SVD timing

Peter Kvasnicka

Charles University, Prague

Belle II Tracking group F2F meeting,
18-19 September 2017, JGU Mainz

SVD timing

- SVD timing status
- To do (to have basic chain in place)
- To do (to make it work sensibly)

Schedule shift

- In July, SVD sw group decided to re-organize the software.
- Therefore, plans were changed and things got delayed.
- SVD timing is still not in basf2 master.

Outline

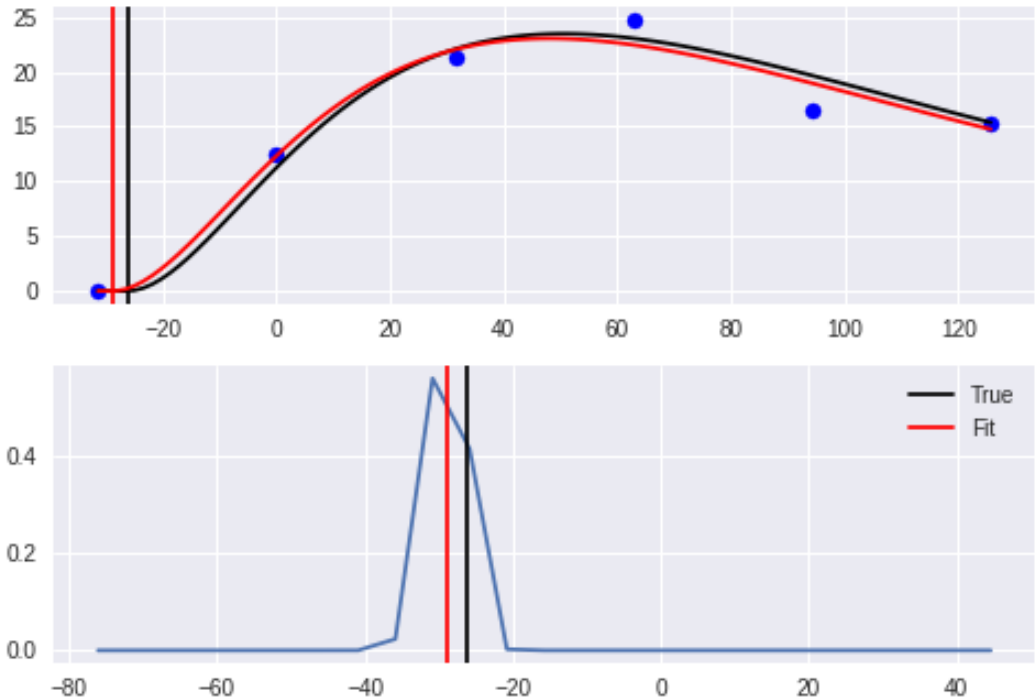
Status since Elba meeting

- No substantial changes
- Branch svd-digit-timer and pull request by end of June.
- Not merged due to SVD re-factoring plan

The task

- We fit a waveform function to 6 (or 3) strip signal samples, 31.44 ns apart, to estimate hit time.
- We use a neural network to perform the fit.
- The network outputs a probability distribution of arrival time.

Figure: Top: Measured APV25 signal samples (blue), true (black) and fitted (red) waveform. Bottom: Time shift probability distribution (blue) provided by the neural network, true and fitted shift times (black and red, resp.)



Parameter	Truth	Fit ± error
t	-26.3	-28.9 ± 2.7
A	23.46	23.03 ± 0.56
$\chi^2_n df$	4.12	
τ	307	

The network is a multiclass classifier:

Inputs Set of 6 strip signal samples plus waveform width parameter for the given strip

Outputs Probability distribution over a set of time bins of the actual hit time belonging to a given time bin.

Network setup

Layers Input layer of size 7 (as in 6 signals plus waveform width), two hidden layers and an output layer of size equal to number of bins

Activation Rectifier (reLU)

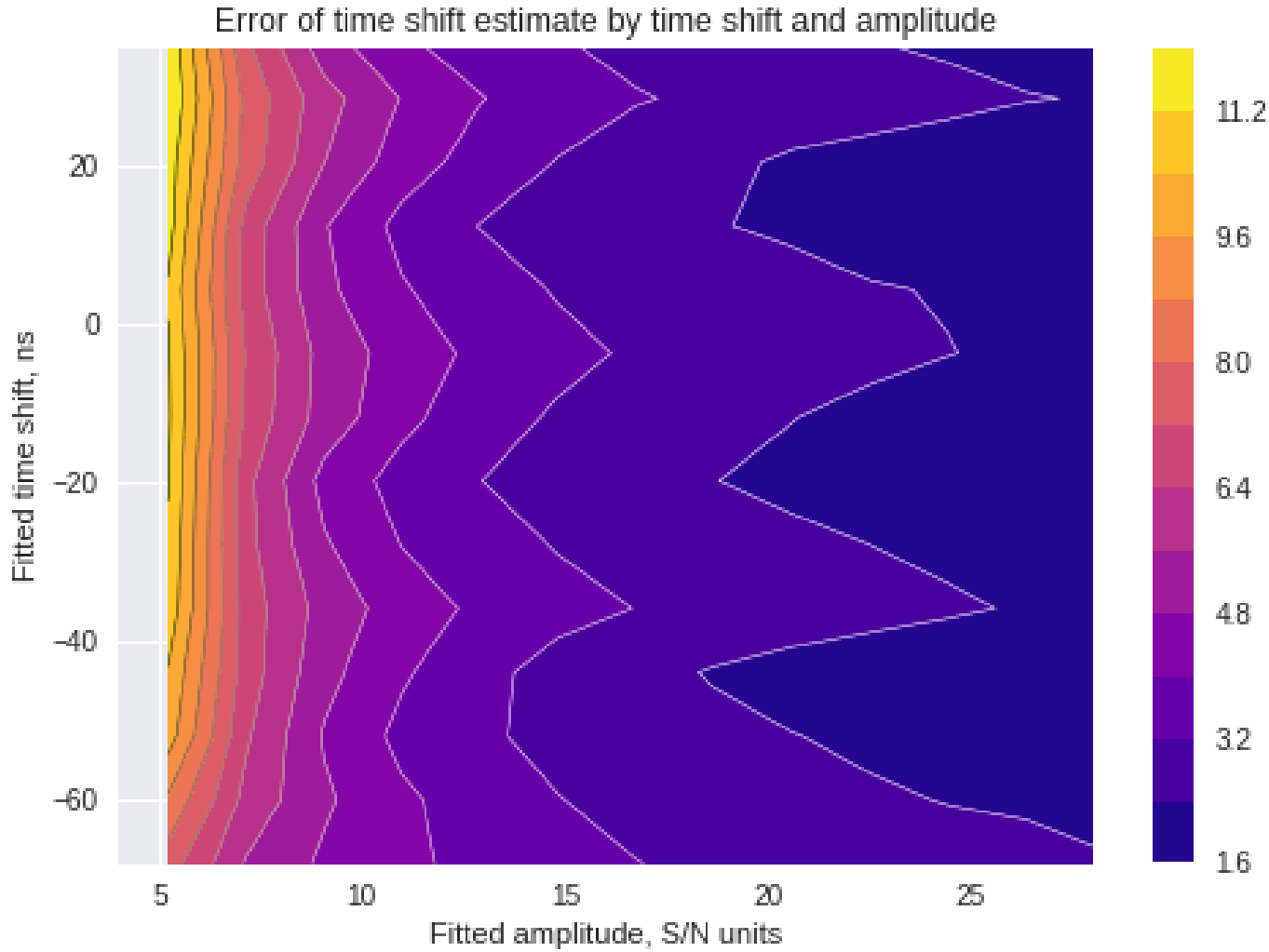
Input normalization Signals: divided by strip noise; width: normalized to signal range.

Output normalization Softmax (to get a probability distribution)

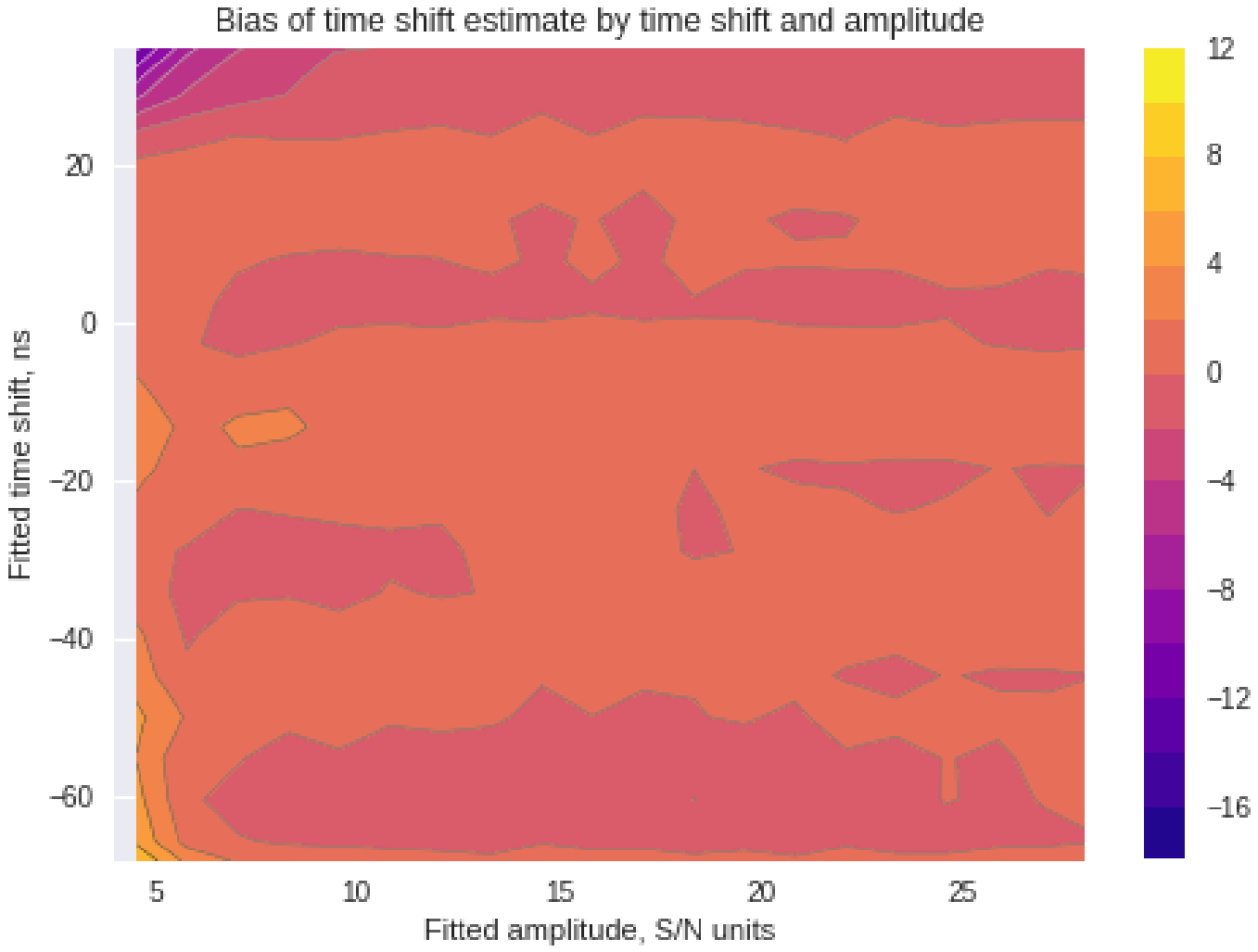
Implementation

- We have a pretty standard network, so any current software is good for the task.
- I used scikit-learn, which is included in basf2 externals
- The network is saved in an xml file or a corresponding DbStore object, which is used by the SVDClusterizer in basf2.
- The time needed for generation of a generous training/test samples and training the network is about 10 minutes on my laptop.
- I made my own implementation of network in C++ to compute network response (no call-back to python).
- The computation of output by the network is a matter of multiplying several matrices, so it is pretty fast.
- The scripts to generate data, train the network and produce some plots can be found in `svd/scripts` as Jupyter notebooks (on `svd-digit-timer` branch).

Neural network performance: Time error



Neural network performance: Time bias



Maintenance

- The network is trained on toy data using a waveform prototype derived from SVD hardware tests.
- The network is de-parameterized, meaning that it does not depend on run-dependent information except via inputs.
- Therefore, a single network works for all SVD sensor types and sides.
- The network basically does not need to be re-created except
  - when the signal waveforms change (due to ageing, for example).
  - when zero-suppression threshold is changed.

Network output

- The network learns from classification of waveforms into a set of time bins.
- Therefore, on the output, we have a probability distribution for the shift time on the set of time bins
- From this, we calculate estimate of time shift and its error and amplitude and its error.
- The probabilities would be an overkill if we only wanted the above.
- The probabilities help us measure time compatibility of signals.

Let us repeat how the SVD timing works:

- SVD reads out a signal sample from each strip each 31.44 ns.
- When a trigger arrives, SVD stores the last recorded sample plus 5 consecutive samples from each strip.
- Thus we know the trigger arrived, by SVD clock, in or near the time interval (-31.44 ns, 0 ns), but as the SVD clock is independent, we cannot sync it with Belle II DAQ time, at least not easily.
- We have to estimate the trigger time, looking fro a bunch of hits in or near the (-31.44 ns, 0 ns) time interval.

Cluster time

- In a cluster, we have sextets of several strips.
- Cluster time is naturally calculated from the product of contributing probability distributions.
- During clustering, we can test each strip for compatibility with the current cluster candidate.
- We can use the cluster timing distribution to estimate compatibility of u/v cluster pairings.

Background estimation

- If we can estimate trigger time, we can not only suppress background, we can estimate it
- Therefore, we can use the whole SVD for background monitoring.

SVD timing parameters

- We need a lot of parameters to get usable SVD timing information for a particular sensor/strip.
- These parameters include:

  Strip noises Used to convert signals to S/N values, to set thresholds etc.

  Strip flags Used to ignore or specially process signals from faulty/non-typical strips.

  Time shift This is a strip-dependent time shift, accounted for by shifting the time-shift probability distribution.

  Waveform width Strip-dependent waveform width is used as input to the neural network.

  Strip gain is used to correct signal values.
- All these parameters are read from SVD hardware test results files, and are updated on a run-by-run basis.

SVD hardware test calibration: status

- A suable calibration is currently only implemented in the DESY 2017 testbeam branch (no database support).
- Users can select the appropriate hardware calibration xml by setting a SVDClusterizer module parameter in their steering file.
- This has limited use, since the testbeam branch can no longer be reasonably supported.
- There is a new version of the SVD calibration framework in construction by Laura and Eugenio, with complete database support, but the implementation is not yet complete.

Re-factoring of SVD software - on critical path

New data objects : two kinds of SVDDigit

- The SVDShaper digit is the "proper" SVD digit, containing measurements from the unpacker or digitizer output (implemented completely as the first batch of SVD sw refactoring)
- The SVDRecoDigit is a calibrated and time-fitted digit (temporary object, under construction).

Digit calibration module is a separate module that applies calibration and time fit to digits, producing SVDRecoDigits (under construction)

SVDClusterizer will only do clustering of SVDRecoDigits (under construction)

Calibration service retrieves strip calibration data from the database and makes them available to modules (Eugenio and Laura, under construction).

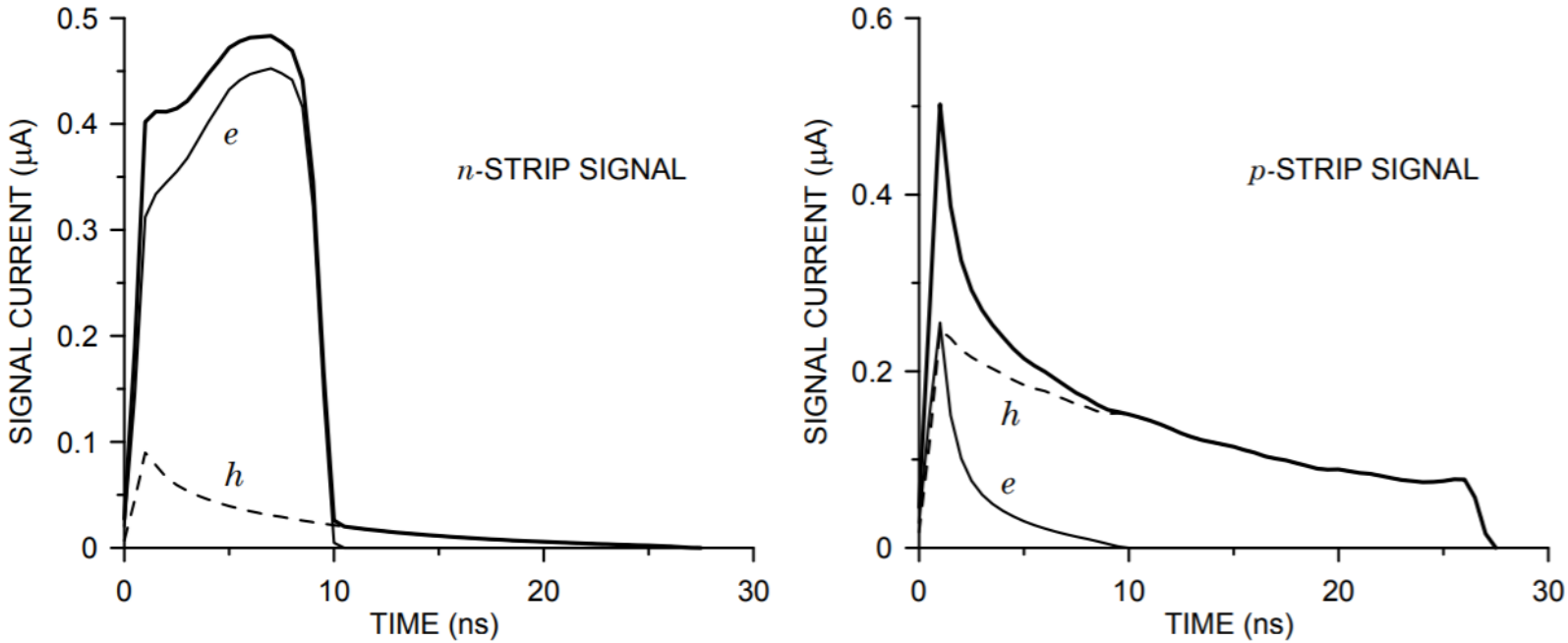
I will make a pull request for the digit calibrator and clusterizer in a few days (it is only re-shaping existing code from the svd-digit-timer branch). To be fully usable, it will need the calibration code.

Out-of-critical path work

- Switch network implementation to mva package
  - +: General interface, it will be possible to plug-in different fitting methods (random forests, SVM etc.)
  - +: Database support is part of the package
  - -: I have to check how scaling will work
  - -: Callback to python may be slower than current C++ implementation, may have to use *lwtnn* for performance.
- SVDDigitizer re-factoring:
  - Original plan was to do this in summer, I need the whole SVD chain to work for that.
  - SVD digitizer is very slow
  - We have the timing simulation wrong (signal at collection time vs. signal during charge movement)
  - This is on the mental critical path

Signal vs. time in a Si sensor

Figure: Current in p- and n-strips of a double-sided Si sensor [H. Spieler, Refresher course on Si Detectors]. For Belle II SVD the saturation times are 9 ns (e-) and 24 ns (h). The current digitizer places a delta-pulse at the surface arrival point for each chargelet.



Conclusions

- The SVD timing estimation is still not implemented due to SVD re-factoring activities.
- Most of the stuff will be implemented within 10 days.
- The overall functionality depends on the availability of calibration data.
- SVD digitizer urgently needs re-factoring, as it is slow and does signal time-dependence wrong.