustments. From the perspective of a user, audio processing in REPAIR might take too long for a satisfactory experience; however, the preview button provides some audio feedback within an acceptable amount of time (under 4 seconds).

Processing time could be reduced by doing audio processing over a server. Audio processing code could then be written in an efficient low level programming language such as C++. Adding a server side to the application could also greatly reduce the amount of memory used on the client. However, server-side processing raises privacy concerns for users (e.g. potential storage of user audio on the server) and adds complexity to the overall system.

VIII. Conclusion

REPAIR can enhance the quality of mobile phone recordings, which make up a large percentage of audio and video content on the internet. It is our hope that this project will inspire video uploading services to include an audio processing toolchain in their workflows.

IX. Deliverables

Web application: https://cplaguna-audio.github.io/REPAIR/

Github repository: https://github.com/cplaguna-audio/REPAIR

References

[1] H. Homburg, I. Mierswa, B. Möller, K. Morik, and M. Wurst, "A Benchmark Dataset for Audio Classification and Clustering.," in *ResearchGate*, 2005, pp. 528–531.

[2] A. Janssen, R. Veldhuis, and L. Vries, "Adaptive interpolation of discrete-time signals that can be modeled as autoregressive processes," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, no. 2, pp. 317–330, Apr. 1986.

[3] C. Laguna and A. Lerch, "An Efficient Algorithm for Clipping Detection and Declipping Audio," presented at the Audio Engineering Society Convention 141, 2016.

[4] L. W. P. Biscainho, P. S. R. Diniz, and P. A. A. Esquef, "ARMA processes in sub-bands with application to audio restoration," in *The 2001 IEEE International Symposium on Circuits and Systems, 2001. ISCAS 2001*, 2001, vol. 2, pp. 157–160 vol. 2.

[5] A. Adler, V. Emiya, M. G. Jafari, M. Elad, R. Gribonval, and M. D. Plumbley, "Audio Inpainting," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 3, pp. 922–932, Mar. 2012.

[6] B. Defraene, N. Mansour, S. De Hertogh, T. van Waterschoot, M. Diehl, and M. Moonen, "Declipping of Audio Signals Using Perceptual Compressed Sensing," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 21, no. 12, pp. 2627–2637, Dec. 2013.

[7] S. Aleinik and Y. Matveev, "Detection of Clipped Fragments in Speech Signals," Int. J. Electr. Electron. Sci. Eng, pp. 74–80, 2014.

[8] S. Godsill, P. Rayner, and O. Cappé, "Digital Audio Restoration," in *Applications of Digital Signal Processing to Audio and Acoustics*, M. Kahrs and K. Brandenburg, Eds. Springer US, 2002, pp. 133–194.

[9] T. E. Riemer, M. S. Weiss, and M. W. Losh, "Discrete clipping detection by use of a signal matched exponentially weighted differentiator," in *Southeastcon '90. Proceedings., IEEE*, 1990, pp. 245–248.

[10] "EBU Technical Recommendation R128 -- Loudness Normalisation and Permitted Maximum Level of Audio Signals." European Broadcasting Union, 2010.

[11] P. J. Wolfe and S. J. Godsill, "Efficient Alternatives to the Ephraim and Malah Suppression Rule for Audio Signal Enhancement," *EURASIP J. Adv. Signal Process.*, vol. 2003, no. 10, p. 910167, Dec. 2003.

[12] N. Wiener, Extrapolation, Interpolation, and Smoothing of Stationary Time Series, vol. 2. Cambridge: MIT Press, 1949.

[13] Nayuki, "Free small FFT in multiple languages," 2016. [Online]. Available: https://www.nayuki.io/page/free-small-fft-in-multiple-languages. [Accessed: 11-Dec-2016].

[14] P. A. Esquef, V. Välimäki, K. Roth, and I. Kauppinen, "Interpolation of long gaps in audio signals using the warped burg's method," in *Proc. 6th Int. Conf. on Digital Audio Effects (DAFx-03)*, 2003, pp. 8–11.

[15] "iZotope Pro Audio Essentials." [Online]. Available: https://pae.izotope.com.

[16] M. Lagrange, S. Marchand, and J.-B. Rault, "Long Interpolation of Audio Signals Using Linear Prediction in Sinusoidal Modeling," *Journal of the Audio Engineering Society*, vol. 53, pp. 891–905, Oct. 2005.

[17] B. Katz, *Mastering Audio: The Art and the Science*, 2 edition. New York: Focal Press, 2007.

[18] Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad, "Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition," in *1993 Conference Record of The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers, 1993*, 1993, pp. 40–44 vol.1.

[19] A. Lukin and J. Todd, “Parametric Interpolation of Gaps in Audio Signals,” presented at the Audio Engineering Society Convention 125, 2008.

[20] V. Stahl, A. Fischer, and R. Bippus, “Quantile based noise estimation for spectral subtraction and Wiener filtering,” in *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, 2000, vol. 3, pp. 1875–1878 vol.3.

[21] C. Laguna, “REPAIR,” 2016. [Online]. Available: https://cplaguna-audio.github.io/REPAIR/. [Accessed: 11-Dec-2016].

[22] W. Etter, “Restoration of a discrete-time signal segment by interpolation based on the left-sided and right-sided autoregressive parameters,” *Signal Processing, IEEE Transactions on*, vol. 44, no. 5, pp. 1124–1135, 1996.

[23] M. B. Cartwright and B. Pardo, “Social-EQ: Crowdsourcing an Equalization Descriptor Map,” in *International Society for Music Information Retrieval*, 2013, pp. 395–400.

[24] Y. Ephraim and D. Malah, “Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 6, pp. 1109–1121, Dec. 1984.

[25] S. Mecklenburg and J. Loviscach, “subjEQt: Controlling an Equalizer Through Subjective Terms,” in *CHI ’06 Extended Abstracts on Human Factors in Computing Systems*, New York, NY, USA, 2006, pp. 1109–1114.

[26] katspaugh, “wavesurfer.js.” [Online]. Available: https://wavesurfer-js.org/. [Accessed: 11-Dec-2016].

[27] C. Rogers, “Web Audio API.” [Online]. Available: https://www.w3.org/TR/webaudio/. [Accessed: 11-Dec-2016].

[28] N. Jillings, D. Moffat, B. De Man, J. D. Reiss, and R. Stables, “Web Audio Evaluation Tool: A framework for subjective assessment of audio.” Georgia Institute of Technology, 2016.