EUDET-Memo-2009-16


# Update on the Status of MarlinTPC

Jason Abernathy[*], Patrick Conley[*], Klaus Dehmelt[†], Ralf Diener[†], Jan Engels[†], Keisuke Fujii[‡], Jim Hunt[§], Martin Killenberg[¶], Frederik Klöckner[¶], Thorsten Krautscheid[¶], Astrid Münnich[‖], Christoph Rosemann[†], Oliver Schäfer[**], Martin Ummenhofer[¶], Adrian Vogel[†], Peter Wienemann[¶], Simone Zimmermann[¶]

December 8, 2009

**Abstract**

MarlinTPC as the TPC R&D package of the ILCSoft software collection provides TPC specific Marlin processors for simulation, digitisation reconstruction and analysis.[1] MarlinTPC has been extended to handle multiple readout modules on a TPC end plate and tools for test beam data have been developed. We report on the new functionality and improvements that have been made since last year's EUDET report[2].

[*]University of Victoria, Victoria, BC, Canada
[†]DESY, Hamburg, Germany
[‡]KEK, Tsukuba, Japan
[§]Cornell University, Ithaca, NY, USA
[¶]University of Bonn, Bonn, Germany
[‖]CERN, Geneva, Switzerland
[**]University of Rostock, Rostock, Germany

# 1 Introduction

In the past year many test beam campaigns have been performed, especially with the LC-TPC Large Prototype currently installed at DESY Hamburg. The end plate consists of several readout modules, like in a TPC at a real collider experiment. To be able to reconstruct the data taken with this detector MarlinTPC had to be extended for multiple modules and to fit curved tracks taken in magnetic field.

Furthermore, MarlinTPC had only been used for Monte Carlo data so far. The real data now available allowed to debug and improve the existing code.

# 2 Making MarlinTPC Multi Module Capable

The main work that has been put into MarlinTPC during the last year was the extension to multiple modules. It is based on the modular TPC description of the Gear package[3]. These changes affected most of the processors and some conditions data classes.

To be able to distinguish different modules the `CellID1` variable of LCIO[4] is used. While `CellID0` was used for the channel/pad ID, `CellID1` had not been used up to now. It explicitly has to be turned on in all processors, otherwise LCIO does not store this variable.

For those processors using Gear information the code had to be changed from using a single pad plane to using modules. Although all of the main reconstruction processors can handle multiple modules now, this does not mean that the full advantage of multiple modules is used yet. The topological hit and track finder for instance only reconstructs track segments on individual modules, but is not able to perform track finding on the whole end plate.

All conditions data classes containing the channel or pad number had do be extended by the module ID. As a consequence also the handler classes, which provide access to the conditions data, had to be adapted, as well as all processors using the conditions data.

Although the extension was rather straight forward it was very time consuming due to the large number of processors and the implicit dependencies on the conditions data classes. The status now is that all processors run with the multi module Gear version and are able to process different module IDs. The only exception is the simulation/digitisation, which still uses a single pad plane. However, the data written contains `CellID1`, which is always set to 0, so it is consistent with the following reconstruction and analysis processors.

# 3 Running With Real Data

Up to the beginning of this year the pad based reconstruction with MarlinTPC had only been tested with Monte Carlo data. Most of these were simple toy Monte Carlos without pedestals and noise. Using MarlinTPC with real data showed that some processors had to be improved.

**Pedestals** The pedestal calculator was one of the first processors in MarlinTPC and had actually never been used. Several bugs prevented the code from running. In order to write LCCD[5] to either a simple LCIO file, a data base LCIO file or a real MySQL data base, classes called `DBEntryMaker` and `DBWriter` had been introduced. In fact, only writing to simple files worked.

The functionality that `DBEntryMaker` and `DBWriter` provided has become obsolete with further developments in LCCD. So these classes have been removed and LCCD is used directly. The bug-fixed code now provides the full functionality and is able to calculate pedestals not only from special pedestals runs but also from files with signal data. An iterative algorithm selects only those regions without signals.

During pedestal subtraction it turned out that the out "of range" information of an ADC in overflow got lost. This has also been fixed by introducing a maximum ADC value. All samples with this value are not corrected, so the information that the ADC reached the limits of its dynamic range persists.

**Pulse Finder** The pulse finder had been taken from the MultiFit[6] code. It was optimised for long pulses like they were produced by the ALEPH electronics that had been used for the data taking with the DESY MediTPC[7], for which MultiFit had been developed. It turned out that some assumptions about the pulse shape were not general enough and prevented data taken at the Large Prototype from being reconstructed. In addition there were some logical bugs in the MarlinTPC implementation, which was a show stopper for the whole reconstruction.

A major rewrite now provides a versatile pulse finder which allows pulse splitting (optional) and the calculation of the pulse time either as the inflection point of the rising edge (for slow signals) or as the centre of gravity (fast signals). It has been tested with data from the old ALEPH electronics as well as with ALTRO[8] data and different Monte Carlos.

**Track Fitting** Up to now only measurements without magnetic field from comparatively small test chambers had been reconstructed. For these data straight line fits were sufficient.

The Likelihood fit implemented in the `TrackFitterLikelihoodProcessor`, which is able to fit curved tracks in a magnetic field, turned out to require a very good calibration of the gas gain to converge numerically. This calibration usually is not available for test beam measurements. In addition the algorithm runs on pads, so it cannot be used for measurements with the Timepix (see section 4).

Thus a numerically robust fit for curved tracks was needed. It has been implemented in the `TrackFitterSimpleChiSquare` fitter. This fitter minimises the residuals measured perpendicular to the track, but does not use individual errors for the different hits. For the expected width of the residual distribution ($\sigma$) it uses one value for the initial detector resolution and one value which scales with the square root of the drift distance (diffusion term)

$$\sigma^2_{\text{trans}} = \sigma^2_{0,\text{trans}} + \sigma^2_{\text{diff},\text{trans}} \cdot z$$

$$\sigma^2_{\text{long}} = \sigma^2_{0,\text{long}} + \sigma^2_{\text{diff,long}} \cdot z$$

This fitter, which was the last missing part in the reconstruction chain, is now available and the reconstruction chain is complete. It is implemented according to the track fitter scheme described in section 6.

For a final analysis this simple fit is not sufficient. A $\chi^2$-fitter which implements the covariance matrix is planned to be implemented, as well as Kalman filter, which is currently being worked on.

# 4 Reconstruction of Timepix Quadboards

The Timepix is an readout ASIC with 256 × 256 pixels per chip and a pixel size of 55 × 55 µm$^2$.[9] The Quadboards[10] produced by NIKHEF have a matrix of 2 × 2 Timepix chips. However, the Pixelman software[11] used to read out the board returns raw data which looks like a 512 × 512 matrix. In first approximation this is what the four chips look like, but it neglects the small gaps between the chips and possible rotations.

For a proper reconstruction the 512 × 512 matrix is separated into four 256 × 256 matrices, which is done by the newly introduced `TimePixMultiChipBoardDividerProcessor`. Afterwards the standard reconstruction chain for individual chips is run (see [2]). It is multi chip capable by treating each chip as a readout module.

In the last Timepix-specific step, the calculation for the hit centres, the position of the individual chips has to be known. Within Gear each Timepix is described as a module with a rectangular "pad plane" with a pad size of 55 × 55 µm$^2$. The alignment of the chips, which can be obtained by measuring with a movable table and a microscope, can be coded into the Gear XML file. Like this a manual alignment can be performed. However, applying module alignment from conditions data is a feature which will be required in future, not only for the Timepix but also for the Large Prototype TPC.

# 5 Reconstruction of Photo Dots

The cathode plane of the Large Prototype TPC is equipped with a pattern of aluminium dots on a copper plate (see figure 1). Pulsing light from a UV laser into the TPC allows to release electrons only from the aluminium dots. If the wave length of the laser is tuned such that the photon energy is lower than the work function of the copper but higher than that of aluminium, a pattern of free electrons in the TPC will become visible.

The `PhotodotReconstructionProcessor` knows the exact positions of the pattern on the cathode. Reconstructing the measured dots and comparing with the real position allows studies of the field homogeneities. Measuring with and without magnetic field one can separate the electric and the magnetic inhomogeneities.

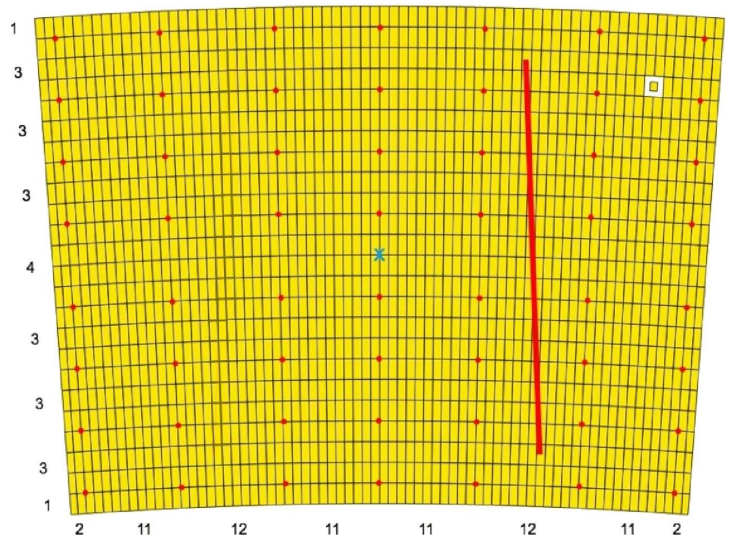


Figure 1: The pattern on the Large Prototype cathode. The red dots are aligned with the pads of the Micromegas modules such that their nominal position is in the edge of four pads.

# 6 The Track Fitter Scheme

The calculation of residuals of the hits wrt. the track depends on the definition (distance perpendicular to the track, distance along the pad row, distance along a coordinate axis) and on the fitting method used ($\chi^2$-fit, Likelihood-fit). The filling of histograms once the residuals are calculated, however, is the same for all cases. Writing a histogramming processor for each method to calculate residuals would lead to a lot of code duplication.

This is especially the case for the proposed "geometric mean" resolution, where the geometric mean of the residual to a track fitted with and without the specific hit is calculated. For this calculation the track has to be refitted, which would duplicate the complete fitting code if the resolution processor does not have access to the fitting algorithm.

To solve this problem a `TrackFitterBase` class has been implemented, its interface definition is shown in appendix 9. The main components are the `calculateResiduals()` function and the `fitTrack()` function. `calculateResiduals()` returns the residuals in the $xy$-plane and in $z$-direction as a vector of doubles for one individual hit. This may be for a hit on the track, or the residuals to a reference track. The `fitTrack()` function performs the fit.

The initial values of the track are taken from a seed track by default. Each individual initial parameter can be overwritten manually. In addition each parameter can be fixed during the fit (for instance keep the curvature 0 to fit a straight line).

The individual fitters are derived from the `TrackFitterBase` class, inheriting its interface. The fitter processors as well as the analysis processors use instances of the fitter implementations. The fitter processors are specific for the individual fitters, while the analysis processors are fitter type independent.

| Data Class | Description |
|---|---|
| `ADCChannelMapping` | Mapping HW Channel ↔ Gear PadIndex / ModuleID |
| `ChannelCorrection` | Channel quality flags and calibration factors |
| `FieldSetting` | Fields and voltages in drift volume and readout structure |
| `GasConditions` | Gas mixture, pressure, temperature |
| `Pedestal` | Pedestals on individual channels |
| `TimePixPixelMode` | Running conditions of Timepix (Time or Charge mode) |
| `TPCConditions` | Calibrated drift velocity, diffusion coefficients ... |
| `WeatherConditions` | Environmental temperature, pressure ... |

Table 1: The conditions data objects defined in TPCCondData.

Each fitter implementation has a unique ID, which is stored with the track. The analysis processor which wants to calculate the residuals or refit the track retrieves the fitter which had been used to fit the track from the `TrackFitterFactory`, using the ID from the track. Like this the analysis code is independent from the fitter implementation.

# 7 Conditions Data

To reconstruct data from test beam measurements many run conditions have to be taken into account. To store these data MarlinTPC comes with a collection of data classes called `TPCCondData`. Table 1 shows the classes currently defined and a descriptions of their functionality. The experience from the test beams shows that some functionality is still missing. This is why currently all data classes are revisited and new classes are being implemented (for instance to describe the readout electronics).

To store and access these data a data base server is currently being set up at DESY, Hamburg. To fill and query the data base LCCD will be used. This allows to access the data which is valid for the event currently being reconstructed and is fully integrated in Marlin, which makes it easy to use in the processors.

# 8 Other Tools and Improvements

**Pad Mapping** A stand-alone programme called `MappingGenerator` allows to generate the required conditions data (`ADCChannelMapping`) from a simple text file (see listing 1 for the syntax) and the Gear XML description of the pad planes. The pad is specified by the row number and the pad number within the row. The (possibly pad plane implementation dependent) Gear pad index, which is needed in the conditions data class, is generated by the tool using Gear.

**Processor Status and Code Cleanup** With more than 50 processors and more than 3 years of development MarlinTPC has come to a point where some processors need to

```
# HWGroup HWChannel GearModule PadRow PadInRow
    0         0           1         0       0
    0         1           1         0       1
    0         2           1         0       2
    0         3           1         0       3
    0         4           1         1       0
    0         5           1         1       1
    0         6           1         1       2
    0         7           1         1       3
```

Listing 1: Syntax example of the pad mapping text file.

be updated, some have not been tested with real data yet. As the functionality of other components in ILCSoft has increased some functionality has even become obsolete.

In order to get an overview which processor is in which state a Wiki site has been installed (http://www-flc.desy.de/flc/flcwiki/MarlinTPCProcessorList). Currently all processors are being revisited and its status noted on this page. Along with this review of functionality the code is being cleaned up (e. g. removal of unnecessary includes) and an automatic beautifier is run to improve the readability of the source code.

**Monitoring Tools** For data taking it is essential that the data acquisition can be monitored. For this reason new monitoring processors have been introduced:

| | |
|---|---|
| `PerPadPulseChargeHistogrammerProcessor` | Creates a charge spectrum histogram for each channel. |
| `PulseCounterProcessor` | Histograms the number of pulses per event for each channel. |
| `DumpHitsProcessor` | Dumps the $x$, $y$ and $z$ coordinate of the reconstructed hits to the console. This can be used for debugging or visualised with a plot programme. |
| `ADCRawEventDisplayProcessor` | Creates an "ADC counts vs. time sample" histogram which serves as a raw data display for the pulses on each channel. |

# 9 Conclusion

In the past year MarlinTPC has shown that it can be used to reconstruct test beam data. It has been extended to handle multiple readout modules, tested and debugged with real data and tools needed to analyse the test beam data have been implemented.

TPC R&D groups from Canada, China, France, Germany, Japan and Sweden started to analyse data with it.

The consistent usage of conditions data is currently being prepared. The necessary step of alignment of the modules has to be addressed. This will lead to an extension of Gear, resulting in a combined geometry and conditions data package which will deliver not only static geometry information, but also values changing at run time.

Fitting algorithms which give better results than the simple $\chi^2$ fit are currently being implemented.

## Acknowledgement

This work is supported by the Commission of the European Communities under the $6^{th}$ Framework Programme "Structuring the European Research Area", contract number RII3-026126.

# Appendix: Interface of the TrackFitterBase Class

## Public Member Functions

| | |
|---|---|
| | **TrackFitterBase** ()<br>*The Constructor.* |
| virtual | **~TrackFitterBase** () |
| virtual std::string | **getRevision** ()<br>*Get the revision of the actual fitter implementation.* |
| virtual unsigned char | **getFitterType** () const =0<br>*Information what fitter implementation this is.* |
| virtual IMPL::TrackImpl * | **fitTrack** (EVENT::Track const *seedTrack)=0<br>*The function which does the actual fitting.* |
| virtual EVENT::DoubleVec | **calculateResiduals** (EVENT::Track *testTrack, unsigned int testHitNumber, EVENT::Track *referenceTrack=NULL)<br>*Return the distance of the hit `testHitNumber` of the testTrack to the point of closest approach on the referenceTrack wrt. the hit.* |
| virtual EVENT::DoubleVec | **calculateResiduals** (EVENT::TrackerHit *testHit, EVENT::Track *trackWithoutTestHit)<br>*Return the distance of the given hit to the test track where this hit has been removed.* |
| void | **setD0Start** (double d0)<br>*Set a d0 start value for the fit.* |
| void | **setPhiStart** (double phi)<br>*Set a phi start value for the fit.* |
| void | **setOmegaStart** (double omega)<br>*Set a omega start value for the fit.* |
| void | **setTanLambdaStart** (double tanLambda)<br>*Set a tanLambda start value for the fit.* |
| void | **setZ0Start** (double z0)<br>*Set a z0 start value for the fit.* |
| void | **setNoZFitFlag** (bool noZFit=true)<br>*Switch to turn off the fitting in z.* |
| bool | **getNoZFitFlag** ()<br>*Get the noZFit flag.* |
| void | **fixD0** (bool fix=true)<br>*Fix or release d0.* |
| void | **fixPhi** (bool fix=true)<br>*Fix or release phi.* |
| void | **fixOmega** (bool fix=true)<br>*Fix or release omega.* |
| void | **fixTanLambda** (bool fix=true)<br>*Fix or release tanLambda.* |
| void | **fixZ0** (bool fix=true)<br>*Fix or release z0.* |

# References


[1] J. Abernathy et. el., "MarlinTPC: A Marlin based common TPC software framework for the LC-TPC collaboration" LC-TOOL-2007-001, 2007

[2] J. Abernathy et. el., "Latest developments in the MarlinTPC software package", EUDET-Report-2008-09, 2008

[3] M. Killenberg and S. Turnbull, "A Modular TPC Endplate Description for Gear", EUDET-Memo-2008-31, 2008

[4] F. Gaede, T. Behnke, N. Graf, T. Johnson, "LCIO - A persistency framework for linear collider simulation studies", CHEP03, arXiv:physics/0306114, 2003

[5] LCCD: The Linear Collider Conditions Data toolkit, http://ilcsoft.desy.de/portal/software_packages/lccd/

[6] Multifit, http://www-flc.desy.de/tpc/projects/multifit/

[7] The DESY MediTPC, http://www-flc.desy.de/tpc/projects/medi_tpc/

[8] L. Musa et. al., "The ALICE TPC Front End Electronics", Proc. of the IEEE Nuclear Science Symposium, 20 - 25 Oct 2003, Portland

[9] X. Llopart, et al., "Timepix, a 65k programmable pixel readout chip for arrival time, energy and/or photon counting measurements", Nucl. Instrum. Meth. A **581**:485-494, 2007

[10] Medipix @ NIKHEF Homepage, http://www.nikhef.nl/pub/experiments/medipix/

[11] T. Holy, J. Jakubek, S. Pospisil, J. Uher, D. Vavrik and Z. Vykydal, "Data acquisition and processing software package for Medipix2", Nucl. Instrum. Meth. A **563**:254-258, 2006, doi:10.1016/j.nima.2006.01.122

[12] The MarlinTPC wiki page: https://twiki.cern.ch/twiki/bin/view/ILCTPC/MarlinTPC