SPE Templates Technote

Extracting and fitting procedure

Spencer N. Axani - January 9, 2019

Introduction

The single photoelectron (SPE) templates represent the probability density function for the charge distributions on all in-ice DOMs. The functional form used to describe the distribution is the sum of two exponentials and a Gaussian, Exp_1+Exp_2+Gaus., explicitly:

$$f(q)_{SPE} = E_1 e^{-q/w_1} + E_2 e^{-q/w_2} + N e^{-\frac{(\mu-q)^2}{2\sigma^2}}$$

Where charge, q, is the quantity calculated in WaveDeform. WaveDeform deconvolve the digitized waveform into a pulse series (pulse times and charges). This is accomplished by fitting SPE pulse templates (templates which are dependent on the version of the AC coupling used in the DOM) to the digitized waveforms via a χ^2. The charge associated with a fitted pulse is determined by dividing the the area of the fitted SPE pulse template by the load resistance (and further scaling it by the gain of the PMT). WaveDeform is capable of fitting multiple SPE pulse templates to multi-PE pulses, and deconvolving them into their separate photoelectrons. Since it may occasionally deconvolving a non-multiPE pulse, we must **always** sum up the pulses from WaveDeform to make a comparison between MC and Data (i.e. we need to reassemble pulses that may have been accidentally split).

For this study, all frames (events) that pass the MinBias filter are then passed through WaveDeform. This filter works by randomly selecting HLC events, at a rate of 1/1000 (? check) triggers. The MinBias filter saves the full digitized waveform from the ATWD and fADC. This allows us to rerun WaveDeform with different settings (this will be useful later). This study will also use the BeaconLaunch dataset. This dataset represents forced time windows. That is, it is not triggered by a pulse, rather, it is triggered by a time. Normally, a random force trigger will readout the ADC baseline, and it is rare that it record actual pulse. Since these frames are primarily baseline fluctuations, we can use this dataset to analysis the accidental pulse reconstruction associated with ADC fluctuations (referred to as noise).

This tech note will being with a set of condensed instructions for those familiar in the procedure already. Then, I'll provide much more informative description of every step. We use a modern simulation Metaproject (e.g. V06-00-03), and recent updates to WaveDeform mean that we need to use the current trunk versions (http://code.icecube.wisc.edu/svn/

projects/wavedeform/trunk/). Later, this modified version of WaveDeform will be added to the release. Changing the meta project involves simply replacing the hashtag Metaproject identifier at the top of the processing scripts. WaveDeform is only used in the harvesting.py script.

The inlocation variable here will be defined as:
/data/ana/SterileNeutrino/IC86/HighEnergy/SPE_Templates/

Condensed instructions

My processing files are located here :
inlocation + /SPE_harvesting/scripts/jobs/
The output from these files are sent here:
inlocation +/SPE_harvesting/

Essentially there are four steps. We first harvest the pulses from the MinBias and BeaconLaunch data. Then, we merge all the found pulses found in a season together into a common JSON file. Finally, we fit the distributions using the convolutional fitter and make a GCD file from the results.

Harvesting pulses

1. Run extract_i3.py, by passing it the season name. Like IC86.2011. It will try to find the PFFilt files of that season, and only extract the events that pass the MinBias filter.
2. The events that pass the MinBias filter are now in an i3 file. Send the output file of (1) through harvest.py. This file will run WaveDeform on every frame. It will find the pulses of interest. You must specify the WaveDeform settings, as well as the identifier name. An example identifier is "923_NewWaveDeform," which identifies the presets on WaveDeform (923) and a description (NewWaveDeform). The output of this will be JSON files that contain the individual pulses extracted sorted by each DOM and digitizer.
3. Merge the all the files in (2), by running merge_harvest.py. During this merging process, the individual charges are binned. We use a 1000 bins from 0 to 5PE for the ATWD and 200 bins for the SLC.

The output of the pulse harvesting is a single JSON per season which includes the binned charges for all DOMs, for ATWD0 and FADC (SLC), for both the MinBias and for the BeaconLaunch.

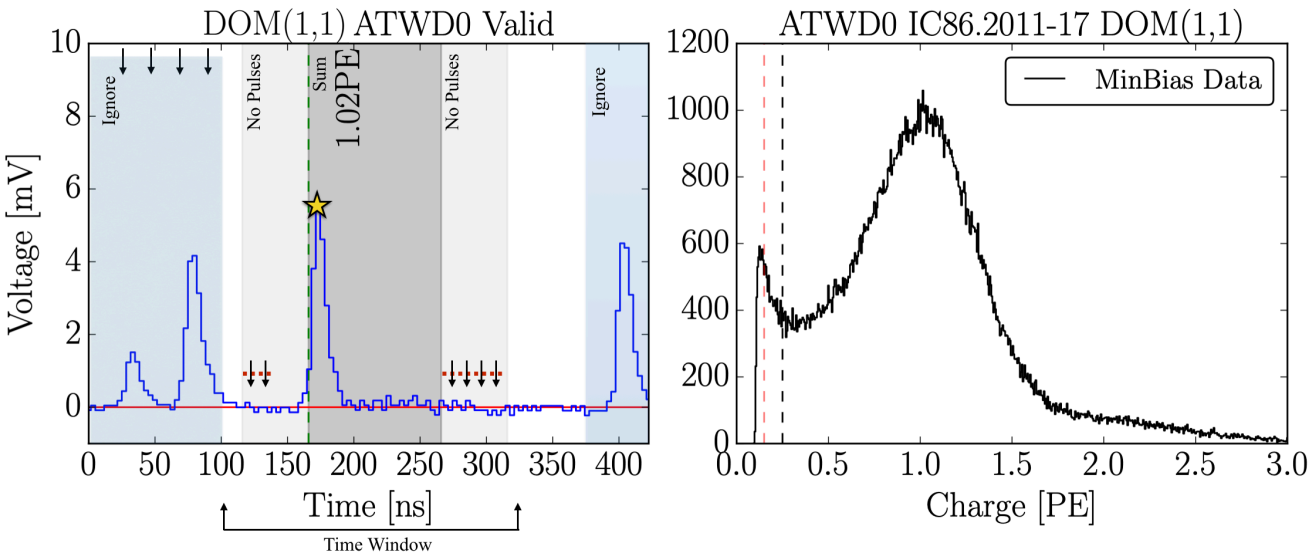
Fitting and making the GCD file

1. Fit one string at a time by passing the string number (1-86) and digitizer (ATWD or SLC) to the convolutional fitter (convolutional_fitter.py). I start the fit for ATWD at bin 30, and bin 14 for the SLC. The convolutional fitter requires the identifier name, which just specifies the file merged in step (3) above.
2. The output will be a JSON file that contains the fit parameters for all the DOM on the desired string and digitizer. These now need to be merged using the merge_convolutional_fitter.py script. The output is a fantastic file that contains all the fit information (plus all other information that I have collected on all the DOMs — it's a really useful file).
3. Pass the fitted JSON file to make_GCD_SPE.py. You must specify what kind of GCD you want. Do you want the Gaussian mean to be at unity? Do you want to simply produce the TA0003 distribution, or do you want to use the SPE Templates?

Extracting pulses

Since we are interested in single photoelectrons, we are using the digitized waveforms from ATWD0 are used. We cannot use the trigger pulse, since this would limit us to the discriminator threshold (0.25PE) therefore we devised a method to extract pulses after the window is open. This allows us to collect charges down to the limit of which WaveDeform will try to extract pulses. The pulse selection is described in this section.

The pulse selection is the method used to extract candidate, unbiased, single photoelectron pulses from data, while minimizing the multi-PE contamination. It avoids collecting afterpulses, rejects late pulses from the trigger, reassembles late pulses, accounts for the discriminator threshold, reduces the effect of droop and baseline undershoot, and gives sufficient statistics to perform a season-to-season measurement. An illustrative diagram of



the pulse selection is shown in the left side of Fig.~\ref{fig::pulse_selection}, while a description of the procedure is detailed below.

Fig. 1: Left: an illustration of the pulse selection. Set the 'pdf' option in SPE_harvest.py to make these plots for every pulse. Right: the extracted charge distribution for DOM 1,1. You could make this plot after JSON files have been merged.

In order to trigger a DOM, the input to the front-end amplifiers must exceed the discriminator threshold. To avoid the selection bias of the discriminator trigger, we ignore the trigger pulse as well as the entire first 100~ns of the time window. Ignoring the first 100~ns has the added benefit of also removing late pulses that could be attributed to the triggering pulse. To ensure we are not accepting afterpulses into the selection, we also enforce that the pulse of interest (POI) is within the first 325~ns of the ATWD time window. In the vicinity of the POI, we check that WaveDeform did not reconstruct any pulses up to 50~ns prior to the POI, or 100-150~ns after the POI (the light-gray region of Fig.~\ref{fig::pulse_selection} Left). This later constraint is to reduce the probability of accidentally splitting a late pulse in the summation window.

Restrictions are put on the full ATWD waveforms as well, such as ensuring that the trigger pulse does not exceed 10~mV (to reduce the effect of the subsequent baseline undershoot due to the AC coupling or other artifacts from large pulses) as well as a global constraint that the time window cannot contain any pulses that exceeds 20~mV.

If a pulse is reconstructed between 100 and 325~ns after the time window is opened and the voltage criteria is met, it is accepted as a candidate photoelectron and several checks are performed on the waveform prior-to and after the pulse. The first check is to ensure that the waveform is near the baseline just prior to the rising edge of the POI. This is accomplished by ensuring that the waveform does not exceed 1~mV, 50 to 20~ns prior to the POI, and eliminates cases where the POI is late pulse. We also ensure the waveform returns to the baseline by checking that no ADC measurement exceeds 1~mV, 100 to 150~ns after the POI (these constraints are illustrated as the red-dotted lines and black arrows in Fig.~\ref{fig::pulse_selection} Left).

If all the above criteria are met, we sum the reconstructed charges from the POI time (given by WaveDeform) to +100~ns (the dark gray area of in Fig.~\ref{fig::pulse_selection}

Left). This ensures that any nearby pulses are either fully separated or fully added (in-case WaveDeform incorrectly split the pulse, and to reassemble late pulses). The 100~ns summation also means that the pulse selection we will occasionally be accepting multi-PE (MPE) events.

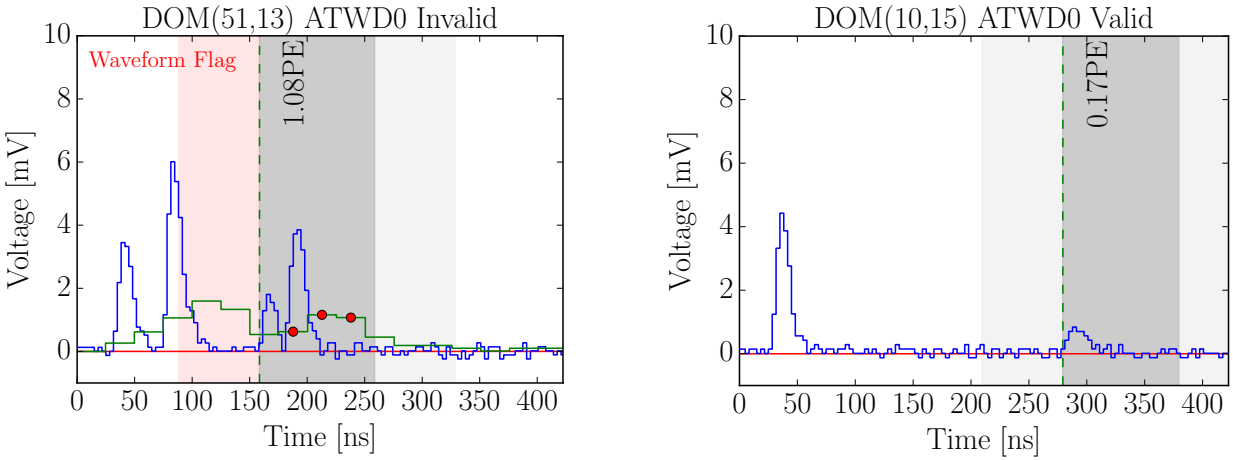


Fig. 3. Left: A failed waveform. It didn't meet the pulse selection criteria. Right: an example of a 0.17PE pulse that was extracted.

Fig. 3 (left) shows a pulse that we reject. The reason for this is that there is another pulse too close to the one of interest. Specifically there is a waveform flag, indicating that the voltage before or after the pulse was too high. You can also see the fADC in green, and the calculated SLC pulse with the red dots. The pulse of interest is the smaller pulse in the grey time window, followed by what looks like a late pulse. The sum of these two pulses is found to be 1.08PE (sum of the two output from WaveDeform). Fig. 3 (right) shows a sub discriminator threshold pulse (0.17PE).

How to extract Pulses

Pulses are extracted using the harvest.py script. This script requires:

1. MinBias files. This file comes from PFFilt, but only contains frames that pass the MinBias filter:

   inlocation + /SPE_harvesting/IC86.2017/i3/2017_10/PFFilt_MinBias_IC86.2017_Y2017_M10_D14_Run130125.i3.bz2

2. The correct GCD file for the PFFilt file:

   /data/exp/IceCube/2017/filtered/level2/1014/Run00130125/Level2_IC86.2017_data_Run00130125_1014_68_344_GCD.i3.zst

3. The processing script harvest.py requires several options:
   1. Infile - the name and location of the MinBias file
   2. Outfile - the name and location of the output file
   3. GCDfile - the name and location of the gcd file
   4. Pdf - This is the desired location to save the waveform of the pulse. If not specified, the pulse will not be plotted. Setting this option, say to "/data/user/saxani"ike the one shown in Fig. 3.
   5. NFrames - The number of frames to process. Set to 0 to process the full file.
   6. Skip_beacons - set to string "True" to skip the beacon pulses and only look at the non-beacon pulses. Otherwise, it will run the BeaconLaunches first, then look through the non-BeaconLaunches.
   7. Tolerance - this is a parameter passed to WaveDeform. It defines a metric of how good the SPE pulse fit should be. The default value, and that used for processing normal IceCube data is 9. I set this to 6 in order for WaveDeform to reconstruct smaller pulses.
   8. Noise threshold. This is a parameter passed to WaveDeform. All ADC values below this will be set to zero. Default value for IceCube data is 2.
   9. Basis threshold. This is also a parameter passed to WaveDeform. It defines the minimum size of bin to consider fitting. Default value for IceCube is 3.

I run setup-harvest.py, which generates the dagman used to submit all my jobs. In this script, I define the IceCube season and identifier name — a string which defines essentially the name of the output folder. The values for the Tolerance, Noise, and Basis Threshold are added to the identifier. For example, for the year 2011, and an identifier NewWaveDeform, the data from harvest.py is sent to this folder:

invocation + /SPE_harvesting/IC86.2011/923_NewWaveDeform

The data in this folder is all the pulses that pass the pulse selection, organized by month, by DOM, in JSON files. It is also separated into pulses extracted from ATWD0 and from calculated SLC pulses. The file should be pretty self-explanatory. The next step is to merge all the files in a given IceCube season. This is done with the merge_harvest.py script.

Again, this can be set up using the setup-merge_harvest.py script. In this script, I specify the identifier, or 923_NewWaveDeform (in the previous example). It then generates a dagman that runs jobs to merge all the JSON files from the seasons selected in this script. The merge_harvest.py script also histograms all the ATWD charges into 1000 bins from 0 to 5PE (The SLC binning is more coarse, we use 200 bins from 0 to 5PE). The merged files are sent to, for example:

inlocation + /SPE_harvesting/Final_charge_distributions/IC86.2011_923_NewWaveDeform.json

Suppose you want the histogram of the charge distribution for DOM (1,1) from IC86.2011 from the WaveDeform with the settings 923, then open the file above with a JSON reader like so:

```
import json

infile = inlocation + '/SPE_harvesting/Final_charge_distributions/IC86.2011_923_NewWaveDeform.json'
with open(infile) as data_file:
   data = json.load(data_file)

charge_dist = data['1,1']['MinBias']['hist']['ATWD']['binContents']
number_of_bins = data['1,1']['MinBias']['hist']['ATWD']['nbins']
min_charge = data['1,1']['MinBias']['hist']['xmin']
max_charge = data['1,1']['MinBias']['hist']['xmax']
```

You could also look at SLC rather than ATWD, or BeaconLaunch rather than MinBias, for any in-ice DOM. There aren't many BeaconLaunch pulses, since the BeaconLauch is a forced time window, which means that a pulse will either be accidental or noise. Fig. 4 shows the charge distribution for summing over all DOMs, over all seasons, for the MinBias and the BeaconLaunches. These are subsequently divided into the set of data with WaveDeform set to 923 and 623. The 623 set is simply used to try and determine the Low charge region.

Fig. 4 shows that BeaconLaunch data in red, and the MinBias in black. The BeaconLaunch data was scaled by 28.4, which is the ratio of the MinBias lifetime to the BeaconLaunch live time. The noise is determined by subtracting the shape of the MinBias data by the BeaconLaunch data. We see when we modify WavDeform (Fig. 4 right), the amount of noise increases (to a maximum of 1.7% of the data), but allows us to examine the distribution down to approximately 0.10PE. We can verify the factor 28.4 by plotting the low charge region for different settings on WaveDeform. After subtracting the background from

the MinBias data, we should see rough agreement between settings. This is shown in Fig. 5 left. If the scaling factor was too large, one would see something like Fig. 5 right.

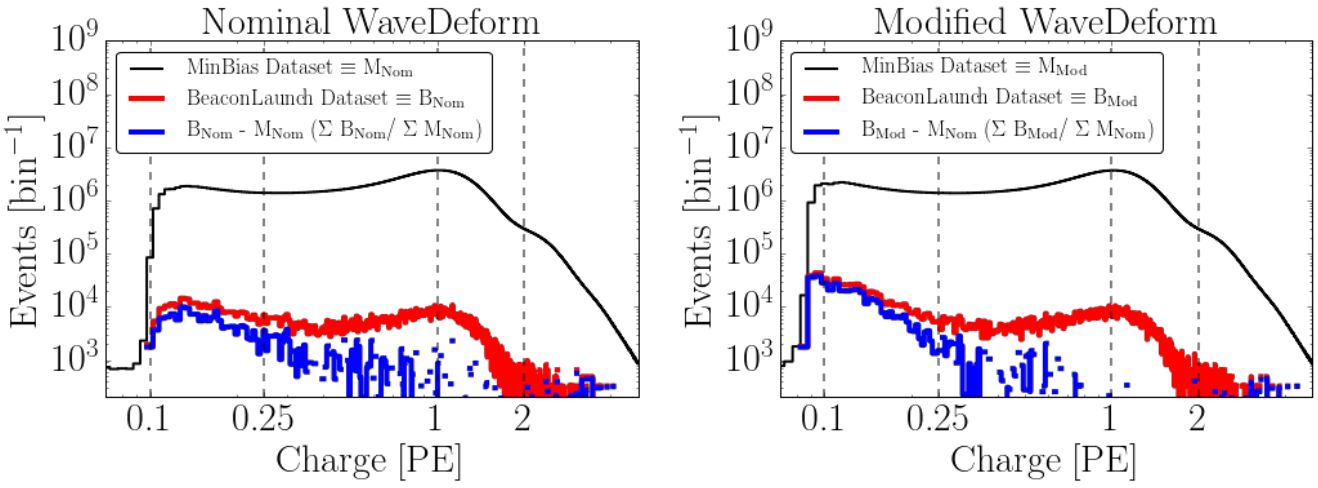


Fig. 4. Left: The normal 923 settings for WaveDeform. Right: stetting WaveDeform to 623.

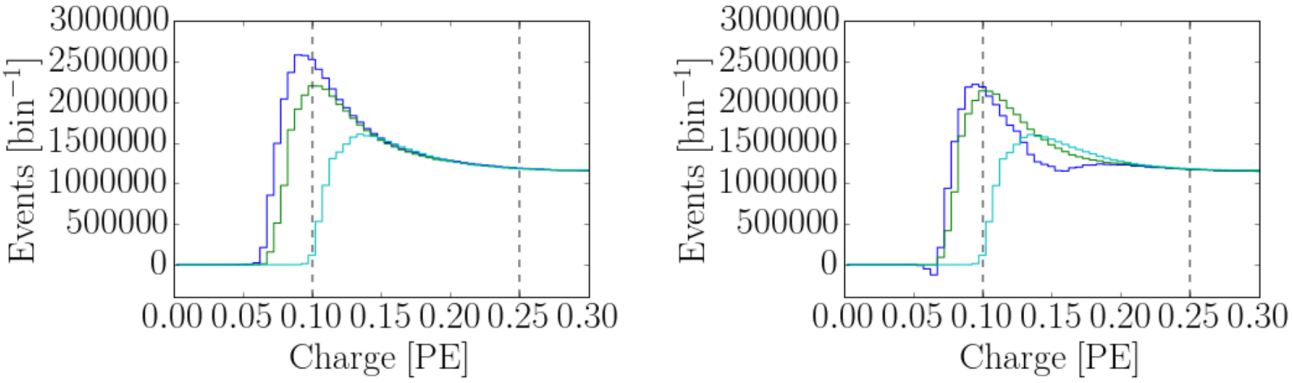


Fig. 5. Using three different settings for WaveDeform 923 in light blue, 723 in green, and 623 in dark blue. Left: scaling factor between MinBias and BeaconLaunch set to 28.4. Right: scaling factor set to 100. You can see that with the improper scaling factor, the curves do not agree.

Convolutional Fitting

As of now, we have the charge distributions for all seasons, ATWD, SLC, per DOM, for two settings of WaveDeform (923 and 623). Now, we use the convolutional fitter to fit the background subtracted data. This is done using the convolutional_fitter.py script located here:

inlocation + /SPE_harvesting/scripts/jobs/convolutional_fitter.py

This file takes a few arguments.

1. Identifier - for example 923_NewWaveDeform.
2. Starting bin - this is the bin you wish to start fitting at. ATWD 923 I set to 30 (0.15PE), SLC 923 I set to 14. For the 623 set, we can lower the starting bin to 20 (0.1PE).
3. year — this is the IceCube season. It selects the year to fit, as well used for the titles on the plots, as well as the name of the outfile.
4. Pdf location — if this is set to a directory, the plot of every fit will be saved to that directory.
5. set_exp — if this is set to the string 'True' then it will use the default shape of the EXP1, ie. 6.9 amplitude and 0.032 for the exponential decay. If set to False, you should be using the 623 set, and it will try to determine the shape of the exp1.
6. DOM — if specified, a string like 1,1, it will only fit that single DOM.
7. String — if specified, it will only fit that string. Example, set to 1 to only fit string 1. When I submit jobs to the cluster to fit, I fit one string per job.
8. digitizer — You can specify either ATWD or SLC. This means that it will fit the ATWD or the SLC distribution from the input file.

An example fit for DOM 1,1 with the year set to AVG (sum over 2011-2016 data), the set_exp to True, the starting bin is 30, ATWD, is shown in Fig. 6. In this figure, you can see that the exponential EXP1 is set to the default value of 6.9 and 0.032 decay width. The other components of the fit: EXP2 and Gaus are shown in blue, along with the 2PE component, determined by the convolutional fitter. The purple distribution shows the Full (all years, all DOMs) BeaconLaunch dataset, scaled by 28.4, then scaled again by a factor of 30 such that it is visible. The charge data from the pulse selection is shown in the black histogram, while the full convolutional fit is shown in the black line. The fit values, are all shown in the description on the side, along with the uncertainty. The large uncertainty in the EXP1 component is due to the fact that we are fitting the 923 set, which can't say much about the low-PE region. The text in the plot also shows the number of entries, the validity (whether the fit succeeded), the

peak-to-valley ratio, the reduced chi2 of the distribution and fit, and the mean charge of the fit.

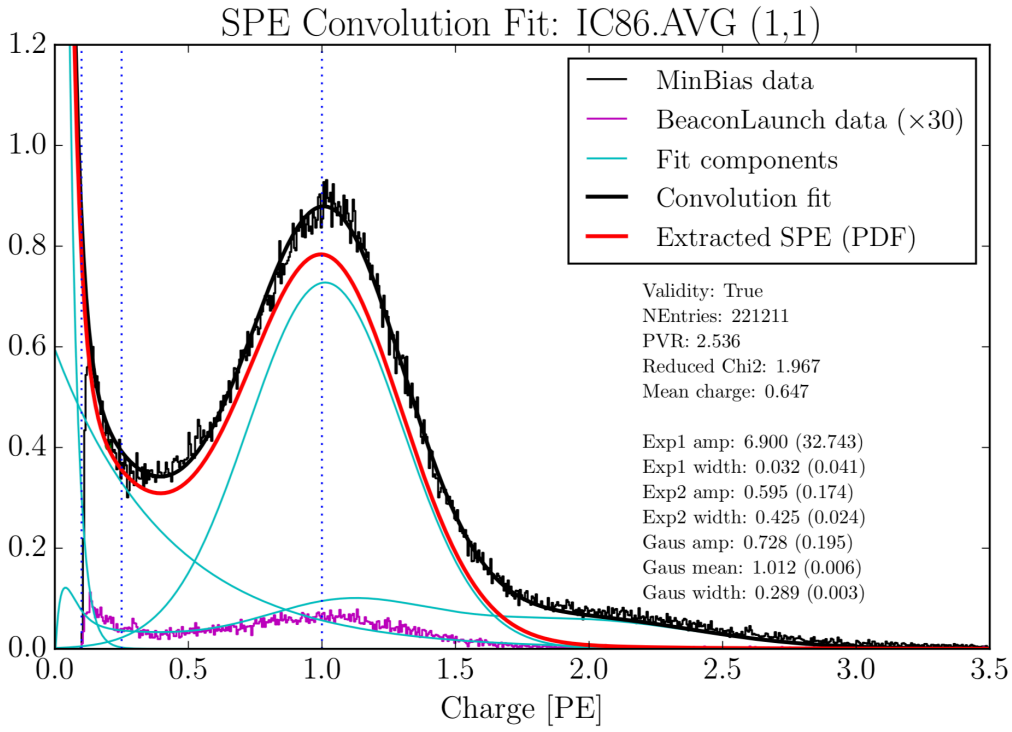


Fig. 6. An example output of the convolutional fitter for the ATWD, DOM 1,1 of all the years data summed together. The SPE Template is the red distribution.

The output of the convolutional_fitter.py is a JSON file that contains all the fit information. I perform the fitting on the cluster using the setup-convolutional_fitter.py script. Which will fit each string, per job. Then I run the merge_convolution_fitter.py script located:

inlocation + /SPE_harvesting/scripts/jobs/merge-convolutional_fitter.py

In the merging file above, it goes through the strings, and merges all the JSON files together. There's often 1 string that might fail on the cluster. If you re-run it, it will succeed. This merging script also populates the final JSON file with all the information related to each DOM. Have a look at this example output file:

inlocation + /SPE_harvesting/SPE_fits/Fits_923_NewWaveDeform/IC86.2011_923_NewWaveDeform.json

In this file, you can find a ton of information, such as:

ATWDFitParams: an array of the components of the fit.

ATWDFitParams_err: the error of the fit values

ATWD_fit: The information related to the output of the convolutional fitter for the ATWD.

Absorption: This is the Spice3.2 absorption value at the depth of the DOM.

AnodeSens: The sensitivity of the anode. As Chris Wendt, I'm not sure what the units are.

BlueSens: DOM sensitivity to blue light?

CalDate: I retrieve the HV and Gain values of every DOM at one point per year to compare. These are the dates of the measure HV and Gain settings.

CathSens: DOM cathode sensitivity?

DOMid: the id of the DOM

DeadDOM: if the DOM is labelled as dead in the GCD file, I set this to 1.

DeepCore: boolean if the DOM is part of DeepCore.

DeploymentDay: the MJD date that the DOM was deployed in-ice

DeploymentSeason: the month and year of the deployment.

Depth: The position of the DOM. Distance from top of the ice.

Gain: I record the gain once per year. The measure time is given in the CalDate.

GainAVG: simply the average of the Gain value above.

HQE: if the DOM is HQE, set to 1.

HV: I record the HV of the DOM once per year. The time is given in CalDate.

HV1e7: this is HV setting to get 1E7 given by Hamamatsu? or lab?

HVAvgInIce: The average of the HV values above.

HVSlope: the slope of the HV values above.

MBid: the id of the main board.

MJD: The modified julian date of the deployment.

MeanCharge: the calculated mean charge of the ATWD fit.

NEntries: the number of entries in the ATWD fit.

NickName: the DOM nickname

OM: the optical module number.

Order: the DOM deployment order?

PMTSerial: the serial number of the PMT. We are missing some of these.

PVRInIce: This is my measurement of the PVR.

PVRLab: This is Hamamatsu's measurement of the PVR.

QE: set to 0 if dead. Set to 1 if standard quantum efficiency, set to 2 if HQE.

RDE: This should be 1 for standard and 1.35 for HQE.

RelSens: relative sensitivity? Ask Chris.

SLC_Fit: the fit values from the SLC.

Scattering: the Spice3.2 scattering values at the depth of the DOM (not taking into account position, just depth).

String: The string that the DOM is on.

TemperatureCalc: Temperature calculated from the depth fit.

TemperatureInIce: the measured onboard DOM temperature sensor.

Toroid: the version of the AC coupling used on the DOM. 1 for old, 2 for new. 0 for dead.

Valid: is the fit valid? 1 for true.

Velocity: the velocity of the DOM.

VelocityGradient: the calculated velocity relative to the DOMs below and above it. So 0.01m/yr means that this DOM is moving 1cm per year relative to the DOM above and below it.

X, Y, Z: the position of the DOM.

Making a GCD file with the SPE Templates

The JSON file that is output from the merge_convolutional_fitter.py script contains the values of the parameters in Eq. 1. These values represent the SPE Templates. We now need to insert them into the GCD file. There is a script to do this called inject_spe_templates.py.

inlocation + /SPE_harvesting/scripts/jobs/inject_SPE_templates.py

This file takes several arguments:

1. Infile - the file containing the SPE Templates. From the previous step.
2. Outfile location - this is the location where you want to save the output of this csript.
3. **year —This is the IceCube season you want to make the GCD for. You will need to modify the script to take in the correct GCD file for you. For MC, it finds the appropriate GCD file to input the SPE_Templates.**
4. Unity - set to string "True", if you want the SPE templates to be scaled such that the gaussian mean is a 1.0
5. Pass2 - set to string "True" if you want this GCD file to be compatible with Pass2 data. Essentially it scales the SPE Templates to match the shift performed during the Pass2 data shift.

6. Pass1 — not tested. But should be good to create a GCD file with the Gausian means set to the values that they were in Pass1 data.
7. default — Set to True if you want to use the TA0003 distribution, not the SPE templates.
8. SPE — set to true you want to use the SPE Templates.

The options above allow you to specify two things: the scaling factor (ie. how to stretch the distribution), and the shape of the charge distribution (SPE Templates, or default TA0003).

So, for example you could set Pass2 and SPE to "True." This would make a GCD file that uses the SPE templates, and scales the distributions to match Pass2 data.

You could set Pass2 and Default to 'True.' This would use the TA0003 distribution by scale the distribution such that the gaussian means matches that of Pass2 data.

You could set Unity and SPE to 'True.' This would use the SPE templates and scale the distribution such that the gaussian means are at unity. This is what would be used for Pass3.

The inject_spe_tempaltes. py script requires an updated version of data classes and DOM laucher. At the top of the script, the meta project that I use is shown in #Metaproject line.

This script creates a field in the domCal called combined_spe_charge_distribution. This holds the fit values for the SPE template associated with that DOM. There is one caveat to this, it was found that the HQE DOMs in IceCube have a measurable different shape than the standard DOMs. This caused the average charge of their distributions to be 3.7% lower than the standard DOMs. In simulation, this would look like a 3.7% change in DOM efficiency for the HQE DOMs. In order to take this into account, we scale the compensation factor by 3.7% (the actual value is calculated in this script).

Anyways, the output of this file will be a GCD file with the update SPE Templates. It will also create a folder with a couple plots that show the distributions of the measure SPE templates values — just for verification that everything went well (e.g. if set to UNITY, one would find that the GausMean.pdf plot shows all the gaussian means to be at 1.0).

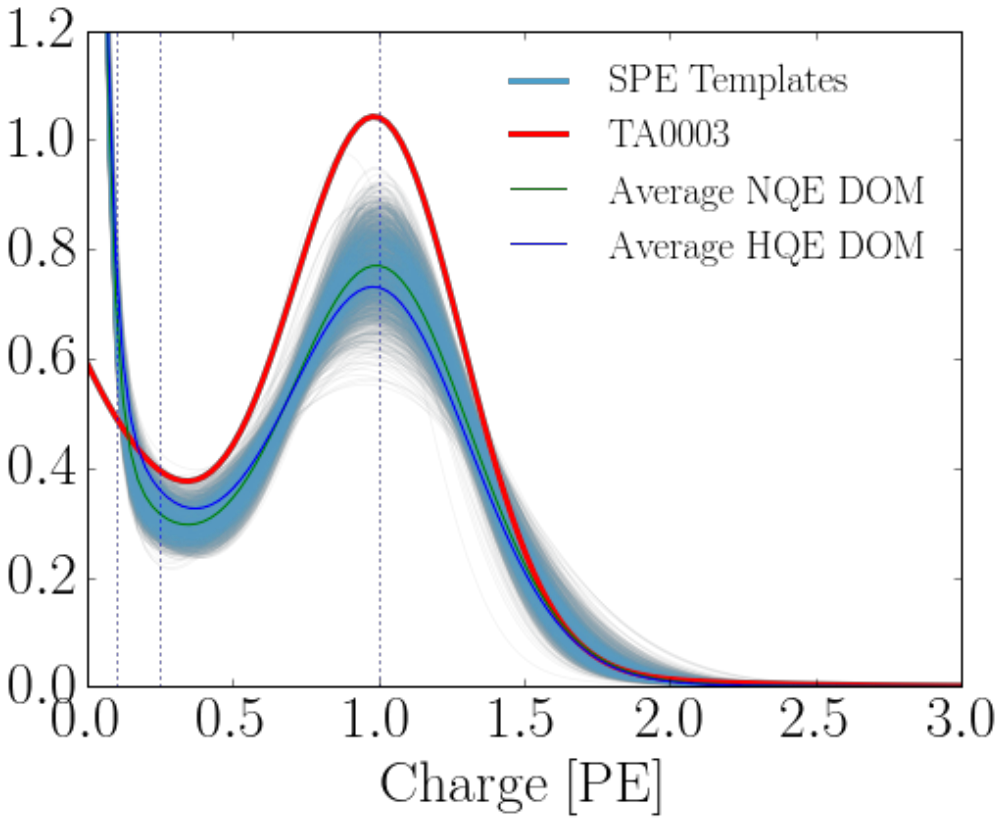
This GCD is now ready to be used for Simulation. If using it for data, make sure that you have the correct initial GCD file — this script only modifies GCD file.

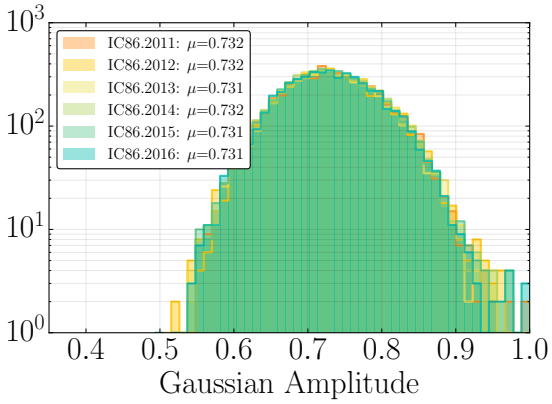
The Compensation Factor

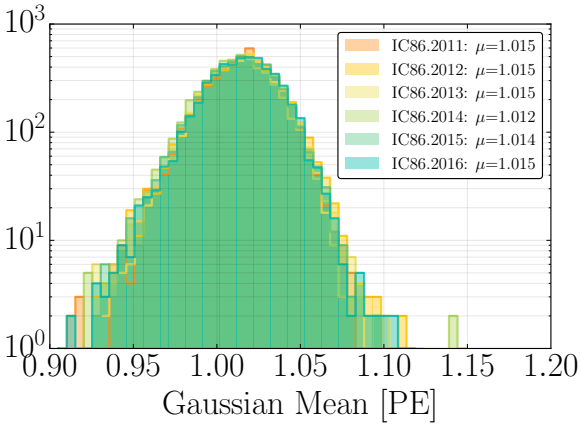
One last thing to mention is the compensation factor. It is a new concept in IceCube. When we measure the SPE templates, we found that the shape was different that what was previously used. The average charge of the measured SPE Templates was about 0.65, whereas the average charge of the TA0003 distribution was 0.85. So, if you wanted to build up enough charge on the detectors, you would need more photons using the SPE Templates — approximately 30% more photons. This is what the compensation factor does. It works the same as the RDE. It simply scales the DOM efficiency by the ratio of the average measured charge such that 0.99 DOM efficiency using the SPE templates, still looks like 0.99 using the TA0003, although behind the scene, it actually ran 30% more photons. The compensation factor is also calculated in the inject_spe_templates.py script, and added to the GCD file.

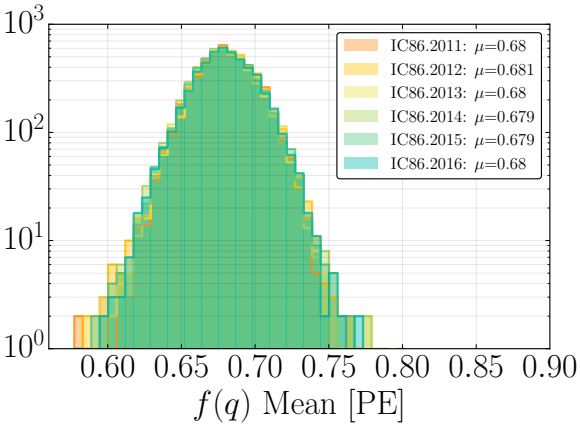
Results from fits

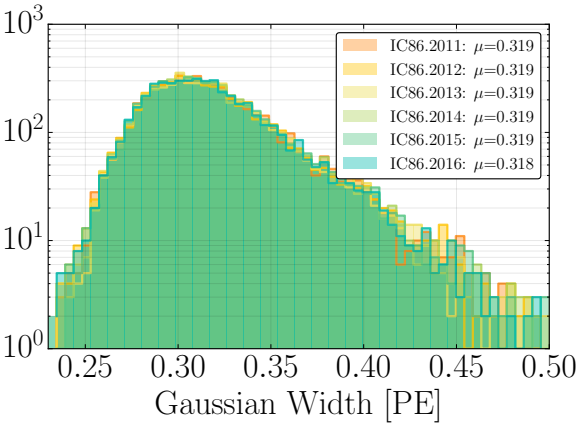
This section will just contain all the plots that I typically generate every time the SPE Templates are re-fitted. I won't describe each plot, although they should hopefully be self-explanatory. If not, feel free to send me an email at saxani@mit.edu**. More information may be available in the paper, as well as in my upcoming thesis.**

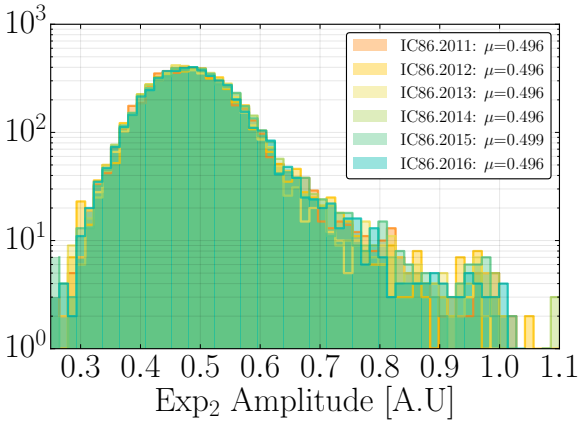


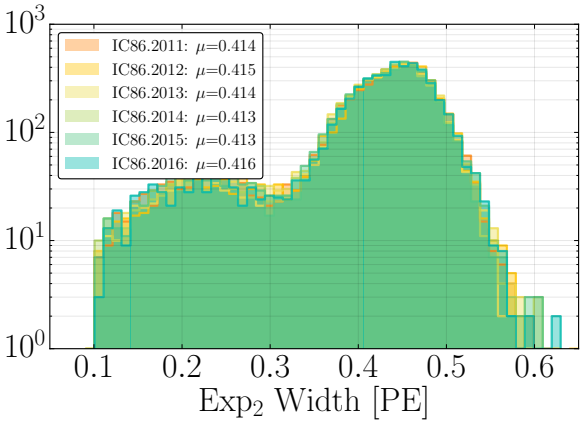

IC86.2011: μ=0.732
IC86.2012: μ=0.732
IC86.2013: μ=0.731
IC86.2014: μ=0.732
IC86.2015: μ=0.731
IC86.2016: μ=0.731
Gaussian Amplitude

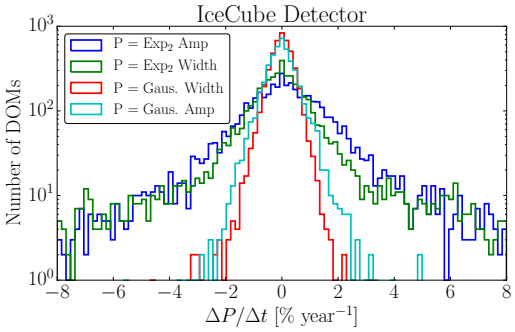

IC86.2011: μ=1.015
IC86.2012: μ=1.015
IC86.2013: μ=1.015
IC86.2014: μ=1.012
IC86.2015: μ=1.014
IC86.2016: μ=1.015
Gaussian Mean [PE]

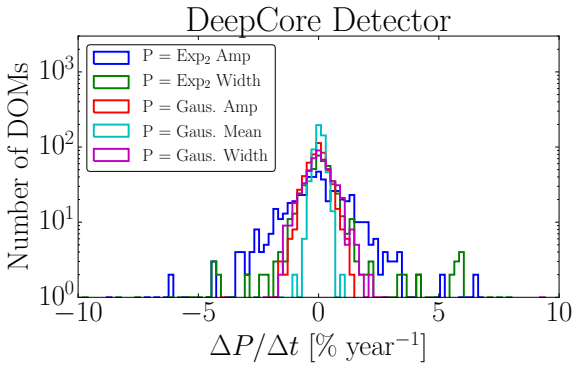

IC86.2011: μ=0.68
IC86.2012: μ=0.681
IC86.2013: μ=0.68
IC86.2014: μ=0.679
IC86.2015: μ=0.679
IC86.2016: μ=0.68
$f(q)$ Mean [PE]

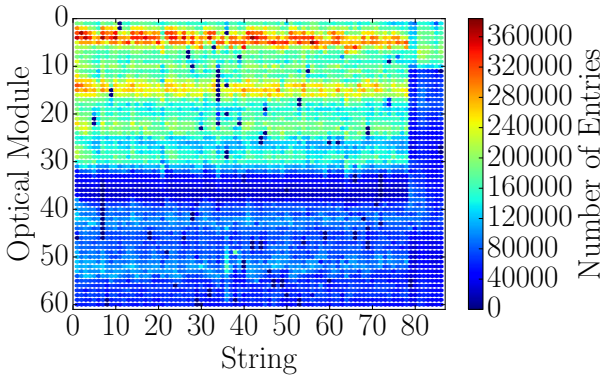

IC86.2011: μ=0.319
IC86.2012: μ=0.319
IC86.2013: μ=0.319
IC86.2014: μ=0.319
IC86.2015: μ=0.319
IC86.2016: μ=0.318
Gaussian Width [PE]

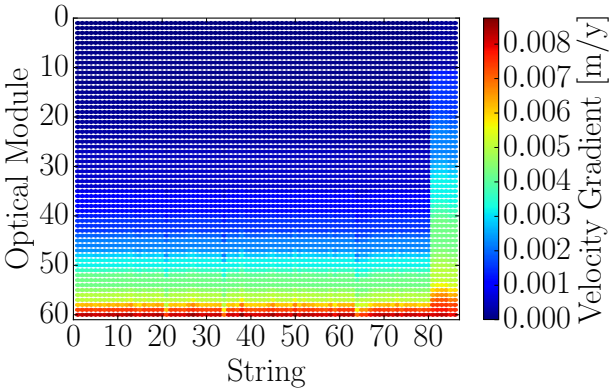

IC86.2011: μ=0.496
IC86.2012: μ=0.496
IC86.2013: μ=0.496
IC86.2014: μ=0.496
IC86.2015: μ=0.499
IC86.2016: μ=0.496
Exp$_2$ Amplitude [A.U]

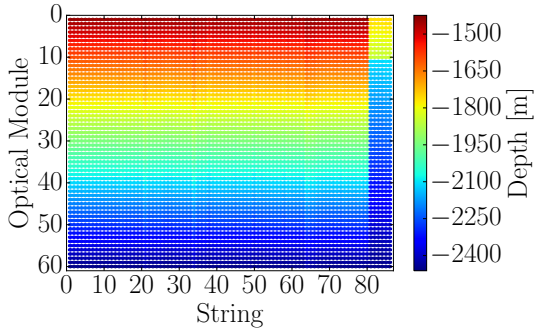

IC86.2011: μ=0.414
IC86.2012: μ=0.415
IC86.2013: μ=0.414
IC86.2014: μ=0.413
IC86.2015: μ=0.413
IC86.2016: μ=0.416
Exp$_2$ Width [PE]

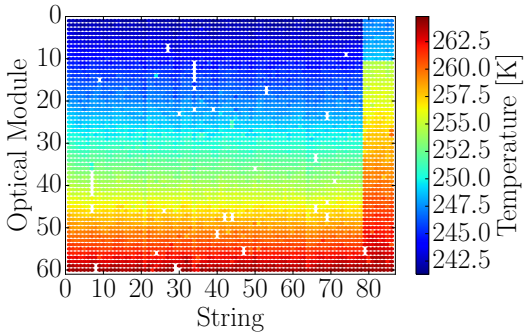

IceCube Detector
P = Exp$_2$ Amp
P = Exp$_2$ Width
P = Gaus. Width
P = Gaus. Amp
Number of DOMs
$\Delta P/\Delta t$ [% year^{-1}]

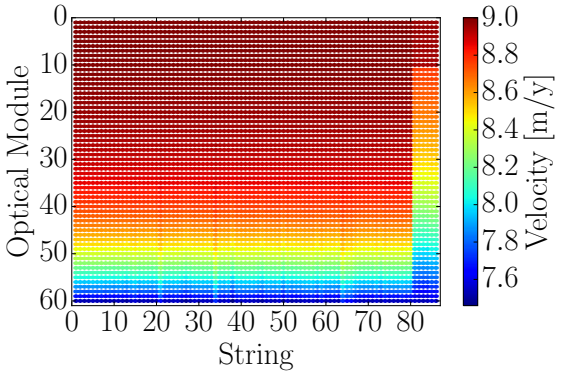

DeepCore Detector
P = Exp$_2$ Amp
P = Exp$_2$ Width
P = Gaus. Amp
P = Gaus. Mean
P = Gaus. Width
Number of DOMs
$\Delta P/\Delta t$ [% year^{-1}]

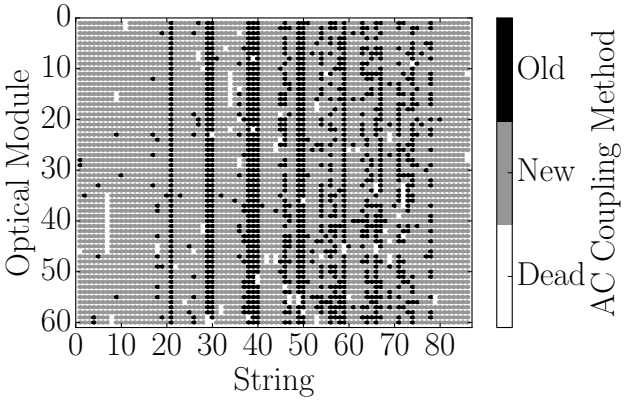

Optical Module
String
Number of Entries

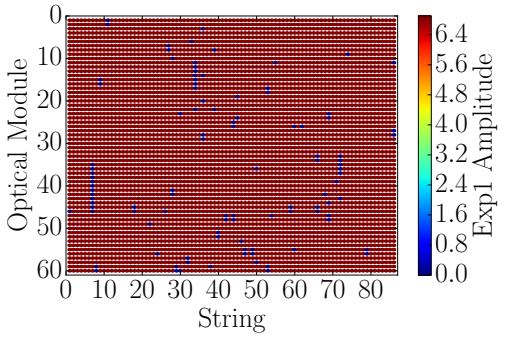

Optical Module
String
Velocity Gradient [m/y]

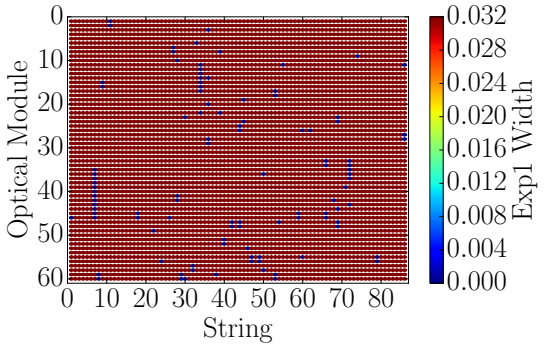

Optical Module
String
Depth [m]

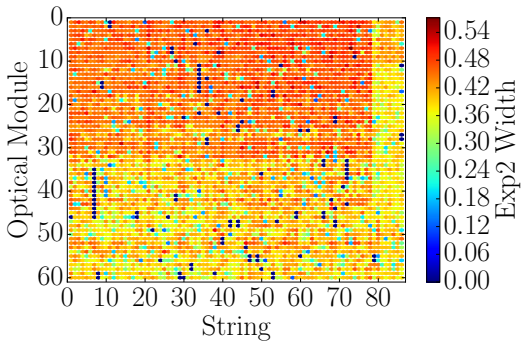

Optical Module
String
Temperature [K]

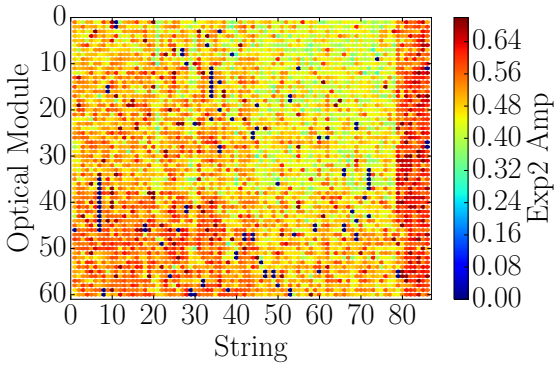

Optical Module
String
Velocity [m/y]

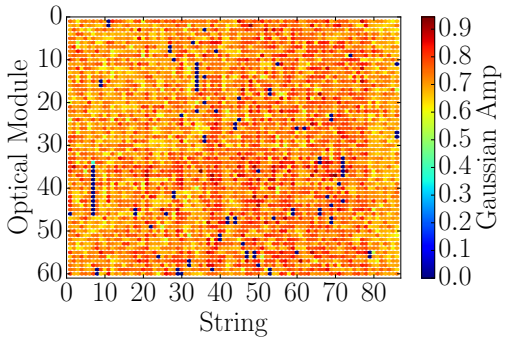

Optical Module
String
Old
New
Dead
AC Coupling Method

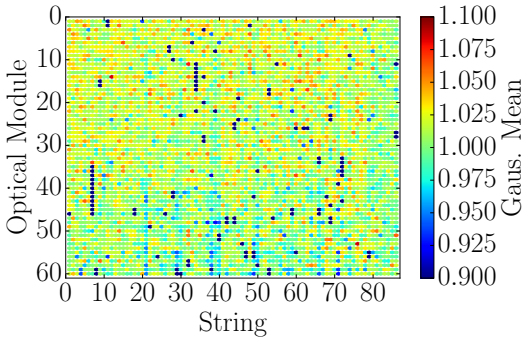

Optical Module
String
Exp1 Amplitude
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
6.4 5.6 4.8 4.0 3.2 2.4 1.6 0.8 0.0

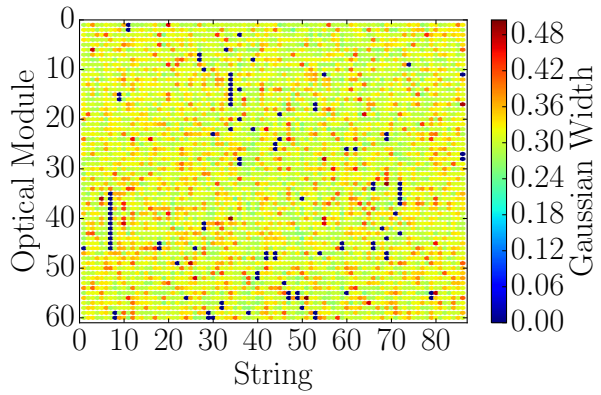

Optical Module
String
Exp1 Width
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
0.032 0.028 0.024 0.020 0.016 0.012 0.008 0.004 0.000

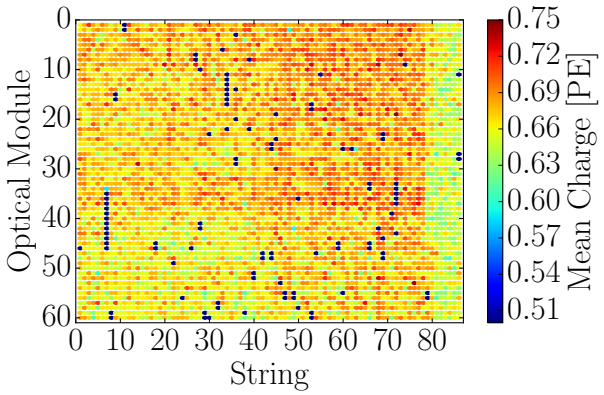

Optical Module
String
Exp2 Width
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
0.54 0.48 0.42 0.36 0.30 0.24 0.18 0.12 0.06 0.00

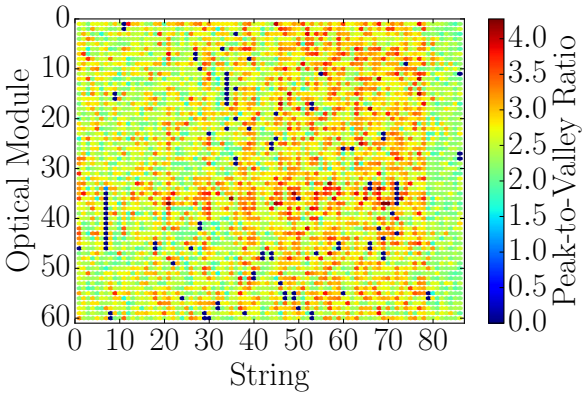

Optical Module
String
Exp2 Amp
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
0.64 0.56 0.48 0.40 0.32 0.24 0.16 0.08 0.00

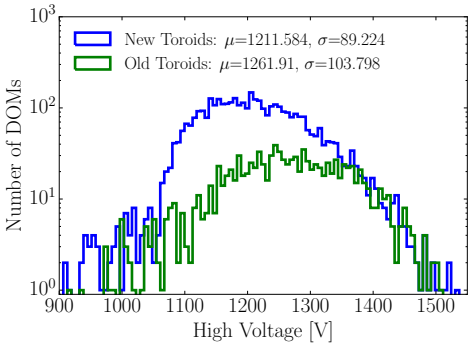

Optical Module
String
Gaussian Amp
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
0.9 0.8 0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0

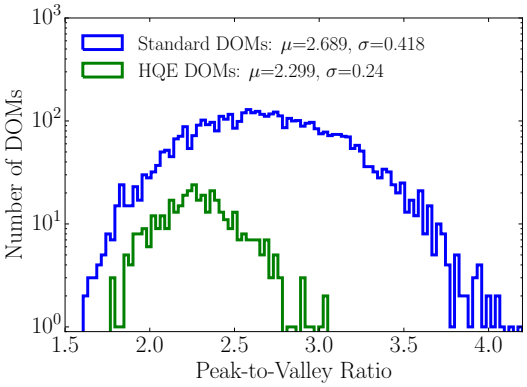

Optical Module
String
Gaus. Mean
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
1.100 1.075 1.050 1.025 1.000 0.975 0.950 0.925 0.900

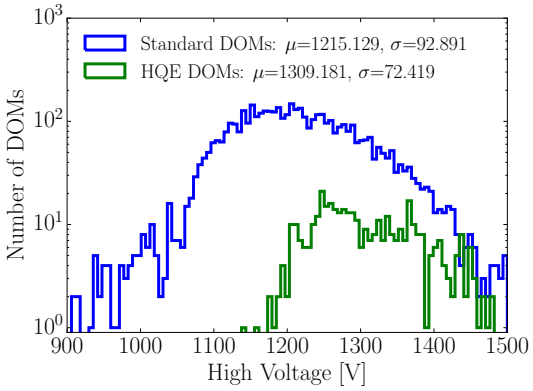

Optical Module
String
Gaussian Width
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
0.48 0.42 0.36 0.30 0.24 0.18 0.12 0.06 0.00

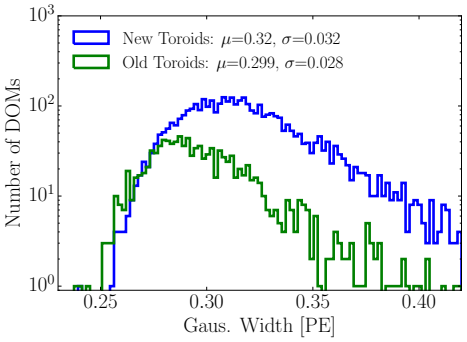

Optical Module
String
Mean Charge [PE]
0 10 20 30 40 50 60
0 10 20 30 40 50 60 70 80
0.75 0.72 0.69 0.66 0.63 0.60 0.57 0.54 0.51

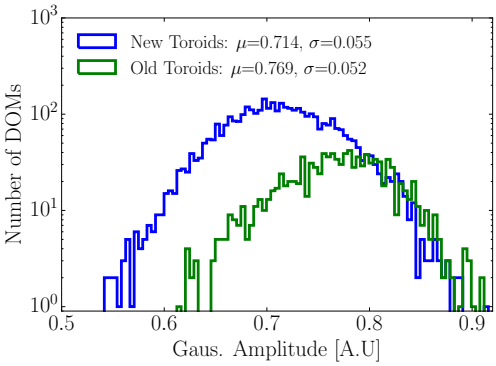

Optical Module
String
Peak-to-Valley Ratio

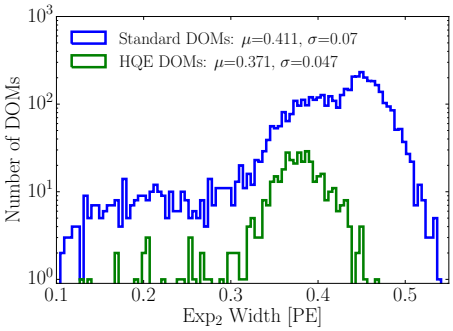

New Toroids: μ=1211.584, σ=89.224
Old Toroids: μ=1261.91, σ=103.798
Number of DOMs
High Voltage [V]

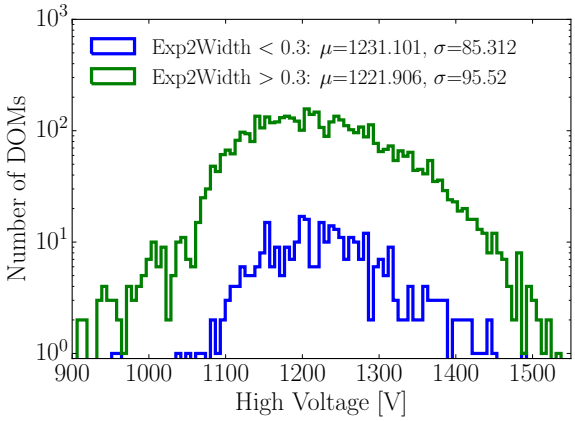

Standard DOMs: μ=2.689, σ=0.418
HQE DOMs: μ=2.299, σ=0.24
Number of DOMs
Peak-to-Valley Ratio

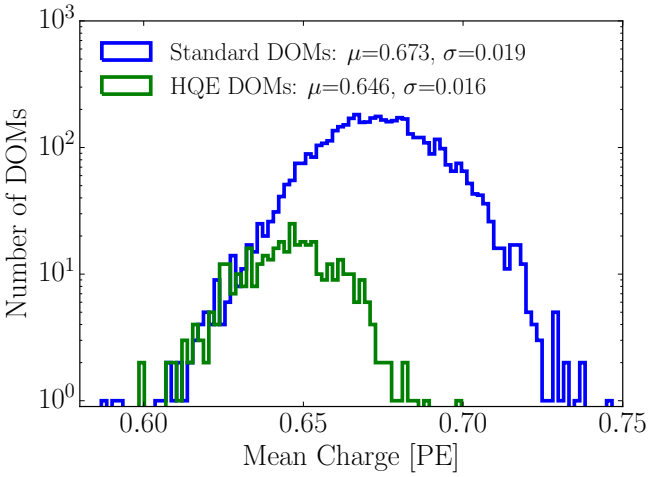

Standard DOMs: μ=1215.129, σ=92.891
HQE DOMs: μ=1309.181, σ=72.419
Number of DOMs
High Voltage [V]

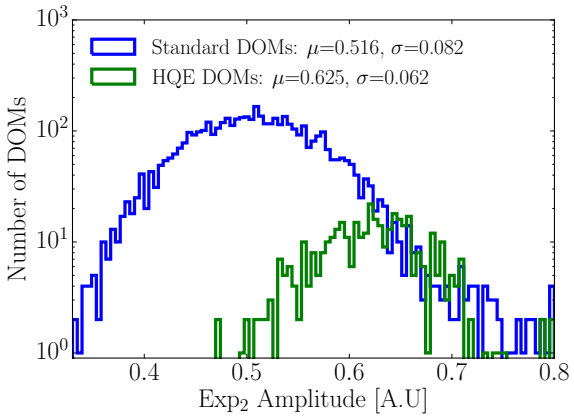

New Toroids: μ=0.32, σ=0.032
Old Toroids: μ=0.299, σ=0.028
Number of DOMs
Gaus. Width [PE]

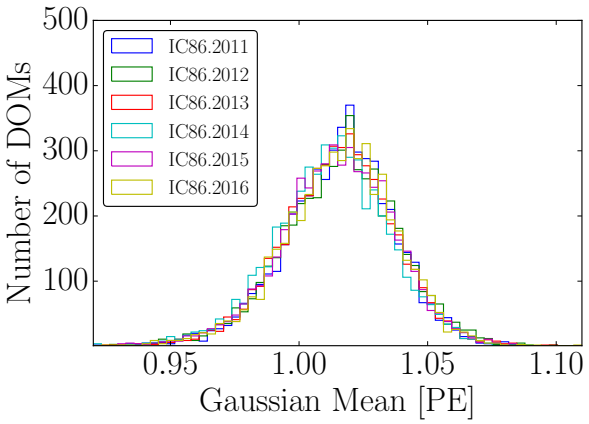

New Toroids: μ=0.714, σ=0.055
Old Toroids: μ=0.769, σ=0.052
Number of DOMs
Gaus. Amplitude [A.U]

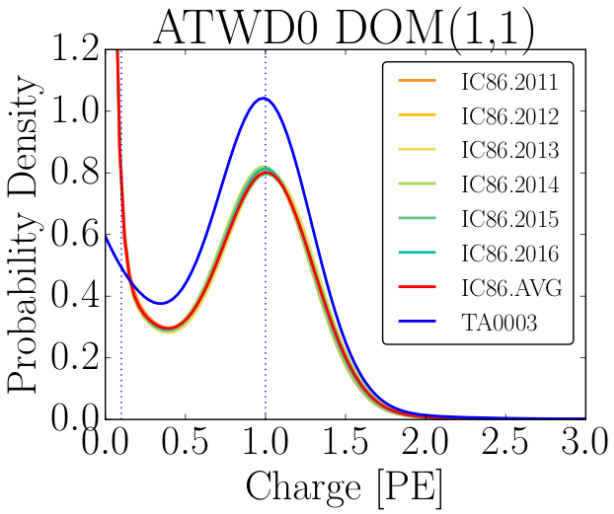

Standard DOMs: μ=0.411, σ=0.07
HQE DOMs: μ=0.371, σ=0.047
Number of DOMs
Exp2 Width [PE]

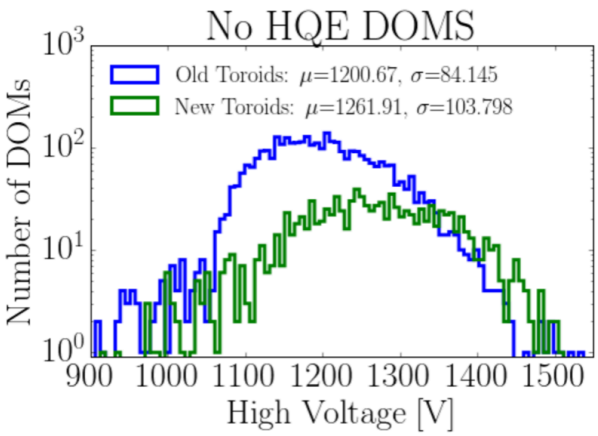

Exp2Width < 0.3: μ=1231.101, σ=85.312
Exp2Width > 0.3: μ=1221.906, σ=95.52
Number of DOMs
High Voltage [V]

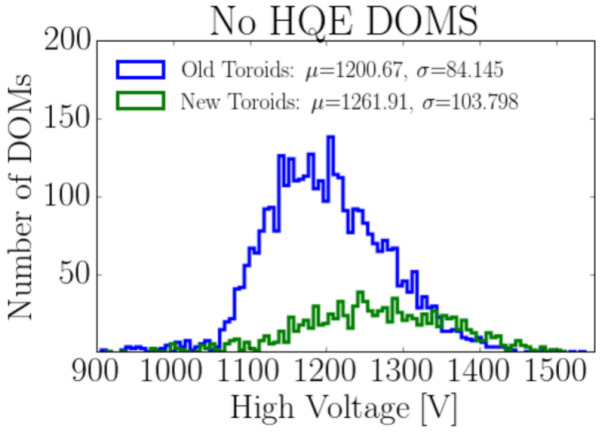

Standard DOMs: μ=0.673, σ=0.019
HQE DOMs: μ=0.646, σ=0.016
Number of DOMs
Mean Charge [PE]

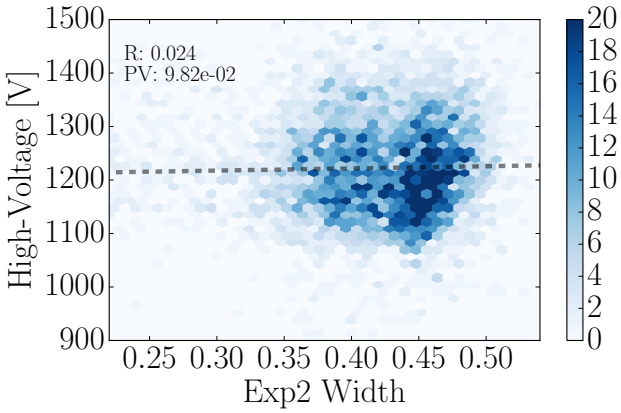

Standard DOMs: μ=0.516, σ=0.082
HQE DOMs: μ=0.625, σ=0.062
Number of DOMs
Exp$_2$ Amplitude [A.U]

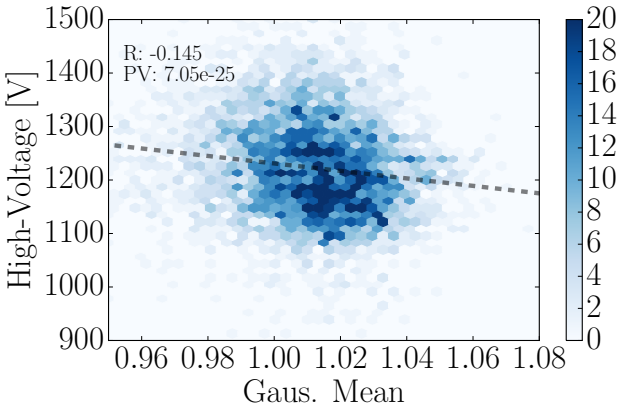

IC86.2011
IC86.2012
IC86.2013
IC86.2014
IC86.2015
IC86.2016
Number of DOMs
Gaussian Mean [PE]

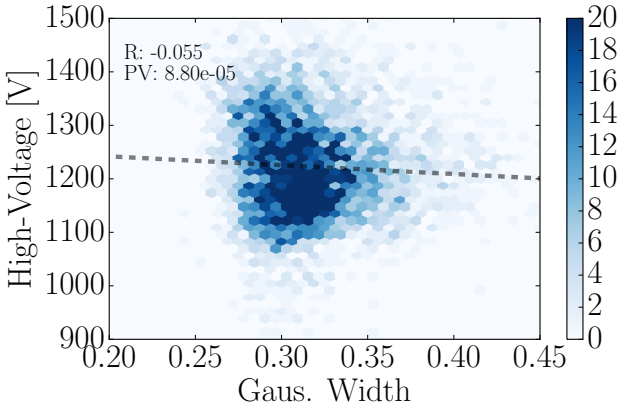

ATWD0 DOM(1,1)
IC86.2011
IC86.2012
IC86.2013
IC86.2014
IC86.2015
IC86.2016
IC86.AVG
TA0003
Probability Density
Charge [PE]

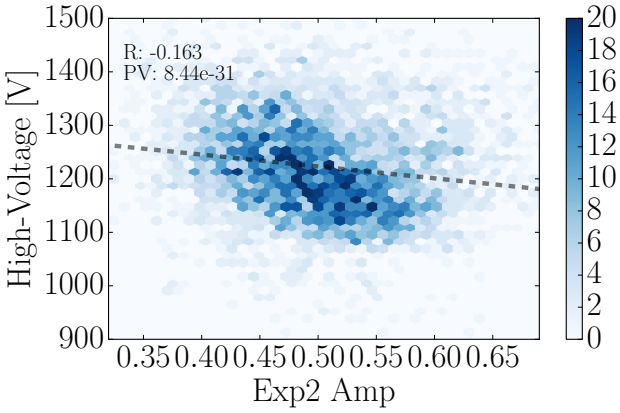

No HQE DOMS
Old Toroids: μ=1200.67, σ=84.145
New Toroids: μ=1261.91, σ=103.798
Number of DOMs
High Voltage [V]

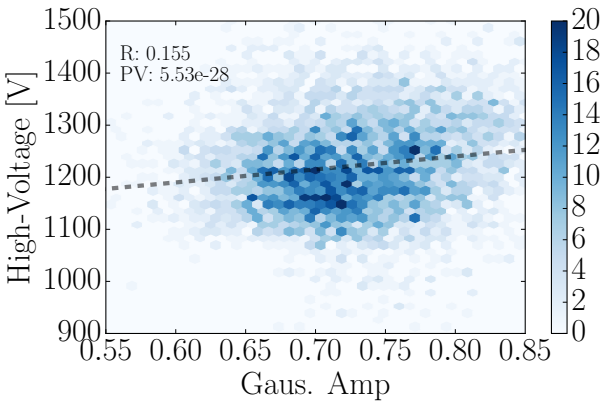

No HQE DOMS
Old Toroids: μ=1200.67, σ=84.145
New Toroids: μ=1261.91, σ=103.798
Number of DOMs
High Voltage [V]

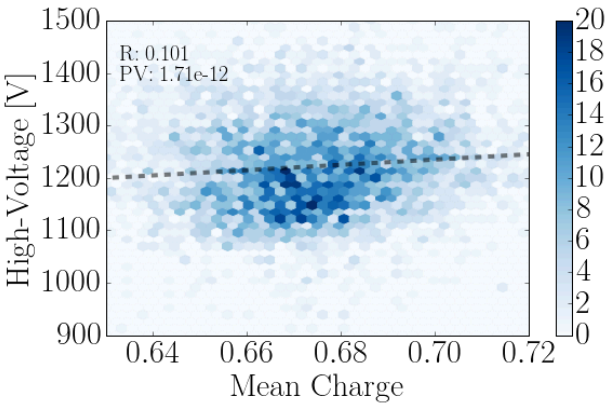

R: 0.024
PV: 9.82e-02
High-Voltage [V]
Exp2 Width


R: -0.145
PV: 7.05e-25
High-Voltage [V]
Gaus. Mean


R: -0.055
PV: 8.80e-05
High-Voltage [V]
Gaus. Width


R: -0.163
PV: 8.44e-31
High-Voltage [V]
Exp2 Amp


R: 0.155
PV: 5.53e-28
High-Voltage [V]
Gaus. Amp


R: 0.101
PV: 1.71e-12
High-Voltage [V]
Mean Charge