Online Adaptive Filtering: The Bork-Space XFCODE

M. Evans

March 24, 2008

1 Introduction

Conceptually, an adaptive filter algorithm is simply a means of measuring the transfer function from a noise source to a signal, and using that transfer function to remove the noise from the signal. The algorithm is adaptive because it uses feedback to adjust the noise-to-signal transfer function. Online adaptive filtering is based on the least mean squares (LMS) algorithm, which can be shown to produce a Wiener filter under certain conditions. The variant most closely related to ours appears to be the “Filtered-X LMS” algorithm (FXLMS), which accounts for system transfer functions. The theory behind the LMS algorithm, and FXLMS variant, can be found on-line and will not be discussed here.

2 Algorithm

The LMS algorithm is often drawn in what I will call an “off-line” configuration (see figure 1). The error signal is simply the difference between the target “Noisy Signal” and the correction signal output by the FIR filter. The entire algorithm can be written as

$$S_{err} = S_{targ} - S_{corr}, \tag{1}$$

$$S_{corr} = \vec{v}_{FIR}^{T} \vec{v}_{wit}, \tag{2}$$

with the adaptive part changing the FIR coefficients $\vec{v}_{FIR}$ by

$$\Delta \vec{v}_{FIR} = \mu S_{err} \vec{v}_{wit} \tag{3}$$

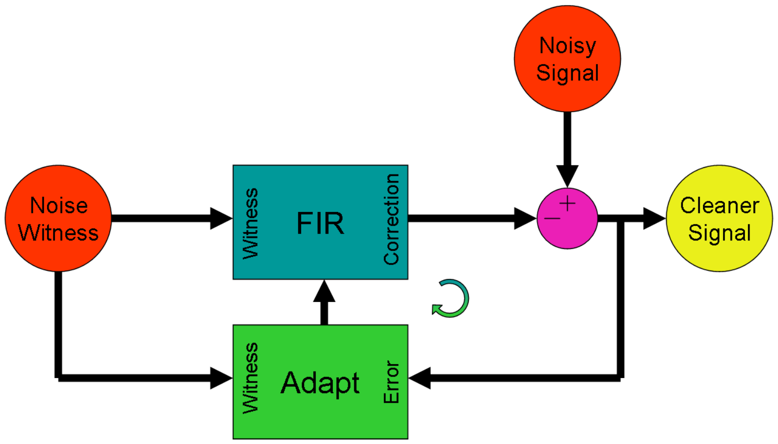

Figure 1: Functional parts of the LMS algorithm.

at each cycle. μ is the canonical name for the adaptation rate.

One might imagine that, in the context of LIGO, we would like to minimize some control signal by using a witness sensor to predict its value and cancel its effect at or near the source. For example, one could use a seismometer to move HEPI so as to cancel seismic noise that would otherwise be seen in DARM_CTRL. [1] If we ignore (or compensate) the actuator transfer functions, and assume that our correction signal is sent into a loop with high bandwidth, we can think of this as a simple rearrangement of the LMS algorithm in which the LMS error signal is replaced by the control signal produced by the high-bandwith loop (see figure 2).

Unfortunately, the stability of the LMS algorithm is dependent on the phase of the transfer function from the correction signal to the error signal $H_{corr \rightarrow err}$. In the original form $H_{corr \rightarrow err} = -1$, but as we move toward a more realistic implementation this simple relationship is lost. While one can imagine compensating actuator transfer functions so as to maintain $H_{corr \rightarrow err} = -1$, there are some inevitable features of real systems which cannot be compensated in on-line operation (e.g., delays). The FXLMS variant is designed to deal with the stability problems brought on by a system complicated by non-unity transfer functions (see figure 3).

As an incomplete summary of the theory behind FXLMS, to have stable

[1] This is, of course, already done, but not adaptively.

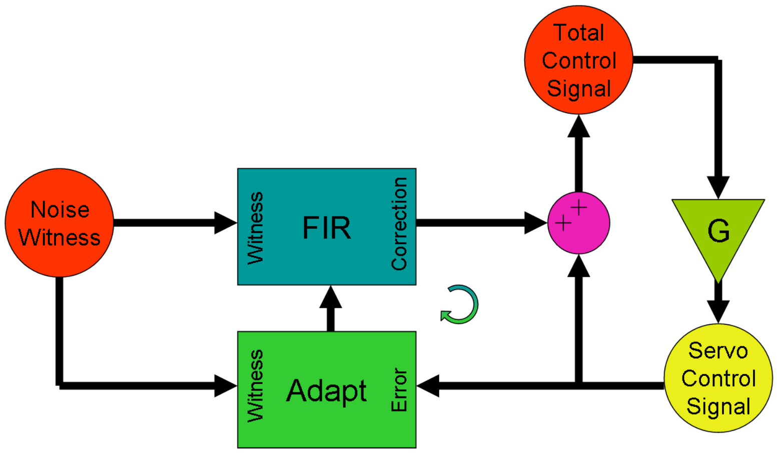

Figure 2: Analogous to LMS arrangement, but with high-gain loop.

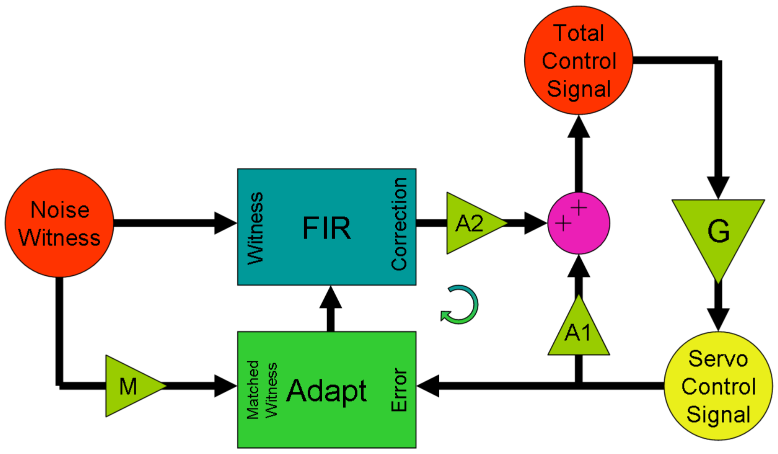

Figure 3: Filtered-X functional blocks with high-gain loop.

adaptation one must ensure that

$$H_{corr \rightarrow err} = -H_{match} \tag{4}$$

where H_{match} is the filter applied to the witness signal before it is sent to the adaptation block. More accurately, it is sufficient to ensure that the phase of H_{match} is close to that of $-H_{corr \rightarrow err}$. A non-unity magnitude will effect the adaptation rate, but it should not prevent stability. This fact can be exploited to target an adaptive filter to a limited frequency band.

3 Implementation

The OAF implementation currently running at the 40m is shown in figure 4. The OAF is quite minimal in that it only implements the FIR, up and down sampling, and adaptation. In order to have a functional system, one must add anti-alias and anti-image filters, as well as compensation and matching filters, as described in the previous section. The configuration parameters of the OAF are:

N_{TAP}	the number of FIR coefficients for each witness channel
R_{dnsamp}	the down-sampling ratio
N_{Aux}	the number of witness channels

These parameters are changed any time the RESET value changes to a value other than zero. Additionally, there are 3 other values that take immediate effect when changed:

μ	the adaptation rate
τ	the FIR coefficient decay rate (fractional loss per downsampled cycle)
$delay$	the number cycles of delay (front-end clock cycles)

The delay given in this field is added to the internally calculated sample-and-hold delay inherent to the up and down sampling performed by the OAF.

The FIR coefficient decay is a feature I added to allow the OAF to forget transients. Typically very small values are sufficient (e.g., $\tau = 10^{-4}\mu$), though this value depends on what you are trying to convince the OAF to do. Setting $\tau = 1$ can be used to clear the FIR coefficients.

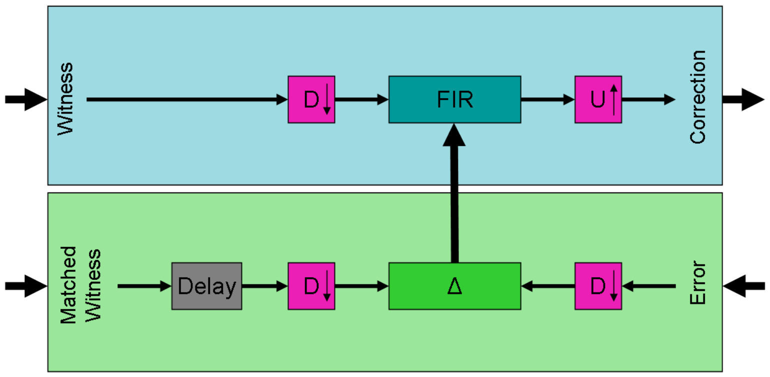

Figure 4: Internal blocks of the OAF implementation.

The OAF implementation at the 40m is very limited, and intended for use only as a prototype or proof-of-principal. It currently exists as a Bork-Space function call block (XFCODE), and is written in such a way that only one such block can exist on a given front-end. The computation is inefficient and poorly distributed, resulting in a limit of about 7 witness channels with 1000 FIR coefficients each. The code, `TOP_XFCODE.c`, can be found in the front-end source directory.

4 Tuning

Since the OAF up and down samples the data, you, the user, must make anti-alias and anti-image filters. These filters should be place in the `ERR_EMPH`, `CORR`, and `PEM_n` filter banks. Since compensating these filters would make them worthless, they have to be matched. This is most easily done by placing the product of the `ERR_EMPH` and `CORR` (AA and AI) filters in the `PEM_n_ADPT` filter banks.

Still, to make the OAF work, the stability requirement given in equation 4 must be met. To do this you should measure the TF from the correction signal to the error signal and either compensate it or match it (ignoring the AA and AI filters which you have already accounted for). Since there is no easy external way to compensate or match delay, the OAF has a delay buffer. Compute the delay in front-end cycles from the phase at the Nyquist frequency of your downsampled system (i.e., for a front-end that runs at

2048 and a down-sample ratio of 16, you should take the phase at 64Hz and multiply by 16/180). Of course, for a pure delay you can use any frequency, but this is the one at which the OAF is most sensitive to unmatched delay.