Signal bias, efficiency and resolution vs ideal track direction

Brett Viren

November 7, 2023

1 What

The **signal bias**, **efficiency** and **resolution** *vs* **track direction** are metric conventions introduced by the MB SP-1 paper. This document describes how to develop a procedure that generalizes, automates and standardizes their reproduction for MicroBooNE and extends their production to other LArTPC detectors. The procedure has the following stages:

1. **Generate** ideal line tracks at different angles in the detector volume. More info in 2.
2. **Simulate** noise and ADC waveform from the tracks.
3. **Translate** apply an optional ADC waveform transform. More info in 3.
4. **Filter** apply WCT noise filtering
5. **Signals** apply WCT signal processing
6. **Metrics** for each track, calculate and output the metrics. More info in 4.
7. **Plots** consume all metrics and make standardized plots.

This general procedure must be run in at least the following modes:

- Automatically by the WCT build system. See 5.
- Semi-automatically by analysts exploring new detectors or parameter space.
- Semi-automatically with UVCGAN inserted as a waveform transform. See 7.

Up to but excluding the final **plots** stage, the overall process execution model will be that of processing a single track through the entire pipeline. The single-track files that will link the stages are described more in section 8. This model will produce maximum per-job overhead but makes tracking per-track metadata simpler which is required for forming the metrics and making the plots. If this overhead becomes too limiting we can look to change this execution model.

2 Track Distribution

For any given run of the procedure, a subset of all possible ideal tracks must be defined by specifying parameters describing:

- track directions

- track locations
- total track length
- ionization yield

The following sections describe issues for each class of parameter specification. For each a "canonical" set of parameter values is defined and options are described.

2.1 Track directions

A direction is defined by two angles and the convention of figures 8, 29 and 30 of the MB SP-1 paper are adopted here. Each wire plane has its own Cartesian coordinate system with the Y-axis directed along the wire lengths and Z-axis directed along the wire pitch direction. The three share a common X-axis that runs counter to the nominal drift direction.

The MB paper identifies the coordinate system of the collection plane as special (eg, X, Y, Z) and either induction plane (eg X', Y', Z') as having a relative rotation. For brevity the two pertinent angles are written here as θ which measures the angle from the common Z-axis to the track direction in the $X - Z$ plane (collection coordinates) and θ' which measures the angle from the track to the Z'-axis in the $X' - Z'$ plane (coorinates of one induction plane). Note, this requires a projection of the track direction into that plane and is not simply the angle between track direction and Z'.

In specifying track angles and other geometry to this process the WCT geometry convention as documented here shall be followed. Specifically, the convention defines the Y ("plane W") and Z ("plane P") directions and origin.

Full enumeration of direction is 2D but the subset of directions selected by the axis in figure 30 of the MB SP-1 paper are taken as canonical.

2.2 Track locations

A track location is defined by the location of its midpoint. The MB paper specifies only the X location:

- $X = 1\,\mathrm{m}$ relative to the collection plane.

Given that wire (Y) and pitch (Z) coordinates are not described, they are defined here.

wires the (Y,Z) coordinate is chosen at the geometric center of the largest polygon produced by the wire crossing pattern (*eg* a hexagon in MB pattern).

strips the (Y,Z) coordinate is chosen at the center of a hole which itself is most centered on a collection strip.

These definitions select for the "longest drift path" and so is a maximally biased selection in some manner. However, this maximum bias is somewhat consistent for both the wires and strips+holes designs and so resulting plots may be comparable. See section 2.5 for entries into studying how this bias may effect the metrics.

2.3 Track length

The canonical track length is taken from the MB SP-1 paper as 1 m.

2.4 Ionization yield

The canonical ionization yield is taken from the MB SP-1 paper as 5333 electrons per mm.

2.5 Variations

For any of the above canonical parameter values the process must support an optional mode that allows user-specified values to be substituted.

2.6 Distributions

The parameters described are all discrete. If a sampling of a continuous distribution is required, the sampling may be performed externally and the results provided as optional parameter values.

3 Waveform Transform

The waveform translation step is expected to either be no-op or will consist of inserting UVCGAN to translate the ADC waveforms toward a different data domain. To support both generically, the ADC waveform output of the simulation can be written to file and read in for NF+SP. However, this adds processing time that is not required for the no-op case and for that case a unified WCT data flow graph covering sim+nf+sp should be constructed.

4 Metric Calculation

The canonical metrics are defined in section 4.3 of the MB SP-1 paper and the process must produce these. Developing novel or variant metrics is also encouraged. Each metric must be provided by the process in a data representation that follows metric-specific data schema. More on this in the section 8.

5 Historical Tests

A number of WCT BATS test files shall be constructed, each to run this processing on one supported detector type. The contents of these files be minimal and delegate to another command that is parameterized only on the detector type name (“pdsp”, “uboone”, etc). This command will be responsible for executing the process with canonical parameter choices. It may also implement options or it may delegate options to lower level programs/calls.

6 WCT Configuration

The reason for this specific factoring pattern is so that these tests, which will be modestly long-running, can execute in parallel by the WCT build system. The parameterization on only the detector type name may passed into a generic top-level Jsonnet file which uses the string to index an object with all possible detector configurations. Examples of this pattern exist in the `cfg/` area.

The historical tests will not support any translation (UVCGAN) step and thus may be configured with a single WCT DFP graph that spans all of sim+nf+sp. However, to facilitate configuration reuse for the UVCGAN translation variant, the configuration should support producing one of several sub graphs with optional file I/O based on a high-level parameter. The split should be allowed both before and after the NF stage.

7 Translations

Requirements for processing needed by the next LArTPC paper from the LS4GAN group align with this work with only minor additional effort.

- We must configure `wire-cell` to split the full graph as described in the previous section.
- This split may be between sim/nf or nf/sp depending on in LS4GAN group decides if NF is applied to the raw or filtered data tier.
- UVCGAN outputs Numpy format but in an array schema that differs from what WCT accepts and so a file conversion stage must be appended after the UVCGAN translation.

What other issues?

8 Tasks

8.1 WC job configuration

A set of `wire-cell` Jsonnet configuration files are needed for the sim+nf+sp parts and which satisfy the generality requirements given above. Some useful patterns can be found in existing cfg files:

- `img/test/tenio-*.jsonnet` implement a similar, generic pipeline parameterized by detector name.
- `sigproc/test/adc-nose-sig-bidir.jsonnet` (used for LS4GAN "toyzero" processing) gives examples how to have optional file "taps".
- `cfg/layers/mids.jsonnet` gives an example of how to make a configuration object parameterized by detector name.

In particular, it may be possible to use the `tenio*.jsonnet` files almost as-is. However, these are already subject to testing so can not be disturbed. Also, they are generated through tangling the file `img/test/test-tenio.org` which may require the developer to learn more arcane things than desired.

Note, the configuration need and should only consider a single `AnodePlane` (APA, CRU, etc).

8.2 Track depos

A new `wire-cell-python` CLI command is needed to generate track depos according to the canonical and optional modes described above. The command `wirecell-gen depo-lines` is close to what is required. To avoid overloading that one a new command should be started in the `wirecell-gen` CLI.

This new command must output two files for a single track:

depos A Numpy file in WCT "depo" format.

params A JSON file containing all canonical and optional parameter values

8.3 Metric calculator

A new `wire-cell-python` CLI command is needed to calculate the metrics. It may be provided by `wirecell-sigproc` and it shall consume a sequence of a pair of files for each track direction.

- The **params** file output described above.
- The **signals** Numpy "frame" file produced by the `wire-cell` job running SP.

This command shall produce at least one file:

metrics A JSON file containing the results of the calculated metrics

8.4 Metric plotter

A new `wire-cell-python` CLI command is needed to plot the metrics. It may be provided by `wirecell-sigproc` and it shall consume the **params** and **metrics** files for all track directions sampled. It will output a plot in either PNG or PDF similar to figure 30 of the MB SP-1 paper.

8.5 Automation

Because of the single-track-per-execute model, the level of sophistication for automating is low for the WCT tests use case. I expect the shell function/script that is called by the BATS file to simply loop over a hard-wired list of canonical track direction specifications and for each track call steps 1-6 in section 1. If/when all steps complete successfully this shell function/script would finish after calling step 7 to produce the plots.

For a non-test context it would be good but not critical to have something that can loop over a set of optional parameter values and do so in a parallel fashion. A suitably crafted Snakemake file could provide this functionality.