LHCb 2003-027
TRIG
28 April 2003

Velo tracking for the High Level Trigger

Olivier Callot

Laboratoire de l'Accélérateur Linéaire – Orsay

and

CERN – Geneva

ABSTRACT

This note describes the new implementation of the Velo tracking, developed for the High Level Trigger (HLT), for which speed is a serious concern. The input data is the DAQ event as defined by the Velo group, the output is a HltTrack, which allows vertexing and further tracking. This note describes the algorithm, and reports the performance in terms of speed and efficiency.

1 Description of the problem

The High Level Trigger must reduce the number of accepted events from 40 kHz input to 200 Hz output, selecting the interesting B decays. The idea is to reconstruct the tracks with accurate momentum measurement, perform vertexing and select the wanted topologies. A first step is then to reconstruct the Velo tracks. This was already done in Level1, but for technical reasons we can't pass enough information from Level1 to HLT. Also, the input data can be somewhat more verbose, giving a more precise Velo cluster position. The strategy is to perform the complete tracking on all tracks, as we want all B-decay products. This differs from the Level1 requirement, where finding some good B-decay tracks is enough to keep the event. In short, we want full efficiency and maximum speed…

2 The algorithm

After some data preparation, the algorithm proceeds in two phases: First the tracks are searched in the R-Z projection, using only R sensors. For tracks coming from the beam line, centre of the R sensor geometry, straight tracks are straight also in the R-Z plane. For tracks having a small distance of approach to the beam line, this is almost true. We are looking for B-decay tracks, it has been shown that their impact parameter is lower than 2 mm, this gives an order of magnitude of the misalignment to be accepted. Note that the Velo sensors are sensitive between 8 and 42 mm in radius.

The second step is to find Phi coordinates along the R-Z track, such that the track is straight in space. A final fit allows providing a fitted trajectory for the next stages in the HLT.

2.1 Preparing the VeloClusters

A first algorithm converts the **`VeloCluster`** objects to the DAQ format. This is not the role of the HLT, but the official Velo software for simulating raw data is not yet available. The format is made of 32-bits words containing packed fields as shown below:

ADC strip 0	First strip	L1 pattern	size
ADC strip 4	ADC strip 3	ADC strip 2	ADC strip 1
0	0	ADC strip n	ADC strip n-1

In the frequent case the cluster has a single strip, a single 32 bits word describes the cluster. The "L1 pattern" is intended to store the bit mask of strips over the Level1 readout threshold in the cluster. "size" is on 4 bits, "First strip" on 11 bits.

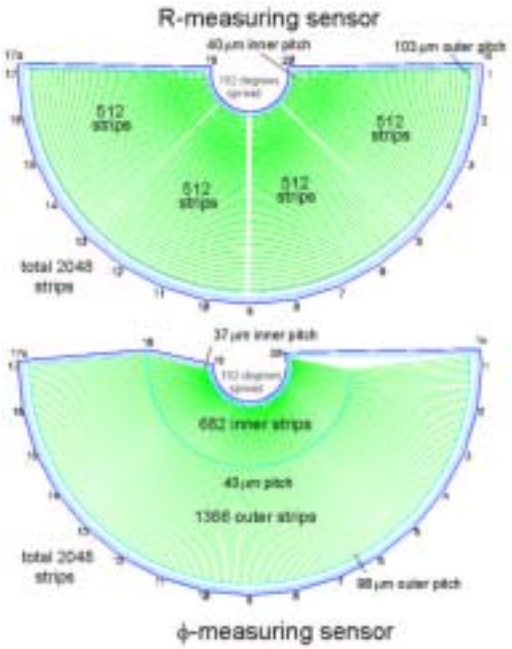
A dedicated algorithm named **`PrepareVeloClusters`** produces a buffer of 32-bits words according to this format, each Velo sensor being prefixed by a word specifying the sensor number (top 16-bits) and the number of clusters (lower 16-bits). No cut on the number of clusters are performed. However a cut on the total signal in the cluster is performed, to remove the noise hits that are not useful for tracking, and part of the spill-over clusters. The cut is at 10 ADC counts.

In order to be able to access the truth information, one keeps also a list of **`VeloCluster`** pointers, one per cluster in the buffer. One can then later have a reference to the original **`VeloCluster`**, which can then be associated to the true particle(s) that created it.

It should also be noted that the clusters are sorted to allow fast search. The sorting is by increasing measured coordinate, which is usually the same as the strip number, except for half the phi sensors where the strips are numbered in the other direction. For those sensors, sorting is by decreasing strip number.

2.2 Geometry preparation

The Velo geometry is described in a **`DetectorElement`**, able to answer several questions. However, speed is our main concern here. One of the main handle to improve speed is to have smart objects, able to cache locally frequently used information. Geometry is one example.



The R sensors are divided in 4 sectors covering 45.5° each, with 512 strips. The radius of the centre of each strip is kept in a look-up table. This is currently the same look-up table for all sensors, but could easily be changed if alignment becomes an issue.

The Phi sensors are divided in two zones, inner and outer part. The difficulty with the Phi sensors is that they don't really measure Phi, but a tilted Phi. The real Phi depends on the strip number AND of the radius at which the strip is hit. One need to know the phi of the first strip in a zone as function of R, this is a formula of the form $\phi = \phi_0 + \arcsin(R_0/R)$ where the two constants have to be computed once. One needs the pitch, constant in angle. One stores also the range in Phi covered by each of the R sectors (numbered from 0 to 7) to allow a fast reject of impossible R-Phi combinations.

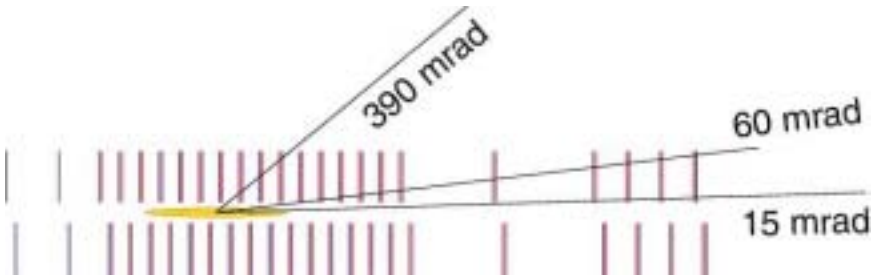
The object "**`VeloSector`**" holds these information, together with the hits in this sector, and has methods to search the best hit in his list.

2.3 Data decoding

The first step in the processing of an event is to unpack the DAQ event into usable objects, called **`VeloCoord`**. These are just functional representation of a hit, holding position, error, radius, angle, cosine and sine (to avoid repeated trigonometric computations). In order to limit the overhead of memory handling, a vector of such objects is created in one go (we know the total number of cluster by a fast scan of the DAQ buffer), and we just set the properties of each cluster in turn during decoding. One should be sure to not extend this vector, as all pointers would become invalid. The **`VeloSector`** object only holds a list of pointers to **`VeloCoord`**. The barycentre of each cluster is computed using the ADC content. It should be noted that too wide clusters are ignored, namely cluster of size greater than 3, as they don't add information on the coordinate of the track, and are creating ghosts too easily.

2.4 R-Z tracking

The basic step in the R-Z tracking is to find a triplet of aligned clusters in three consecutive R sensors, on the same side of the beam line. We work in each of the 8 sectors independently, i.e. we don't look at tracks crossing a 45.5° boundary. The range of slope in the R-Z plane is limited, we want the track to have a radius increasing with Z, and a maximal angle of 400 mrad.



2.4.1 Finding triplets

Let's call S0, S1 and S2 the three sensors in the triplet. We start with the most downstream sensor for S0, S1 is the previous one, S2 is just before S1. This means that we go opposite to the track's direction, starting from the sensor where the tracks are most separated. We iterate on the clusters in S0, for each cluster we iterate on the clusters in S2. We loop on the possible pairs in the angular range 0-400 mrad. The first valid cluster in S2 is memorized, and we restart from that one for the next cluster in S0, as previous clusters will be outside the angular range. For each pair, we predict the position in S1, and search for the closest cluster. Here also we keep track of the first good cluster in the search range, to restart from it the next time, with automatic detection when this assumption becomes invalid. Loops are stopped (`break;`) as soon as the rest of the clusters can not satisfy the condition. The search window for the cluster in S1 has a width of 0.90 times the pitch at this radius.

In order to avoid finding again the same piece of track when starting from the next sensor, we ask the first cluster to be unused.

2.4.2 Track extension

Once we have a triplet, we try to extend it as much as possible, predicting the radius in the next sensor and finding the closest cluster. The search window is now 3.5 times the pitch. This large value is needed to accept non pointing tracks, for which the R-Z projection is not exactly a straight line. The way to compute the predicted radius is also important. A simple linear fit of all previously found R coordinates is used.

If the triplet is not extended, we ask all hits to be unused, as the probability to have a ghost in this case is quite high. For longer tracks, we ask for at least 2 unused hits on the track. If the track has 4 or more clusters, they are tagged as used.

As sensors can be inefficient, we search also triplets with a missing sensor, namely (S0, S1, S3) and (S0, S2, S3) but only for clusters not used in a previously found track. This avoids finding three times the same track. The cost in time is quite low, as long as the sensors are reasonably efficient.

After having exhausted all combination, the starting sensor S0 is changed, going towards the interaction point, until no track with the maximal slope can come from the luminous region. We don't perform a search for the tracks going backwards, i.e. not in the spectrometer acceptance. They may be useful for vertexing, and this is easy to add if/when needed.

2.4.3 Clones at vertical boundary

The two halves of the Velo have some overlap, about 3°. Tracks in this region are then measured twice, on the right and on the left sensors. To decrease this number of clones (they are the same track, but don't share any cluster), we perform a dedicated search: When in one of the sectors in this overlap region, we search for clusters in the corresponding zone of the other sensors, with a search window of only 0.6 times the pitch. If we have 3 or more such clusters, they are added to the track, and tagged as used. We can then obtain tracks which are on the two halves, which will be useful for aligning the detector.

2.4.4 Event rejection of busy events

It is well known that the trigger tends to select very busy events, but the offline analysis doesn't like them. A cut is applied on the total number of R-Z tracks found. If higher than 250, the event is not interesting and is rejected.

2.5 Space tracking

The space tracking is somewhat new compared to previous Velo tracking algorithms. The idea is to collect hits in Phi sensors which, when associated to an R-Z track, make a straight line in space. Of course, it is difficult to select immediately which Phi cluster to associate. The trick is to build lists of compatible clusters, and once all clusters in all sensors have been processed, to select the best list. Reducing the number of combinations is the issue. Fast decision is also important. We process all R-Z tracks, one after the other. Tracks are sorted by length, i.e. by number of R measurements. This allows to search first for the best tracks, and to remove their clusters for future searches.

2.5.1 Select sensors

First, one defines the first and last Phi sensors that can be crossed by the track, simply by testing the radius at the sensor's position. We start again by the last sensor, the one at the highest Z. The two halves are handled simultaneously, as tracks close to the vertical boundary between the two halves will have part of their point on one side, part in the other, and until we have found them we have no method to avoid searching on both sides. Except of course for tracks in the two central quarters of each R sensor: The only possible Phi hits are in the sensor of the same side.

2.5.2 Building the list of Phi clusters

For each sensor, the coefficients to convert a Phi strip to a Phi coordinate (see 2.2) are computed, and the range of Phi coordinates values is computed, from the boundary of the R sector in which the track sits. A quick test on an empty range allow to skip Phi sensors without overlap with the R sector.

For the first two sensors (one on each side), we just create a **`VeloPhiList`** for each Phi cluster.

For the next sensors, we try to match each cluster with an existing list. If there is no match and the cluster was never used, a new **`VeloPhiList`** is created, but only for the first 3 pairs of sensors. After that, the track would have too many missed hits and it is not worth adding more lists to test.

For each cluster, the operation is first to convert the strip number to its angle, which gives then a point (x,y) in space. The matching with existing **`VeloPhiList`** is performed by comparing the distance in space between the predicted trajectory and the point. Note that the radius of the point

is taken from the R-Z projection of the track, using an interpolation of the R measurements, and the matching is performed using the projected x-z and y-z straight line parameterization of the **`VeloPhiList`**.

A difficulty is to allow multiple combinations: For each **`VeloPhiList`**, we keep only the best cluster for a given sensor. The same coordinate can be used in several **`VeloPhiList`**, and we want to create a new list if the coordinate is not used when in the first tested sensors. When a new cluster is kept, the parameterization of the track is adjusted, using the (x,y) point corresponding to the cluster. We use the R measurements only indirectly, because they are used in converting Phi to (x,y).

When all Phi sensors have been scanned, the best **`VeloPhiList`** is selected. The one with more clusters, and the one with the best χ^2 if several candidates have the same number of clusters. Of course a minimal length is requested, 70% of the tested Phi stations (35% of the sensors) should have a cluster.

2.5.3 Final fit and storage

Once all the R and Phi hits are collected, the track is fitted using all clusters and the best estimate of the errors, and eventually converted to the storage format of **`HltTrack`** and put on the transient store.

2.6 Availability of the software

The software is available in cvs, as package **`Hlt/HltVelo`** in the cvs repository. It requires the package **`Event/HltEvent`** for specifying the output track, **`Hlt/HltTools`** for time measurement tools, and **`Hlt/HltChecker`** for efficiency measurement.

3 Performance

The efficiency should be measured on signal events, as we want the maximal efficiency on B-decay tracks. The difficulty is to decide which B-decay tracks. It may be easier to find the two pions of $B\rightarrow\pi\pi$, with large P_T than the low momentum tracks of a larger multiplicity channel, like $B\rightarrow J/\psi(\rightarrow\mu\mu)\ \phi(\rightarrow KK)$. The tracks one want to find are "long" tracks, meaning they must have enough clusters in the Velo (3R,3Phi) and in the T stations (X and U/V hits in each station). "Long B tracks" should have also a particle with a b quark in its ancestors. And if this ancestor has all its final decay products either "long track" or photons over 1 GeV, this is a "Good B track".

The events should also have passed the Level0 and Level1 triggers.

Type	Ghost rate	Long tracks	Long B tracks	Good B tracks
Minimum Bias	5.7 %	95.5±0.2 %	95.8±1.5 %	-
$B\rightarrow\psi(\mu\mu)\ \phi(KK)$	4.6 %	95.8±0.1 %	96.5±0.2 %	98.0±0.3 %
$B\rightarrow\pi\pi$	4.7 %	94.9±0.2 %	96.4±0.3 %	97.6±0.5 %

The speed of the algorithm is relevant on Minimum Bias events passing Level0 and Level1 triggers, as this will be the input of the High Level triggers. It is difficult to measure accurately very small time on a Linux computer. Either we use the "user time" which is known by 10 ms steps, requiring a very large statistics to get a few percent resolution on a time close to 5 ms. Or we use the "clock time", which is accurate but sensitive to the load of the machine. Several measurements are needed to avoid spikes and accidental long delays. The measurements are done with about 1000 events on an Lxplus7 machine, with a Pentium III at 1.0 GHz, and expressed in ms.

The last row indicates the time for minimum bias events having passed Level0, and is an indication of the speed of this algorithm if used in Level1.

Type	Decoding	R-Z tracking	Space tracking	Storage	Total
Minimum Bias	0.99	0.83	3.65	0.46	5.93
$B \rightarrow \psi(\mu\mu)\ \phi(KK)$	0.82	0.66	2.73	0.37	4.58
$B \rightarrow \pi\pi$	0.80	0.74	2.67	0.36	4.57
Min.Bias. after L0	0.76	0.58	2.17	0.32	3.83

This speed is faster than the current Velo offline tracking by a factor about 40. The performance for the use in Level1 are adequate for this application too.

Without the cut on the cluster charge, the rate of ghost tracks is multiplied by 3, the efficiency decreased by 0.5 to 1 % and the time taken is increased by 0.8 ms. The ghost rate, and maybe the efficiency, could be recovered by checking the average charge of a track, as part of the tracks may come from spill-over particles, with a lower average charge.

4 Acknowledgements

I want to thank Mariusz Witek for useful suggestions, in particular for the setting of the compiler flags. The Velo tracking was first developed by Frederic Teubert and Ivan Kisel, and I clearly benefited from their work.