Generic VELO Pattern Recognition

Internal Note

Issue: 1
Revision: 0

Reference: LHCb-002-2007
Created: January 20, 2007
Last modified: January 29, 2008

Prepared by: Tomáš Laštovička [a]
[a]CERN,Switzerland

Abstract

The VELO detector is designed for fast and accurate reconstruction of tracks pointing to the nominal interaction region and assuming VELO being in its closed position. Such tracks are linear in $r - z$ projection and about constant in ϕ coordinate. Nevertheless, for various VELO applications this linearity is broken. The Generic Pattern Recognition aims at the reconstruction of such generic tracks, including the case of test-beam events and VELO being in its open position.

Document Status Sheet

1. Document Title: Generic VELO Pattern Recognition			
2. Document Reference Number: LHCb-002-2007			
3. Issue	**4. Revision**	**5. Date**	**6. Reason for change**
Draft	1	February 6, 2007	First version. Pictures missing.
Draft	2	May 5, 2007	Final draft ready.

Contents

List of Figures

List of Tables

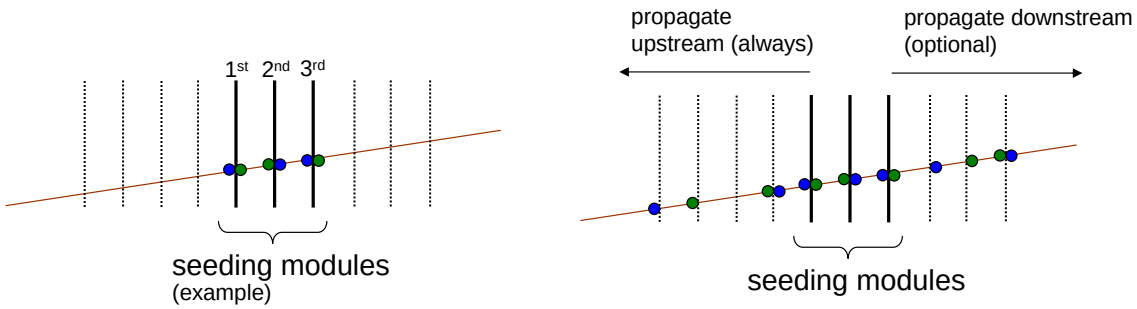

Figure 1 The track seeding performed by minimising triplet sagitta (left) and the propagation of track seeds in upstream and downstream directions. Closed circles mark R and ϕ clusters assigned to the track (line).

1 Introduction

The default VELO pattern recognition employs $R-\phi$ design of VELO modules thus providing fast and efficient pattern recognition in two steps - first R and then ϕ sensors. The Generic Pattern Recognition acts as a complementary pattern recognition in the sense that it does full 3D pattern recognition at once and includes a couple of features designed for specific tasks[a]. Nevertheless, there is a speed penalty due to potentially large combinatorics following from not splitting the pattern recognition into two two-dimensional steps.

It is worth to mention the Generic Pattern recognition is not using space-points but rather it is fitting R and ϕ clusters directly in three dimensions. Thus the distance in z between R and ϕ sensor is correctly accounted for and does not need to be small or neglected. In principle, R and ϕ clusters do not even need to be paired.

2 The Pattern Recognition Algorithm

The way the track seeds are found and propagated through the VELO detector is illustrated in Figure 1. For simplicity only a number of modules in one VELO half is shown as vertical lines. The algorithm starts from the downstream part of VELO and continues to the upstream end.

1. *Seeding:* A triplet of neighbouring modules is used to find track seeds, i.e. a combinations of clusters which are already sufficient to define tracks. The seeding of tracks is based on the three dimensional triplet sagitta defined as a distance of a cluster on module 2 from a line connecting 2 $R-\phi$ pairs on modules 1 and 3. [b] The two pairs are found by looping over all combinations of R and ϕ clusters within one R-sector (i.e. approximately 45° wedge). The clusters are submitted to the fitter, see Section 2.1, and fitted by a straight line in 3D. If the line is not matching acceptance of a cluster the cluster combination is ignored for the track seeding. If there are two clusters on module 2 close enough to the fitted line they are added to the fitter and the line, now called the track seed, is re-fitted.

   If needed, a triplet of R-sectors with a total number of clusters exceeding *ClusterCut* (see Section 3) is ignored for track seeding purposes. The motivation is to avoid too busy sectors which could eventually lead to a large performance degradation due to enormous number of cluster combinations. By default, however, this cut is set to a rather high value of 10^5.

2. *Cleaning:* In the next step, if required by flag *CleanSeed*, the track seeds found on the triplet of modules are checked for uniqueness. This means the track seeds sharing clusters are dumped. The motivation for this option is to provide clear tracks, especially for alignment purposes, even in the case of events with large local densities of clusters (e.g. hadronic interactions in VELO

[a] Such as, for instance, working with large misalignments but high cluster density or dealing with various data peculiarities
[b] Note that the usage of triplet sagitta is the usual way of searching for track seeds and it was used in 2D case frequently, for instance in VELO (LHCb) or BST (H1) two dimensional $R-z$ pattern recognition.

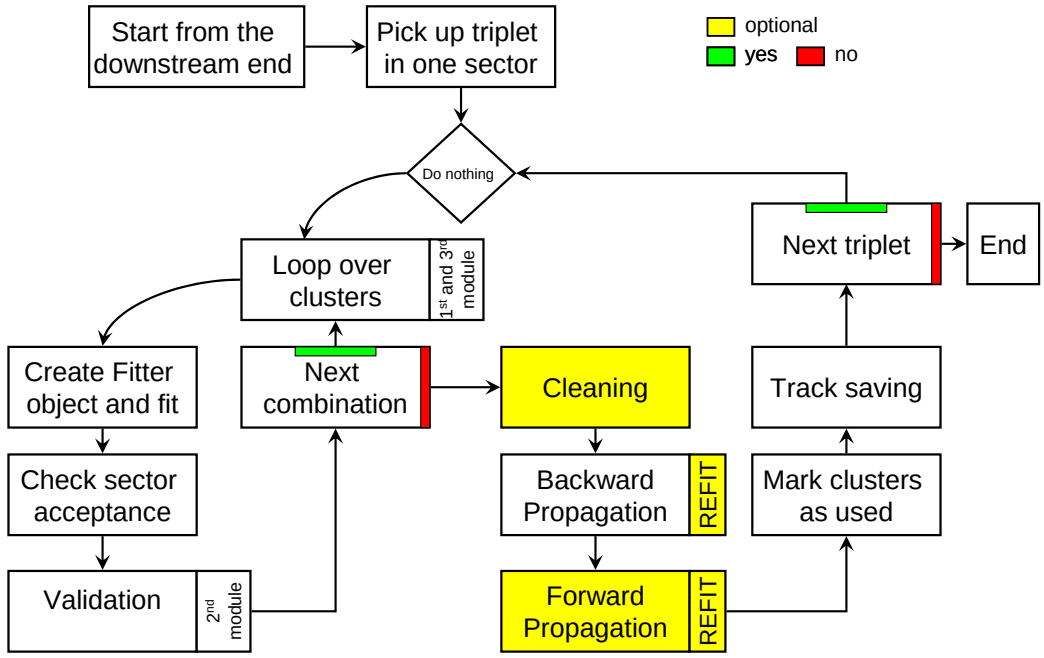

Figure 2 A flowchart diagram showing the Generic Pattern Recognition algorithm implemented in PatVeloGeneric class. See Section 2 for details.

sensor) and/or for eventually corrupted data (e.g. if clusters would appear twice and very close to each other etc.).

3. *Track propagation:* Once track seeds are cleaned, they are propagated upstream. The track is extrapolated to the next module and if R or ϕ clusters are close to the extrapolated point they are added to the track seed which is re-fitted. It is sufficient to have only one cluster per module assigned. During a track propagation it is allowed to accept a number of modules with no clusters contributing to the track, the number being specified by option *MaxSkip*. If the number of skipped modules exceed *MaxSkip*, the track propagation is terminated.

   Option *ForwardProp* allows to propagate the track seeds in the downstream (forward) direction. The propagation algorithm is identical to that of the default upstream propagation except the direction of the propagation.

4. *Saving:* Once the track propagation is finished the track is added to TES. The location of the track container is defined by *Output*. There is a number of options regarding the position and type of track State. See option *KalmanState* in Section 3 for more details. The track has a default Q/P assigned based on value of *DefaultMomentum*. Note that this allows the track to be fitted within the LHCb fitting machinery straight away, without a need to be prepared by an extra algorithm.

See Section 3 for a complete list of steerable options.

2.1 The Fitter Algorithm

Class PatGenericFitter is used to store information about clusters assigned to track seeds as well as to fit the clusters by a straight line in 3D. The least square fit is based on a minimization of χ^2 functional:

$$\chi^2 = \sum_i d_i^2/\sigma_i^2, \tag{1}$$

where sum is over all clusters, σ_i^2 is a digital resolution[c] and d_i is a distance of a track intercept (x', y') at sensors z_i-position from the cluster. The latter can be written as:

[c]Throughout this note the term *digital resolution* corresponds to a strip pitch divided by $\sqrt{12}$.

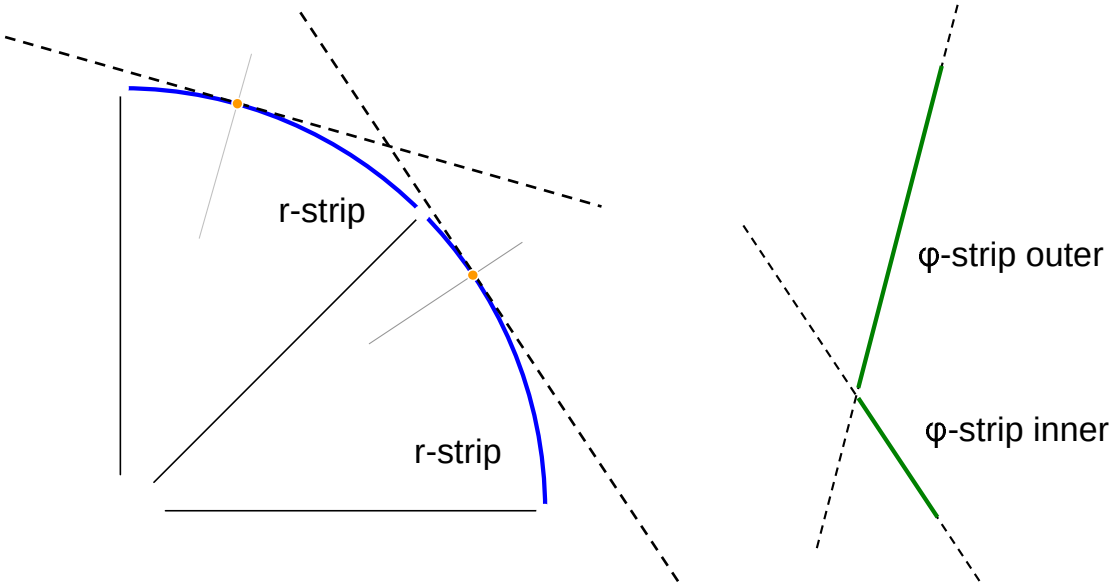

Figure 3 An illustration of the strip treatment in the fitting procedure. Dashed lines correspond to R-strips (blue arcs) and ϕ-strips (green lines) to be fitted. R-strips are approximated by tangents at ϕ coordinate from the previous fit (fitting is an iterative procedure). Acceptance of strips is handled outside the fitter from within the pattern recognition algorithm.

$$d_i = (a_i\, x' + b_i\, y' + c_i)/\sqrt{a_i^2 + b_i^2}, \tag{2}$$

where a_i, b_i, c_i are the parameters of line representing cluster i. The track intercept parameters are

$$\{x', y'\} = \{x_0 + s_x\, z_i, y_o + s_y\, z_i\}. \tag{3}$$

The 3D straight line parameters are hereby denoted as $\{x_0, s_x, y_0, s_y\}$. Equation (1) is then solved using matrix techniques.

Note it is rather straightforward to fit ϕ-clusters in this manner since these can be treated as straight lines at fixed z. However, R-clusters must be approximated by a tangent, as it is shown in Figure 3. This implies the fit to be iterative. The initial ϕ of a track is estimated from its ϕ clusters which quickly draw the tracks to its correct $\phi(z)$ dependence during the following iterations. Usually only few iterations are needed to make the fit and consequently the track position stable. As a criterion of stability it is sufficient if, between two consecutive iterations, a track moves by less than 0.1 μm in the whole VELO volume (-200 mm $< z <$ 800 mm).

It is worth to mention that the fitting algorithm is able to fit any sufficient combination of R and ϕ clusters and due to *stereo angle* of ϕ-strips it does not even require any R clusters at all.

3 Options and Flags

Due to required flexibility of the pattern recognition algorithm there is a large number of steerable options and flags. The options currently available[d] are listed below:

int KalmanState Sets the track State to the first (2) or the last (3) sensor contributing to the track, covariance matrix is large in this case, see options *ErrorX2* etc. Another option (1) is to re-fit the tracks using PatVeloSpaceTrack and put the State close to the LHC beam (z-axis). Recommended, but not default, option is (4) which uses the covariance matrix from PatGenericFitter.
Default: **1**

[d] Corresponding to the current version Pat/PatVelo v2r14 and, in general, to majority of other version. See CVS for a particular PatVelo version.

bool CleanSeed When seeds of tracks share same clusters they are rejected. In other words seeds are required to be unique with no ambiguity of cluster assignment. This option is rather useful for alignment purposes for cases with large misalignments, regions with large cluster density and various data artifacts.
Default: **true**

int ClusterCut The track seeding is not run for regions corresponding to R-sectors where the total number of R-clusters is exceeding ClusterCut value. This allows either to select very simple and clear events and/or to speed up pattern recognition in very busy sectors.
Default: **1E5**.

bool FullAlignment Performs own full alignment correction in PatVeloFitter, for closed VELO only. It pre-calculates strip position corrections once per event and stores the info in unused member variables of PatVeloCoordinate. To be used along with PatVeloLoadClusters.IgnoreAlignment=true.
Default: **false**

bool ForwardProp The track seeds are propagated forward, at top of the default backward propagation.
Default: **false**

bool DoNotRefit Forces no re-fitting during the propagation of track seeds leading to a modest improvement in speed. The tracks are, however, re-fitted when stored.
Default: **false**

bool CheckReadOut Only modules in readout are considered, others are skipped during pattern recognition (both track seeding and propagation). Used for ACDC3 test-beam configurations with 6 of 10 mounted modules in readout.
Default: **false**

std::string Output Track container output location.
Default: **LHCb::TrackLocation::Velo**

double DefaultMomentum Default charge to momentum ratio Q/P. Note that charge is not randomly generated. Tracks from Generic Pattern Recognition thus can be directly used in Kalman filter without any need to prepare them using e.g. TrackPrepareVelo algorithm.
Default: **1/10GeV**

double ErrorX2 as well *ErrorY2*, *ErrorTx2*, *ErrorTy2* and *ErrorQOP*. Diagonal errors for the large covariance matrix which is eventually (see option *KalmanState*) used to seed Kalman filter.

bool ACDC The flag was introduced to handle the fact that R-sensors are flipped around the x-axis in real VELO modules compared to the geometry used in simulations. This issue is expected to be corrected in the detector element[e] in future.
Default: **false**

bool PrivateBest When there are more cluster candidates per sensor to be assigned to a track they are all ignored. This may happen, for example, when extra broad selection corridors are allowed due to largely misaligned sensors.
Default: **false**

double RAliTol, *PAliTol* Alignment tolerance for the case of misaligned VELO. Note that values are in standard deviations and will be multiplied by the value of *SigmaTol* when calculating selection corridor width.
Default: **0**

double SigmaTol Tolerance to accept cluster candidates in units of the standard deviation. The corridor to assign clusters to a track is composed from sensor resolution[f] and the alignment tolerance.
Default: **4**

int MinModules Minimum number of modules used in a track. Set to 3 to accept very short tracks and even track seeds only.
Default: **5**

[e] Concerns localPhiToGlobal() method in DeVeloRType class.
[f] Assuming digital resolution, i.e. strip pitch divided by $\sqrt{12}$.

int MaxSkip Maximum number of modules allowed to be skipped during track propagation, i.e. not contributing to the track by any clusters.
Default: **2**

4 Examples of Applications

Test Beam During its construction VELO was tested two times at test-beam in CERN Prevessin, test runs are known under abbreviations ACDC2 and ACDC3. These runs were valuable to test both VELO hardware and software. Peculiarities of the data included partial readout of VELO modules, different geometry compared to the one used in simulations, effects originating from lower levels of data processing etc. The Generic Pattern Recognition was the default algorithm used to reconstruct first tracks seen in VELO (ACDC2) and, along with the Generic Vertexing Algorithm [3], as well as first vertices (ACDC3) [3].

Open Velo During injection and tuning of LHC beams VELO is in its open position. The maximal aperture is 30 mm on both sides and smaller when VELO is being closed. In this position VELO is still supposed to be able to reconstruct not only the position of the beam spot but also its profile. Generic Pattern Recognition reconstructs tracks in this 'non-linear' setup which then may be refitted by Kalman filter in order to reconstruct interaction vertices. See [1] for more details.

Beam Gas Events It was proposed in [2] to measure the absolute luminosity of LHC by measuring directly the parameters of both LHC beams via the reconstruction of beam-gas interactions. Generic PR was used for first studies of the beam-gas event reconstruction due to specific requirements for the reconstruction. Beam-gas events are also supposed to be seen during engineering LHC runs expected in November 2007. In these runs VELO will start to operate in its open position (see first item in this list).

K_s^0 reconstruction When run after the Standard Pattern Recognition the Generic Pattern Recognition can find some remaining tracks in VELO thus improving efficiency of e.g. K_s^0 reconstruction.

Interactions in material Secondary hadronic interactions in the VELO material may be used to measure positions of VELO sensors as well as of the RF-foil. Studies of a feasibility of such measurements are in progress.

5 Comments

Open Velo Treatment

PatVeloGeneric may be used for Open Velo pattern recognition straight away. The pattern recognition is performed in the coordinate system of each VELO half (box). When tracks are stored corresponding positions of Track States are corrected according to the box alignment, i.e. to both *shifts* and *rotations* of Velo boxes. In the present version the track direction, stored in State, is not box rotation corrected. This is a minor issue since the rotations are expected to be small and tracks are correctly re-fitted in the subsequent fitting procedure with Kalman filter. However, it limits usage of PatVeloGeneric Tracks directly for vertexing (as it was done during test-beam runs) for the case of both Velo halves used simultaneously, no Kalman filter re-fitting or non-negligible relative rotations of Velo boxes.

Speed Issues

The algorithm was not design to perform in terms of high speed but rather to be robust and reliable. This was proven during test-beam runs when it was extensively used and tested. Depending on number of clusters in VELO, however, the speed may become an issue for some applications. There are few tweaks possible which should significantly speed up the algorithm and are foreseen to be implemented in future.

- Number of combinations of 2 R and 2 ϕ clusters during track seeding is driving the algorithm speed performance. While there is not much to be improved regarding loops over R clusters[g] looping over ϕ clusters can be optimised. The main idea is that the acceptance of ϕ clusters with respect to R clusters is currently checked only after the fit is performed. I.e. in order to reduce a number of fits a range of ϕ clusters in acceptance of a particular R cluster can be known *a priory*. In the ideal case this would decrease the number of fits by factor of up to 16 thus leading to, naively, order of magnitude speed increase for the case of busy events.
- The algorithm is, by default, re-fitting track seeds after every cluster added and thus providing a new track estimate and a new covariance matrix of track parameters. Albeit this is the correct way to propagate long tracks, for relatively short tracks (3-5 neighbouring modules crossed) it may not be absolutely necessary. Thus an option *DoNotRefit* is provided to block the track re-fitting after addition of any clusters to the original quadruplet of seeding clusters. As a complementary speed tweak *CleanSeed* flag may be switched off. Note that this could potentially lower the quality of recognised tracks and increase the overall number of tracks thus potentially slowing down vertex reconstruction due to more time spent during Kalman filter fitting.

Generic Vertexing Algorithm

As it was mentioned, the Generic Pattern Recognition was used during test-beam runs as the default Pattern Recognition. In ACDC3 test-runs there was enough tracks to reconstruct vertices, however, they were not assumed to be on the z-axis (i.e. not coming from the default beam position) and thus a generic algorithm was used called Generic Vertexing. This algorithm can be found in Velo/VertexGeneric package and it is described in reference [3] in more details.

6 References

[1] T. Laštovička, *Open Velo Tracking*, lhcb-003-2007.

[2] M. Ferro-Luzzi, Nucl. Instrum. Meth. A **553**, (2005) 388.

[3] T. Laštovička, *Generic Vertexing Algorithm*, lhcb-063-2007.

[g] Except allowing only 'interaction vertex pointing' combinations, which would be against the idea of generic pattern recognition