Reduction of Static Noise in Real Time

ETIN80 *Algorithms in Digital Signal Processors*

Alexander Skafte tfy13ask@student.lu.se
Mattias Josefsson tpi13mjo@student.lu.se

Supervised by: Mikael Swartling

February, 2017

Summary

In this document we examine a simple algorithm for reduction of static noise in real time. First a prototype is developed in MATLAB, after which the algorithm is implemented on a SHARC digital signal processor using the Visual DSP++ framework.

Contents

1 Introduction

A problem with capturing audio input is static noise generated by either internal hardware or by external factors such as nearby fans, air conditioners or other constant auditory disturbances. In this paper we investigate a method and an algorithm for reducing such noise while retaining the relevant parts of the audio. To test the algorithm we implement it on a widely used digital signal processor.

2 Theory

By means of block processing we obtain, for each block k, an input vector $\mathbf{x}(k)$ of 32 audio samples, with the amplitude of $\mathbf{x}(k)$ being normalized to have a value between 0 and 1. $\mathbf{x}(k)$ is then passed through a set of narrow-band analysis filters which give us 64 frequency sub-bands. We express this quantity, for each sub-band n, as $\mathbf{X}_n(k)$.

For purposes of identifying changes in amplitude and thus separating relevant input from noise, we keep a temporal recursive average of the power of each sub-band of $\mathbf{X}_n(k)$, which is, for each sub-band n, done by the update

$$\mathbf{P}_n(k) = \alpha\mathbf{P}_n(k-1) + (1-\alpha)|\mathbf{X}_n(k)|^2$$

where $\mathbf{P}_n(k)$ is the power of $\mathbf{X}_n(k)$ at time instant k, and α denotes a forgetting factor that is usually in the range $[0.90, 0.99]$ (we use $\alpha = 0.93$). $\mathbf{P}$ is preferably initialized to have the same power as the noise, but it converges quickly either way. The value of α should be fast (i.e. low) enough to "notice" and follow changes in amplitude, e.g. when someone is speaking.

$\mathbf{P}$ is thereafter transformed into the dB domain, and from that value a constant noise level is subtracted which in our case is approximately $-67\,\mathrm{dB}$, so we have

$$\mathbf{q}_n(k) = 10\log_{10}[\mathbf{P}_n(k)] - (-67\,\mathrm{dB})$$

Ideally, the noise level would be estimated by some mechanism and not approximated by a constant.

We then in the dB domain create a damping value D by means of interpolation between a lower and upper threshold L and U using a Sigmoid-curve. In our case, we use $L = 5\,\mathrm{dB}$ and $U = 20\,\mathrm{dB}$. For $\mathbf{q}_n(k) > U$ we do not suppress any noise (since we have actual input, not just noise) and for $\mathbf{q}_n(k) < L$ we suppress the noise by some level, in our case $D = -15\,\mathrm{dB}$.

The interpolation function has the following form, where s is a ”squeeze” factor that controls how steep the interpolation is:

$$\text{Interpolate}(\mathbf{q}_n(k)) = \begin{cases} -D, & \infty < \mathbf{q}_n(k) \leq L \\ D + \frac{0-D}{1+\exp\left(2s\left(-\mathbf{q}_n(k)+\frac{U+L}{2}\right)\right)}, & L < \mathbf{q}_n(k) < U \\ 0, & U \leq \mathbf{q}_n(k) < \infty \end{cases}$$

By interpolating $\mathbf{q}_n(k)$ in such a way and thereafter converting the result back from the dB domain we obtain a gain factor with which we can scale each $\mathbf{X}_n(k)$ to obtain an output frequency domain signal $\mathbf{Y}_n(k)$.

$$\begin{aligned} \mathbf{g}_n(k) &= \text{Interpolate}(\mathbf{q}_n(k)) \\ \mathbf{G}_n(k) &= 10^{\mathbf{g}_n(k)/20} \\ \mathbf{Y}_n(k) &= \mathbf{G}_n(k)\mathbf{X}_n(k) \end{aligned}$$

$\mathbf{Y}_n(k)$ is thereafter synthesized into a 32 sample output audio block $\mathbf{y}(k)$ by passing it through an inverse of the filter bank mentioned at the beginning of the description of this algorithm.

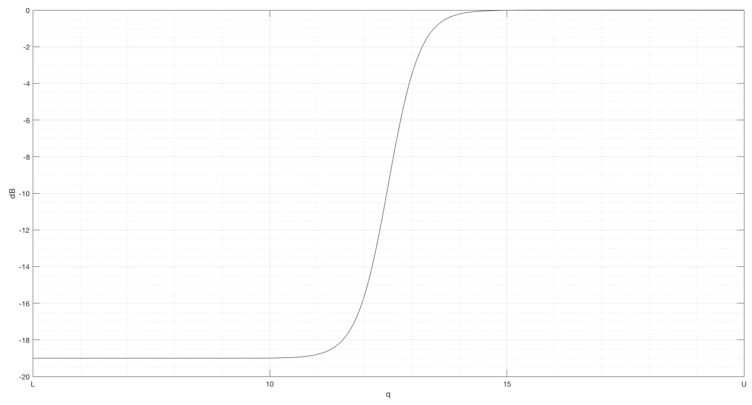

Figure 1: The interpolation function.

In conclusion, one may express the algorithm as the following pseudo-code:

```
function Process[$\mathbf{x}(k)$] $\rightarrow \mathbf{y}(k)$
    Input vector $\mathbf{x}(k)$, $0 \leq |\mathbf{x}(k)| \leq 1$
    For each block $k$, do
        $\mathbf{X}(k) = \text{Analyze}[\mathbf{x}(k)]$
        For each frequency sub-band $n$ in $\mathbf{X}(k)$, do
            $\mathbf{P}_n(k) = \alpha \mathbf{P}_n(k-1) + (1-\alpha)|\mathbf{X}_n(k)|^2$
            $\mathbf{q}_n(k) = 10 \log_{10}[\mathbf{P}_n(k)] - (-67\,\text{dB})$
            $\mathbf{g}_n(k) = \text{Interpolate}[\mathbf{q}_n(k)]$
            $\mathbf{G}_n(k) = 10^{\mathbf{g}_n(k)/20}$
            $\mathbf{Y}_n(k) = \mathbf{G}_n(k)\mathbf{X}_n(k)$
        $\mathbf{y}(k) = \text{Synthesize}[\mathbf{Y}(k)]$
```

3 Implementation

We chose to first implement the algorithm in Matlab due to its vast math library and development speed. After having created a working prototype acting on a recorded signal with added artificial static noise added we moved on to the Visual DSP++ environment, where we would rewrite the algorithm in C.

3.1 Matlab progress

In our initial Matlab implementation we used an ordinary discrete Fourier transform (with Matlab's `fft` command) to separate our signal into frequency sub-bands. By doing so we encountered the issue of not being able to remove input noise without removing parts of the speech as well, resulting in a fragmented output. By instead using a multi-rate polyphase filter bank—a set of narrow-band filters, the details of which can be found in [1]—our algorithm managed to remove the noise without significantly affecting the speech part of the signal.

Initially we also used *two* recursive averages of the power of $\mathbf{X}$; a ”slow” one meant to follow the noise and a ”fast” one meant to react to fast and large changes in amplitude (i.e. relevant input such as speech). At this stage we encountered the issue of the slow average taking at least 20 seconds to converge to the correct noise level, and without long enough pauses between relevant input (e.g. speech) the slow average would start to increase undesirably, going over the noise mean making the algorithm remove a large part of the speech as well. To counter this we modified the algorithm to instead have only a fast moving average and a constant estimated value for the noise level.

3.2 Visual DSP++ progress

For the DSP we had to change programming language to C instead of MATLAB. This meant converting the algorithm to C and implementing functions as we is included in the standard MATLAB library but not in C, such as complex conjugate and element wise multiplication. Aside from that we now had a continuous discrete signal to work with in real time instead of a prerecorded finite signal. This made us change our parameters around to suit the static noise from the DSP, and use interrupts on the DSP to handle processing the signal while constantly getting new samples to handle.

4 Results

For purposes of displaying the results we recorded two audio samples; one without noise reduction and one with a noise reduction of $-19\,\text{dB}$. They can be found by following the links below:

Before: alexanderskafte.com/static/dsp_noise_0.wav

After: alexanderskafte.com/static/dsp_noise_19.wav

5 Conclusion

By implementing the algorithm explained in this report on the DSP we were able to remove most of the static noise without significantly reducing the quality of the relevant input signal. We however hard coded our parameters to fit a specific environment of the DSP. Should the noise occur under different circumstances the parameters would have to be different, which is obviously suboptimal. As such, an improvement would be to have the algorithm automatically estimate the properties of the noise and then choose parameters based on the estimate.

The usage of the α parameter could also be modified further so that the rise time and fall time of $\mathbf{P}$ would be different. Before updating $\mathbf{P}$ one can also have an update of α of the form

$$\alpha = \begin{cases} 0, & |\mathbf{X}_n(k)|^2 > \mathbf{P}_n(k-1) \\ \alpha', & \text{otherwise} \end{cases}$$

meaning that when the amplitude is rising with time (first case) the reaction is ”instant”, and when the amplitude is constant or falling with time the usual recursive average is applied for some $\alpha' \in [0.90, 0.99]$.

During this project we have mainly focused on the processing part of the noise reduction and how to reduce noise in a pre-filtered signal. Since the filter bank was provided to us by our adviser, an extension of the project would be to implement one ourselves for use in the algorithm.

Another way to extend this project would be to look at more advanced models to reduce static noise in a signal such as using a hidden Markov model.

References

[1] *Direction of Arrival Estimation and Localization of Multiple Speech Sources in Enclosed Environments*, Mikael Swartling. Karlskrona: Blekinge Institute of Technology, 2012.