Thesis draft:
Online processing of the large area SiPM detector signals for the DarkSide20k experiment

Giacomo Petrillo

January 24, 2021

Contents

Chapter 1

Signal to noise ratio of PDM signals after filtering

The DarkSide20k detectors will use photodetector modules (PDMs) made up from many silicon photomultipliers (SiPMs). For what concerns us, the output is similar to a single SiPM output. When a photon hits the photomultiplier, the electrical output is a sudden voltage spike, with a rise time on the order of nanoseconds, which decays slowly in some microseconds. The amplitude of this signal is proportional to the number of photoelectrons produced, so it is discrete, apart from some random fluctuation. See figure 1.1 for an example.

Our goal is to study the performance of some filters in finding and measuring the amplitude of the signals amidst electrical noise. We'll now introduce the dataset, list the filters tested, define a performance measure and show the results. Finally we'll compute and comment the noise spectrum. The code for this work is online at `https://bitbucket.org/Gattocrucco/sipmfilter/src/master/`.

1.1 Data

For this study we used test data taken at liquid nitrogen temperature from a setup in the Gran Sasso National Laboratories (LNGS). A laser pulse is shot at regular time intervals on the PDM. Both the laser trigger and the detector output are sampled at 1 GSa/s with a 10 bit ADC and saved separating the data in "events" where each event correspond to a single laser pulse. See figure 1.2.

We used the PDM slot 8 data, which as per figure 1.3 corresponds to tile 57

Make a figure with 1, 2, 3 photoelectrons signals from LNGS data; number of photoelectrons maps to tone, repetition maps to alpha. Do 3 repetitions.

Figure 1.1:

A figure with some events. Maybe start from plotwav2.py.

Figure 1.2:

Import PDMadcCh.png

Figure 1.3:

(what are these tiles anyway?) which means the data is in the directory `http://ds50tb.lngs.infn.it:2180/SiPM/Tiles/FBK/NUV/MB2-LF-3x/NUV-LF_3x_57/`. We used the file `nuvhd_lf_3x_tile57_77K_64V_6VoV_1.wav`.

In the dataset there are a couple of problems. The first is that signals with many photoelectrons saturate, however this won't trouble us since we'll need only single photoelectron signals. The second is the presence of some spurious signals which do not correspond to the laser pulse. I filtered these out by putting a threshold in the part of each event *before* the laser trigger, which should be flat apart from the noise; there were 72 of them out of 10 005 events.

In principle a spurious signal arriving *after* the "official" laser-induced signal matters too, however I'm ignoring them out of this logic: spurious signals hitting earlier raise the official signal in a somewhat uniform way with their slowly decaying tail, so the detected amplitude will have a bias which is significant, but possibly small and as such not identifiable. Spurious signals hitting later will add a large spike in the tail of the official signal, so the amplitude will be noticeably higher, such that a single photoelectron pulse gets confused as a double or higher one, and we'll automatically ignore it as we'll consider signals detected as single.

This reasoning may fail depending on the details of filtering and the specific relative timing of the signals, however the final most important consideration is that I expect less than 100 spurious late pulses since there are 72 early ones and the laser pulse is in the middle of the event, so less than 1 %. See figure 1.4 for some examples of spurious/saturated signals.

The afterpulse stuff I had not understood properly may change all this.

1.2 Filters

A filter operates by converting the original sequence of ADC samples $(x_1, x_2, \ldots)$ to a new "filtered" sequence $(y_1, y_2, \ldots)$. The filters are causal, i.e. the filtered sample y_n can be computed only using the original samples up to x_n. This limitation is because we are interested in using the filters online, i.e. produce the filter output continuously as samples are read.

We tested three filters: the moving average, the exponential moving average or autoregressive filter, and the cross-correlation filter.

Spurious and saturated signals. Start from plotwav.py.

Figure 1.4:

The moving average consists in taking the average of the last N samples:

$$y_n = \frac{1}{N} \sum_{i=1}^{N} x_{n-N+i}. \tag{1.1}$$

The exponential moving average weighs past samples with an exponentially decaying coefficient, and can be written recursively as

$$y_n = a y_{n-1} + (1-a) x_n, \quad a \in (0,1). \tag{1.2}$$

The scale of the exponential decay is given by

$$\tau = -\frac{1}{\log a}, \tag{1.3}$$

$$\approx \frac{1}{1-a} \text{ for } a \text{ close to 1.} \tag{1.4}$$

The cross-correlation filter is the most sophisticated we considered. Let $\mathbf{h} = (h_1, h_2, \ldots, h_N)$ be a *template* of the signal waveform we want to detect. This means $\mathbf{h}$ should ideally match the shape of the signal waveform we want to find in the noisy data. The filter is then the cross-correlation of $\mathbf{x}$ with $\mathbf{h}$:

$$y_n = \sum_{i=1}^{N} h_i x_{n-N+i}. \tag{1.5}$$

Under the assumption that the data is white noise plus a signal that perfectly matches the template apart from amplitude, this filter is optimal in the sense that in the filter output there will be a peak corresponding to the signal and this peak will have the maximum possible height relative to the standard deviation of the filtered noise.

The differences we have from the ideal case are:

1. the shape of the signal probably changes a bit each time;
2. the signal is not aligned always in the same way to the ADC timebase;
3. the noise is not white.

The variation of the actual signal shape is difficult to address directly. The noise spectrum can be corrected by appropriately transforming $\mathbf{h}$, in this case the filter is called *matched filter*. Let $\mathbf{s}$ and $\mathbf{w}$ be the signal and noise, such that the waveform to be filtered is $\mathbf{x} = \mathbf{s} + \mathbf{w}$. Let R be the noise covariance matrix, i.e. $R_{ij} = \mathrm{Cov}[w_i, w_j]$. Then the template for the matched filter is

$$\mathbf{h}_\mathrm{m} = R^{-1}\mathbf{h}. \tag{1.6}$$

We tried implementing the matched filter with mixed results we will not report in detail. We computed the noise covariance matrix on the event samples before the signal. Since the noise is stationary, R is a Toeplitz matrix, i.e. the

Make a plot of the autocovariance. Use `make_template.py` but with a line plot instead of the matrix map.

Figure 1.5:

The cross-correlation filter template.

Figure 1.6:

covariance depends only on the lag between two samples. Figure 1.5 shows the noise covariance obtained.

The matched filter worked slightly better than the cross-correlation filter, as expected, but only for N sufficiently small, getting worse as N increased. This could be due to numerical accuracy problems in solving the linear system R in equation 1.6, or in computing R. We didn't work on this further since the gain is probably small.

We computed the template for the cross-correlation filter by taking the median of single photoelectron signals. We did not align the signals, we operated as if they occurred always with the same alignment relative to the event time window. The template obtained is shown in figure 1.6. When using the template, we normalize it to unit sum such that the output from the cross-correlation filter is comparable to the output from the moving averages, i.e. if we send a flat waveform into the filter, the output has the same value as the input.

Since we will try various lengths of the filter template, we have to decide how to truncate the full template. When truncating to N samples, we pick N contiguous samples from the template such that their sum is the minimum possible (recall the template is negative). Of course normalizing the template to unit sum is done after truncation.

See figure 1.7 for an example of filter output.

1.3 The fingerplot

To measure the performance of filters, we define a signal to noise ratio (SNR) as follows: the SNR is the ratio of the average peak filter output value for single photoelectron signals over the standard deviation of the filtered noise.

We could consider other similar measures, for example we could include the standard deviation of the peak filter output value since that influences where we should place a threshold to discriminate signals, however it is sufficient to use any reasonable definition for the purpose of comparing different filters. Effectively we computed the SNR without exactly respecting the definition above, we'll see

An event from LNGS data with the three filters. I had already done something like this, find it. Change "matched" to "cross-correlation".

Figure 1.7:

A figure like the first one in the November 3 slides.

Figure 1.8:

A reasonable fingerplot. Maybe one that has not perfectly separated peaks.

Figure 1.9:

in detail.

For each event we compute the filter output at a fixed temporal delay from the leading edge of the laser trigger pulse (see figure 1.8). Then we compute the average of the samples before the trigger pulse and take that value as "baseline". We subtract the baseline from the filter output, and finally we change the sign to obtain positive values since the signals are negative.

We take the list of these baseline and sign corrected filter outputs and compute an histogram. One of these histograms is shown in figure 1.9. It is called "fingerplot" due to the descending peaks reminding of fingers.

The first peak is centered on zero and thus corresponds to cases where the laser pulse produces no photoelectrons. Since the instant where we are evaluating the filter output in each event is independent of the output itself (instead of e.g. being determined by peak finding), this peak is the distribution of the noise after passing through the filter.

The various other peaks correspond to an increasing number of photoelectrons. The second peak is the distribution of the filter output at a fixed instant for single photoelectron signals. Assuming for the moment that the instant where we evaluate the filter yields the highest signal response, this means that the SNR is the mean of the second peak divided by the standard deviation of the first.

Since the peaks are often overlapping, and that we will have to repeat the calculations for many fingerplots automatically without checking each one, we use robust measures of location and scale instead of the average and standard deviation. We run a peak finding algorithm on the histogram and divide the data by putting boundaries midway between peaks, so that we assign each datapoint to a peak. For the second peak, we take the median instead of the average. For the first peak, we take the half symmetrized 68 % interquantile range instead of the standard deviation, i.e. half the difference between the 0.84 and 0.16 quantiles. On a gaussian distribution these are equivalent to the mean and standard deviation, however they are less sensible to messing up the tails of the distribution, which happens since the boundaries cuts away the tails and there is contamination from the tails of the neighbouring peaks.

1.4 SNR versus filter length

When determining how to compute the SNR from the fingerplot we assumed that we were evaluating the filter output at the optimal instant. Since we do snrplot. tau-¿filter length (legend title), matched-¿cross correlation.

Figure 1.10:

snrmaxplot. use logscale for the two bottom plots. matched-¿cross correlation.

Figure 1.11:

not know it a priori, we repeat the computation for a range of values of the filter evaluation point. A simpler solution that comes to mind is taking the minimum (the signals are negative) of the filter output in each event, however this would bias upward the SNR measure because the minimum can yield lower values due to noise peaks. Instead, by fixing the point independently from the data, the random oscillation due to noise is symmetrical and the averaging recovers the actual amplitude of the filter output for the signal.

The resulting SNR curves are shown in figure 1.10, repeated for a range of values of the filters length parameter. For the moving average and cross-correlation filter, the length parameter is the number of samples, N. For the exponential moving average, it is the scale of the exponential decay τ.

The maximum of each curve gives the actual SNR figure we are interested in. We expect by intuition that the width of the maximum is approximately proportional to the temporal resolution we could achieve if we used the filter output to locate temporally the signal. So we do another plot (figure 1.11) where we show the maximum SNR value and the width of the peak versus the filter length parameter. We measure the width as the distance between the two points where the SNR is 1 less than its maximum value.

1.5 Effect of the baseline computation

In constructing the fingerplot, we subtract from the filter output value the baseline, i.e. the average value of the waveform in absence of signals. We compute the baseline for each event as the average of the samples before the signal. More precisely, we averaged 8000 samples. The value obtained varies randomly due to the noise; its standard deviation is that of the noise divided by $\sqrt{8000}$. The maximum filter length we use is 2000, so in that case the width of the peaks gets an additional contribution (summed in quadrature) which is $\sqrt{2000/8000} = 1/2$ of the width we would have with a noiseless baseline, i.e. the width of the first peak is $\sqrt{1 + 1/2} - 1 = 22\,\%$ larger, lowering the SNR by the same percentual.

This is not just a problem of this test that can be worked around in the real application, how the baseline is computed is a relevant part of the signal finding, since it requires either a fast online algorithm estimating it reliably or sending enough samples to subsequent stages in the data processing chain.

To get an idea of the effect of the baseline, we repeat everything computing the baseline with just 1000 or 200 samples. The result is in figure 1.12. We see

A new version of snrmaxplot merging the different baselines. Drop the middle plot.

Figure 1.12:

Plot from spectrum.py.

Figure 1.13:

that, for example, going from 8000 to 200 the maximum SNR drops from about 20 to about 9.

1.6 Noise spectrum

We take the region of each event before the trigger pulse, compute its periodogram, and take the median across events for each frequency. We use the median instead of the average in case there were spurious signals or irregularities we missed. We do the same for data obtained with the same sensor but when it was in the Proto0 setup, biased below breakdown voltage and sampled at 125 MSa/s. We thus obtain two plausible noise spectra, shown in figure 1.13.

We said the cross-correlation filter is optimal if the noise is white. Actually, it is sufficient that the noise spectrum is flat in the support of the signal spectrum, because if we filtered away frequencies outside of it, all the signal would still be in the waveform. In figure 1.14 we show the power spectrum of the signal and its integral. We see that 90 % of it is below XXX MHz, which corresponds to an approximately flat region in the noise spectra. This means that the cross-correlation filter is close to optimal.

I've still not done this, hopefully it will turn out to be true.

The CDF of the template power spectrum.

Figure 1.14:

Chapter 2

Temporal resolution of signal detection

In the previous chapter we studied the performance of filters, but we did not define an algorithm to search for signals. Neither we will do it here, we'll skip it and measure the temporal localization precision once we know there's a signal.

To this end, we make a simulation where each "event" contains only one signal at a known position. In principle we could use the LNGS data (section 1.1), but we don't know the jitter of the trigger pulse and we may reach a temporal resolution below the sampling period, while in the simulation we have the exact actual temporal location of signals.

2.1 Event generation

Each event is the sum of a signal and a noise waveform. We don't add a baseline, so the noise has mean zero and the signals taper down to zero. The signals are negative. We use the same scale of the LNGS data; it actually does not matter because we are not simulating digitalization.

2.1.1 Signal

We obtain the signal waveform by averaging single photoelectron pulses from the LNGS data (see section 1.1 for a description of the dataset). We do not try to align the signals, assuming that they are aligned relative to the beginning of each acquisition event. We take 3584 1 GSa/s samples. Figure 2.1 shows the obtained signal template.

(A study not reported here shows that better alignment is achievable but makes a small difference, and worse alignment means the peak of the signal is smeared thus yielding lower performance in signal localization, and so our choice is irrelevant at best, conservative at worst.)

The toy template.

Figure 2.1:

The series of interpolated templates. I think I did this in `savetemplate.py`.

Figure 2.2:

The template is at 1 GSa/s, but the simulation is at 125 MSa/s. We have to downsample the template and shift its temporal position continuously instead of by $(1\,\mathrm{GHz})^{-1} = 1\,\mathrm{ns}$ steps. Given the continuous temporal position where we want to place the template, we round it by excess and defect to the 1 ns timebase. Then we downsample the template by averaging samples in groups of 8, once with the groups aligned to the floor rounded position, once with the ceiling rounded one. Finally we interpolate linearly between the two downsampled templates. Figure 2.2 shows a series of waveforms obtained in this way.

(Downsampling with an average is not the best antialiasing filter doable, but it should be reasonably fine since the higher spectral part is already suppressed in the signal template.)

2.1.2 Noise

We used three different kinds of noise: gaussian white noise; noise sampled from the LNGS data; noise sampled from Proto0 data.

The white noise is generated in the simulation. The LNGS noise is copied from the same data we used to make the signal template by taking the part of the events before the signals and filtering out a few events that contained spurious signals in that location. The Proto0 noise is copied from an acquisition made on the same PDM when it was used in the Proto0 setup with the SiPMs under breakdown voltage, thus inactive.

The noise obtained from data is normalized to the variance required by the simulation. For both LNGS and Proto0 noise the data comes divided in events and we normalized separately for each event in case the variance changed.

We downsample the noise in the same way as the signal, by averaging nearby samples. Both noises have spectra that go down with frequency (see section 1.6), so this crappy antialiasing should be sufficient. The normalization to the desired variance is done after downsampling. The order matters because downsampling with averaging reduces the variance of the noise. See figure 2.3.

A figure similar to the one made by `toy1gvs125m-plot.py` with both LNGS and Proto0 noise and downsampled.

Figure 2.3:

An event. Show all filters together, plot `X`s on the minima after removing localization offsets at the end of `Toy._run`.

Figure 2.4:

The figure with the truncated templates. Was it in `toytest.py`?

Figure 2.5:

2.1.3 Event layout

Each event is the sum of a noise waveform and a shorter signal waveform. Before the beginning of the signal there's a noise-only region long enough for the filters to be in a stationary state when they reach the signal; in particular its length is the highest filter length parameter used in the simulation (2048 ns) plus 256 ns.

The simulation is repeated for various signal to noise ratios (SNR). We define the SNR as follows: the peak height relative to the baseline of the original 1 GSa/s signal template over the noise standard deviation.

Simulations with different SNR differ only in the multiplicative constant of the noise, so we used exactly the same noise and signal arrays for each SNR to speed up the code. This means that there's no random variation between results obtained at different SNR (or with different filters), keep this in mind when the smoothness of some plots would seem to suggest that the Monte Carlo error is negligible.

Figure 2.4 shows a complete example event.

2.2 Temporal localization

We run the three filters described in section 1.2 (moving average, exponential moving average, cross correlation), then take the minimum (the signals are negative) of the filtered waveform as the location of the signal. We also take the minimum of the unfiltered waveform as baseline comparison.

The minimum of the filter output occurs in some sense later than the signal location, this is not a problem since the choice of the point of the signal to be taken as reference is already arbitrary.

To make the template for the cross correlation filter, we first cut the signal template (section 2.1.1) to the required filter length, keeping the part of the template with maximum euclidean norm, then we downsample it by averaging nearby samples. See figure 2.5.

To allow for a localization eventually more precise than the sampling timebase, we interpolate the minimum sample and its first neighbours with a parabola. We also try upsampling the waveform to 1 GSa/s (with sample repetition) prior to filtering.

We repeat the simulation for 1000 events for each filter, filter length parameter, and SNR in some range. Figure **??** shows