FastVelo, a fast and efficient pattern recognition package for the Velo

O. Callot

FastVelo, a fast and efficient pattern recognition package for the Velo

Olivier Callot

Laboratoire de l'Accélérateur Linéaire - Orsay

ABSTRACT

This note describes the new implementation of the Velo pattern recognition package developed in autumn 2010 to cope with the changed requirements for LHCb tracking: No need for RZ track in the trigger, higher collision rate (so called μ) around 2.5 instead of 0.4 and the need for a faster algorithm for HLT1 to cope with the high rate and large occupancy.

After describing the main idea, this note describes the algorithm in details with the meaning of the control parameters. Some performance is also given.

1 Introduction

The most important LHCb tracking detector is the Velo. This is where the position and direction of the produced particles are measured. The rest of the tracking system is there 'just' to measure the momentum. Reconstructing efficiently the tracks in the Velo and with as few as possible wrong tracks (so called ghosts) is thus a critical item for the overall LHCb performance.

The algorithm in use for the first data taking year at LHC was written by me in 2003 (1), updated in 2004 (2) and 2006 and then maintained by David Hutchcroft with some adaptation to the updated geometry (3).

During the 2010 run, it became clear that the HLT doesn't need RZ tracks, because the Velo can't close completely and thus the RZ tracks are not as good as foreseen on Monte-Carlo for selecting tracks with high impact parameter. The second change is that, instead of working with a single interaction per crossing as foreseen in the preparation of the experiment, we took data with an average number of interactions per crossing (μ) of 2.5, and we intend to run with a value of μ in the range 1.5 to 2.5 in 2011. The code should then be revisited and tuned for this new context. Last, the HLT is saturating the computing resources of the online farm, partially due to the more complex events selected at high μ and an improvement in speed is needed to cope with the expected 1 MHz rate and will give some margin for other modifications in HLT.

During the autumn 2010, a new package has been written from scratch, of course with some inspiration of my previous implementations. This note describes the new approach, released in a package called `Tf/FastVelo`.

2 Basic principles

The Velo is made of two types of sensors, R sensor with strips at constant radius covering 45° in a so called sector, and Phi sensors with strips almost radial, grouped in two crowns. A schematic view of the Velo sensors is given in Figure 1 and the overall Velo is shown in Figure 2.
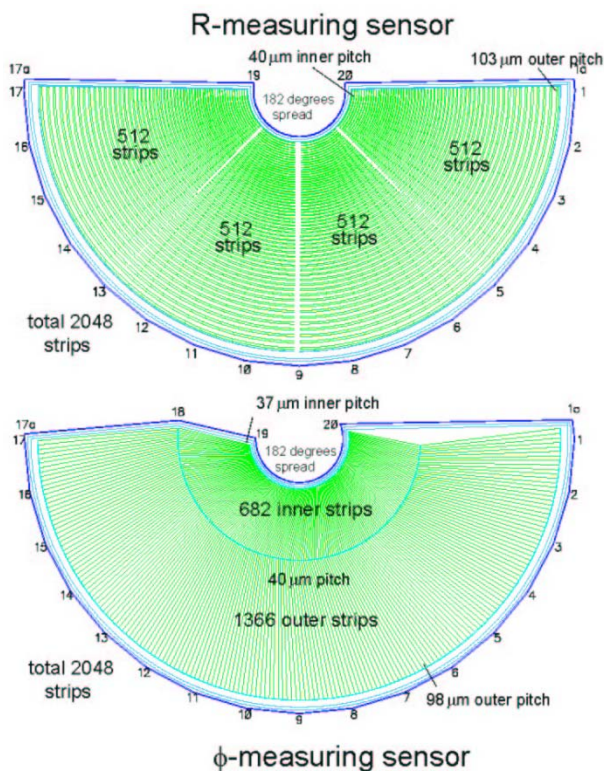
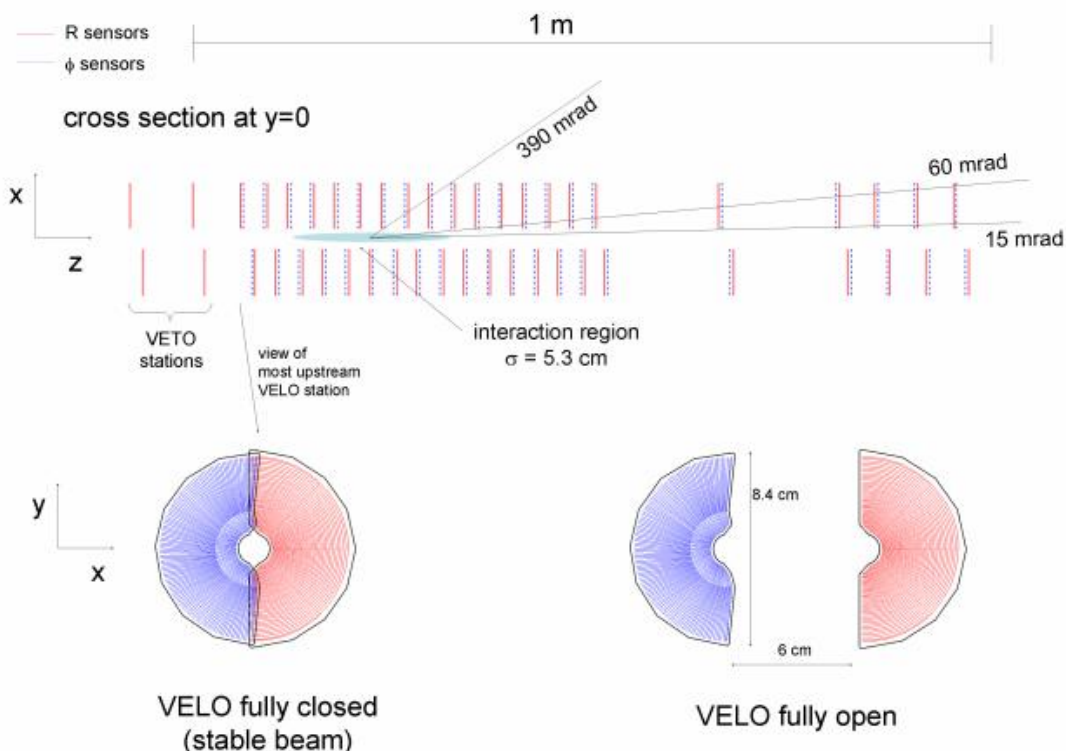

Figure 1 Layout of the Velo sensors

Figure 2 Schematic view of the Velo

2.1 Reference frame

The real detector is not as perfect as in the design: The sensors are not absolutely perpendicular to the Z axis, their centres are not perfectly aligned, and the closing of the two halves is far from perfect, there are about 300 micrometers between the centres of the opposite side sensors. It is then very important to define properly in which reference frame the tracks are searched for. In the FastVelo implementation, one works **always in the LHCb global frame**. This is where the strip position is best known by the alignment. This frame is easy to use for the phi sensors, where the strip is a straight line. For the R sensors, where the strips are an arc of circle centred off-axis, one has to make an approximation in the first part of the code, where one works in R-Z projection: For each 45° sector, the 'R' coordinate of a strip is the distance along the bisector between the projection (perpendicular to the bisector) of the beam axis and the intercept of the R strip.

2.2 R-Z tracking

Tracks mostly come from the beam axis, or close to it, and they originate from the luminous region. In a first approximation, this means that tracks are also straight in the R-Z projection and that they come from a region in Z around 0 with a sigma around 5 cm. A first step of the pattern recognition is to find these (almost) straight lines in the R-Z projection, i.e. using only R sensors. But we are interested in B-decay daughters; these tracks come from a displaced vertex that is <u>not</u> on the beam axis. Similarly, and with larger effect, tracks from K_S^0 or Λ are not at all coming from the beam axis and thus are not straight lines in the R-Z projection. It should be noted that having spider web geometry, i.e. straight strips in 45° sectors, would have made the pattern simpler as straight lines (tracks) stay straight lines in this projection. But one can't change the geometry now!

It was noted that most of the tracks we want are crossing at least 4 R sensors. In order to be faster, tracks with 4 hits will be searched first. Once all are found, tracks with 3 hits will be looked at, but using only unused clusters: Tracks made with 3 hits have a larger rate of ghost; asking for unused hits, i.e. for hits not already used on longer tracks, reduces the rate of ghosts.

The tracking is done first for forward going tracks, then backward going tracks. The **quadruplets** are searched starting from the last four sensors, where tracks are most separated. Once a quadruplet is found, the track is extended towards the lower Z region as much as possible, allowing for some inefficiency, i.e. missing sensor. If the track is in one of the 45° sectors close to the vertical boundary between the two Velo halves, one searches for clusters on the opposite side that can be on the same track. If there are enough, they are added to the track which is then assumed to be almost in the vertical plane. Clusters are tagged as used and cannot be reused as seed for another quadruplet. The quadruplet is also allowed to have a missing sensor, so 4 out of 5 sensors are enough. Once all hits of the last sensor have been processed, one continues with the next sensor. The search for quadruplet stops when the 4 sensors cannot contain tracks coming from the luminous region, i.e. even at the maximum allowed angle the track would cross the Z axis before the luminous region.

A second step is to perform the same quadruplet search in the reverse direction, i.e. for tracks going towards negative Z. The code is the same, and one stops also as soon as tracks cannot come from the luminous region.

Then short tracks (triplets) are searched for, first backward then forward, requesting that the three clusters are unused. Tracks are also extended, both towards the luminous region, and if relevant in the overlap. As three points is not a very strong constraint to produce a straight line, triplets are compared and if two triplets share a hit, only the best one is kept.

This R-Z tracking is in fact very similar to previous implementations, only the search for quadruplet first, plus tuning and details in the logic that will be described later, are really new.

2.3 Space tracking

Starting from a RZ track, one searches Phi measurements that can make a straight line. In previous implementations, the real phi coordinate was obtained using the radius in the phi sensor with some trigonometric functions, which is not very fast. The new approach developed in the FastVelo is to use the distance to the strip, a linear function of x and y. However, to select the hits to be considered, one needs an estimator of the distance of the hit plane of the bisector, as hits from the same track should group for this estimator. We use the sine of the angle to the bisector of the R sectors: The radius is known from the RZ track, and one computes the intersection of the phi strip with a circle (R-strip) centred on the sensor's centre. This implies to compute one square root, but there is no trigonometric function. As the track is in a 45° sector, the sine has to be smaller than about 0.4. Hits are stored per station: the phi sensors of the right side can measure coordinates for a track on the left side and vice versa. They are sorted according to this sine value. One searches for a triplet of hits in three stations having a close enough sine, and then performs a full space fit with the R measurements and these 3 first phi measurements. If OK, the track is extended in space as much as possible. The triplet is first searched in consecutive sensors, and then a missing sensor is allowed. Some cleaning is done to remove identical or very similar solutions.

2.4 Non pointing tracks

Some good tracks are not found by this procedure. David Hutchcroft (3) had implemented a recovery for non pointing tracks in the previous generation of code, called **PatVeloGeneralTracking**. A similar recovery for non pointing tracks has been implemented in **FastVeloTracking**. One tries to find tracks starting from phi measurements only, and only for forward going tracks: backward tracks are useful only to find the primary

vertex, non-pointing tracks won't help. As there are fewer constraints in a phi tracking compared to the RZ tracking, this search is performed only on unused hits. A triplet of unused phi measurement has to be found in consecutive sensors. A rough test of compatibility is made using the centre of the phi strips: they have to be reasonably aligned with large tolerance. Then a first R hit is searched for in the R sensor corresponding to the first phi sensor. With these 4 measurements one gets a parameterisation of a space track. Hits in the two other R sensors are searched for. A good track requires 6 hits (3 R, 3 Phi) in 3 consecutive stations. At least 2 of the 3 R hits should be unused so far. The track is then extended as much as possible, using the space parameterisation. It should be noted that there is almost no pointing constraint in this search for track; the only request is that the first two phi hits are reasonably close.

2.5 Debugging

In order to understand the inefficiencies, or the ghost rate, it is important to be able to follow the processing of a selected **MCParticle**. The code should not have any reference to MC truth, but we have implemented a tool with a simple interface to select if a hit is from a given **MCParticle**. By job options, one can enable this tool with the option

```
FastVeloTracking().DebugToolName = "PatVeloDebugTool"
```

and selectively debug the processing of hits from the selected **MCParticle** by the option

```
FastVeloTracking().WantedKey = 2021
```

and follow what the code does, and why the track is not found. This tool also prints the truth information for a hit, in the form of a list of **MCParticle** keys.

An algorithm is added in the checking sequence of Brunel, called **DebugTrackingLosses**, which prints the truth information of missed track, or ghost, for a selected part of the tracking. This will give the event number and MCParticle key of the tracks one wants to debug. The following options are useful. Note that the last 4 are exclusive

```
DebugTrackingLosses().Velo = True
DebugTrackingLosses().MinMomentum = 1000.
DebugTrackingLosses().FromBeauty  = True
DebugTrackingLosses().FromStrange = True
DebugTrackingLosses().Ghost   = True
DebugTrackingLosses().Clone   = True
```

In fact, a large fraction of the code is made of debug statements, conditional to the detection of the use of a hit from a selected **MCParticle**. To use it, one should enable the decoding of MC information before running the reconstruction; this is done in Brunel by adding the option "Truth" in the `RecoSequence` before the "VELO" option.

3 Technical implementation

The package has one algorithm to perform the complete tracking. One tool handles the data; the track object handles also the fit. In this note, the main features are described, for all details on the code the only reliable information is to scrutinize the code itself!

In the following descriptions:

- **ThisName** is the name of a C++ class

- **ThisOption** is the name of a job option, and is usually followed by the default value.

3.1 Hit Manager

As the representation of a measurement is different from the previous implementation, and to optimise memory usage and speed, a clean and simple implementation of the hit manager has

been coded. The hit manager has two purposes: Give access to a geometry description of each sensor, and give access to the hits of this sensor.

3.1.1 Geometry description: `FastVeloSensor`

The **FastVeloHitManager** owns a list of **FastVeloSensor** objects that can answer many questions about geometry, but also provides the hits of this sensor. The geometry information cached in **FastVeloSensor** is extracted from the Velo detector element. One large part of it is the parameterisation for the phi strips: This is 2048 times 5 numbers (3 for the line parameters, 2 for the strip centre) for 42 sensors. Access should be fast, and experience showed that one need to reduce the overall size, storing them as **float** and not as **double**. As this is pure geometrical information, it could eventually be included in the Velo detector element. The other parameters are the position and inclinations of the sensor itself, and the offsets for the R sectors, i.e. the projection of the centre of the sensor on its bisector.

3.1.2 Hit description: `FastVeloHit`

The hits are constructed from the **VeloLiteCluster** decoded from the raw data. The relevant geometrical parameters (position, weight = $1/\sigma^2$, strip centre, parameterisation of the distance) are stored in the hit, plus some working variables: One is the number of tracks using this hit, another is a working coordinate, typically a distance to something, used to sort the hits and get the best ones. For details, see the source files. Contrary to the previous implementation, the R and Phi hits are represented by the same class.

3.1.3 The tool itself: `FastVeloHitManager`

This is the tool used in the tracking to provide geometry and hits. It owns a pool of **FastVeloHit** that is reused from event to event. The geometry is constructed at initialisation, and an update is triggered whenever the geometry information (i.e. position in the LHCb frame) of a sensor is changed, this means when the Velo is open/closed. In fact one checks only on the first sensor on each side. The hits are cleared on the `BeginEvent` incident, and built from the **VeloLiteClusters** in a dedicated method.

3.2 The pattern algorithm: `FastVeloTracking`

The pattern recognition is a single algorithm that handles the sequencing of the track reconstruction as described earlier, and then stores all the tracks in the official container of **LHCb::Track**. Inside the algorithm, tracks are built and stored in a dedicated object called **FastVeloTrack**.

3.2.1 The internal track: `FastVeloTrack`

The internal track has basically two lists of hits, R and Phi. It has also a few parameters (zone in R sensor, backward flag, valid flag and missed sensors) and the temporary storage and final results of the fits, R-Z fit and full space fit. Several methods are implemented to access this information, add or remove hits, produce the state vector and the covariance matrix. Three methods are providing the transformation of an RZ track to a space track, with different input according to the needs. A second family of methods is related to the space fit: add or remove cluster, fit a space line, remove the worst iteratively until its χ^2 is below a specified value. A last useful method is to select the best cluster in a list to add to the track if good enough: This is the basic method to extend the track once it has a good seed.

3.2.1.1 The track fit

The fit of the space track is performed using the linear form giving the distance to the strip. For Phi strips, this linear form is constant; this is a property of the strip, more exactly of the cluster as we interpolate between strips. For R strips, this linear form describes the tangent to the strip at the proper azimuth. This means that this form should be updated whenever the parameterization of the track changes, i.e. when the azimuth changes. In practice, there is no need to iterate as the form changes in an insignificant way if the track moves by a phi strip pitch: As the minimal radius is 8 mm, a displacement of 50 micrometer will change the radius by less than 0.4 micron, totally negligible. But care should be taken to update all R hit parameters after adding Phi hits and before fitting a track.

A track has four parameters: x0, tx, y0, ty. The χ^2 to minimize is: $\sum \frac{(ax+by+c)^2}{\sigma^2}$ which is: $\sum \frac{(a(x0+tx.z)+\ b(y0+ty.z)+c)^2}{\sigma^2}$. Computing the derivatives with respect to x0, tx, y0, ty gives four equations with four unknown, whose coefficients are sums of terms depending on a, b, c and z. The sums are computed and updated when adding/removing a hit. Several methods were tried to solve the system as fast as possible, including the recent Choleski decomposition method available in Root. The simplest solution by substitution was shown to be the fastest, around 5% of the total time compared to the Choleski decomposition.

3.3 RZ tracking

As explained earlier, two methods are used: Finding and extending quadruplet, finding and extending triplets of unused hits.

3.3.1 Finding quadruplets

One loops on the 'first' sensor, which would be the last one hit by the track in fact. For this sensor referred as 's0' one loops on the four 45° sectors, named "zone" in the code. It is assumed that a track doesn't cross a zone boundary, which is true for 99.9% of the tracks when the Velo is closed. For the open Velo, tracks crossing the boundaries are found by the thirst step, the 'unused phi' tracking.

Four cases are then processed one after the other

0.	take 4 consecutive sensors,	X X X X
1.	Allow the fourth to be missing	X X X . X
2.	Allow the third to be missing	X X . X X
3.	Allow the second to be missing	X . X X X

There is no need to allow the first to be missing; this is like using the next sensor...

For each unused hit in s0, starting from the innermost hit, define a range for the hit in s3: The minimum radius is given by the radius of the current s0 hit scaled with the maximum acceptable slope **maxRSlope** = 450 mrad to the z position of s3. The maximum radius in s3 is the radius in s0 plus one millimeter, this allows finding beam halo tracks, and also beam gas tracks coming from far. For backward tracks, the maximum radius is defined to find track coming from up to z = **zVertexMaxBack** = 1200 mm. This is enough to find beam2-gas tracks. The hits in sensor s3 are then scanned, starting from the saved first hit: This is the one that was the first inside the search window for the previous s0 cluster: When the radius increases in s0, the search windows in s3 shifts also, and hits in s3 that were below the window are definitely no longer to be looked at further. This gains some speed in the loop. The loop on s3 is aborted as soon as the hit is at a larger radius than the maximum. It should be noted that for the cases 1-3, i.e. with a missing sensor, the hit in s3 should not be used already.

From these two hits, one can predict the radius in s1 and s2, assuming the track is a straight line in the R-Z projection. The tolerance to find a hit is **rMatchTol4** = 1.0 in units of the local pitch.

This means that the centre of the cluster should be within ± 1 pitch from the expected position. The cluster width is not taken into account. If hits have been found in s1 and s2, a valid quadruplet is found and a track created.

The track is extended towards its origin until either the extrapolation is outside the sensitive area of the sensor, or until there are more than **maxMissed** = 1 consecutive sensors without hits. The tolerance to find a hit is defined as **rExtraTol** = 4 times the error on the extrapolated position, plus one pitch. After that, one checks that at least 2 hits are unused, to avoid finding again the same tracks. Clones should be killed as early as possible!

If the track is in zone 0 or 3, i.e. can be close to the vertical plane and thus have hits also in the other side of the Velo, one search for hits on the opposite side. The prediction is quite accurate, and the tolerance to collect hits is **rOverlapTol** = 1.0 in units of local pitch. If we have found more than 2 hits, all these opposite side hits are added to the track.
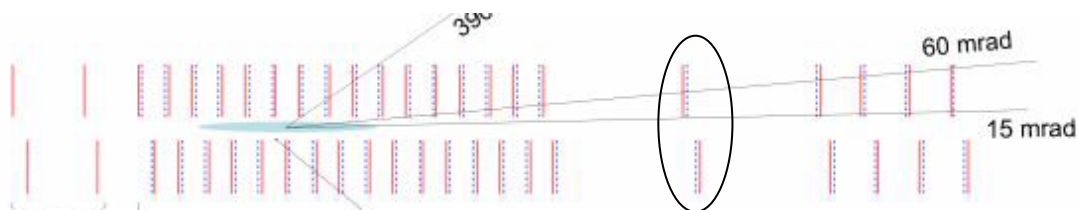

Figure 3 Location of the gap sensor

If a forward track ends at or just before the "gap sensor" indicated in Figure 3, i.e. in an area where the z distance between sensors is big, one tries to extrapolate the track downstream, i.e. find hits in the last sensors. The experience shows that the lever arm of the extrapolation is big, and two short segments before and after the sensor in the gap are frequently built.

The track is now stored in the local track vector. If the track has more than **minToTag** = 4 hits, the hits are tagged as used.

3.3.2 Finding triplets

The procedure is very similar to the previous one for quadruplets. The difference is that we have only 3 hits to find, so only 3 cases to consider, all 3 hits should be unused and the tolerance for the central hit is **rMatchTol3** = 1.1 times the pitch. If the track has only 3 hits after extension, one checks that the track has not missed too many sensors downstream i.e. doesn't stop too early in the Velo: The density of ghosts in short incomplete tracks is too high. There is also a special handling around the "gap sensor": We frequently find tracks with 3 hits in the last 3 sensors, a missing hit, and a <u>single</u> extra hit in the gap sensor or before. This is because the extrapolation tolerance is proportional to the distance in z of the extrapolation, which becomes quite large in this case. It was found that this extra far hit should be removed as it is almost always a randomly picked up hit.

A special operation is performed to clean up 3-hits tracks: If two tracks of only 3 hits share at least a hit, only the one with the best χ^2 is kept. The accepted tracks are stored in the same local track vector and the hits tagged if the track is long enough.

3.4 Space tracking

Each RZ track is processed in turn. In order to build the more constrained tracks first, the RZ tracks are sorted by length, and longest tracks are processed first. Any criterion of unused hit is then robust, as previous tracks are longer and have then very few ghosts.

A first processing step is to define the first and last Phi sensor to use. This is done by checking when the radius of the track (in the local sensor frame) is inside the silicon. Then Phi hits in this selected sensor range are collected, selected and stored per station. The grouping per station is needed to handle properly the overlap of the two sides of the Velo when closed.

3.4.1 Computation of the selection variable.

The key ingredient here is the computation of the selection variable, which is the sine of the angle to the bisector.

R from RZ, at central
azimuth of the R zone

Phi strip

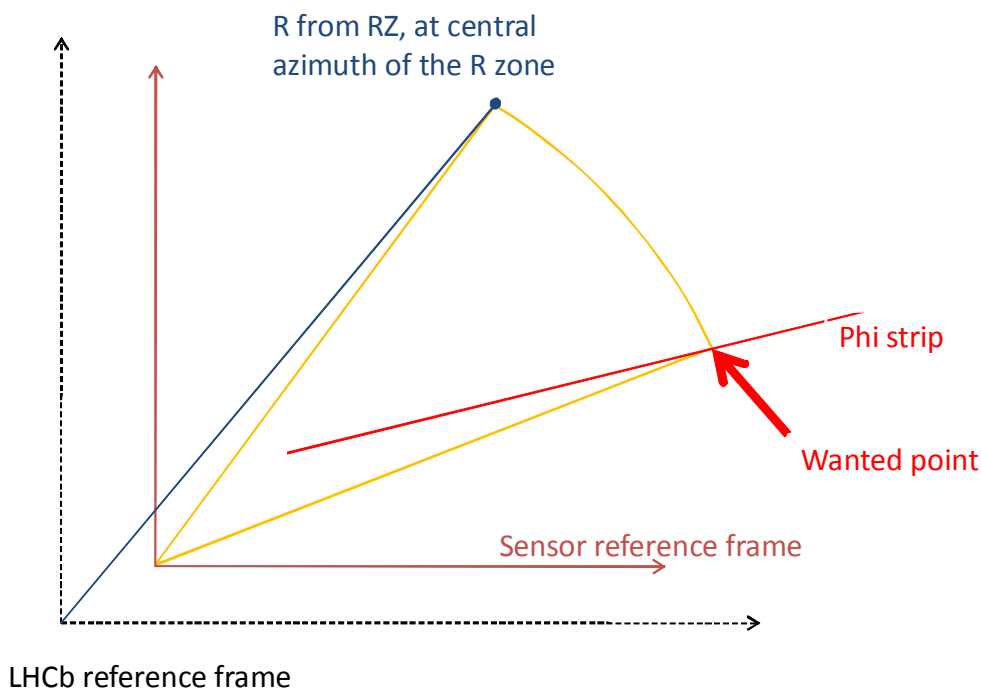Wanted point

Sensor reference frame

LHCb reference frame

Figure 4 Computation of the intersection point

For that, and as shown on Figure 4, we start from the blue line, i.e. the R value at the central azimuth of the R zone. We want the intersection of the circle, passing by this point and centred at the centre of the sensor, with the Phi strip. This is the intersection of a line and a circle, a second degree equation, with two solutions. However, as the Phi strips are almost radial, only one solution is in the proper R zone. A small technical detail: We work in Cartesian coordinates, and to avoid accuracy problems, the resolution is done in the X coordinate for zones close to the vertical, and in Y coordinate for zone near the horizontal.

For RZ tracks in the overlap, i.e. having R hits on both sides, the reference azimuth is the vertical line, and angular tolerances are reduced.

Before resolving the equation, a fast filtering of hits is applied, as we process all the hits of the two Phi sensors of the station: The projection of the strip's centre to the bisector should have a minimal value. This excludes all the opposite side strips plus part of the same side but far from the zone, at the cost of a simple scalar product. A second fast filter is that the distance of the starting point to the strip, measured by the linear form of the strip, is within a reasonable range. This range is defined as the radius times **PhiMatchZone** = 0.410, except for RZ tracks in the overlap, where the angular tolerance is the radius times **PhiCentralZone** = 0.040.

The selected hits should have their sine inside ±**PhiMatchZone** (or **PhiCentralZone** if in overlap) to be selected. The z position of the hit is adjusted as we now know a good approximation of the coordinate of the track. This sine is used as sorting variable inside each station. Stations with hits are counted, and the search range is restricted by ignoring the first and last stations if they have no hits.

3.4.2 Search for a triplet of hits

We start from the first station with hits, counting always in the opposite direction to the track propagation, i.e. for forward going track this is the last station along the track. Five cases are looked at for the 3 stations s1, s2 and s3:

0. Only unused hits on sensors   X X X
1. All hits on sensors   X X X
2. All hits on sensors   X X . X
3. All hits on sensors   X . X X
4. All hits on sensors   . X X X

When a track with a full length, i.e. as many Phi hits as R hits, is found at the end of a case, the loop on the five cases is aborted. For each hit in the first station s1, the hit with the closest 'distance' in s2 is selected, if below **MaxDelta2** = 0.05. The 'distance' is the absolute value of the slope perpendicular to the bisector plane, i.e. the ratio of the change of distance to the bisector to the difference in z. This is a measure of the fact that the track comes more or less from the beam line. With these two points (which are space points) one compute the straight line joining them in space. For all hits in the station s3 the χ^2 of the hit to the track is computed, and the smallest selected. This is the square of the distance, computed with the linear form, multiplied by the weight, inverse of the square of the error. If the smallest χ^2 chi2 is below 4 times **MaxChi2ToAdd** = 40, a good triplet is found.

The track is updated as space track adding the 3 hits; the Z position of the R hits is adjusted as we know now x and y; the track is fitted, and the highest χ^2 is computed and has to be below **MaxChi2PerHit** = 12 for the candidate to be kept. This cut is tighter than the previous one, because now we have the complete and exact geometry, while the computation of the distance used in the first selection has approximations if all sensors are not perfectly aligned.

If in overlap, hits from the other Phi sensor of the same stations are added if compatible, i.e. their χ^2 is lower than **MaxChi2ToAdd**. This is because, due to the dogleg shape of the sensor, there is some overlap between the two Phi sensors of a station, even more as their Z is not the same so the radius at which the track crosses them is different. Having as many tracks as possible with hits on the two sides of the Velo is a must for the alignment of the detector,

3.4.3 Extend the track

Phi hits are searched in the next stations, until two consecutive stations have no hits, or until we exit the selected range of stations. Hits should have a χ^2 contribution less than **MaxChi2ToAdd**, except if the z distance to the previous hit is larger than 140 mm. This is again to handle properly the extrapolation in the large gap shown on Figure 4. Similarly, if in the overlap, hits from the other sensor of the same station are added. A final fit is made, removing the worst Phi hit until its χ^2 contribution is less than **MaxChi2PerHit**. If there are not enough Phi hits after, the candidate is discarded. This minimal number of Phi hits is **FractionFound** = 0.70 times the number of stations. A quality factor is computed, this is the χ^2 per degree of freedom, plus twice the fraction of already used Phi hits. If the quality factor is higher than **MaxQFactor** = 6, the candidate is discarded. A test is performed to ensure that the track crosses the R sensor in the proper zone, at most one incompatible hit is allowed. A final check is made that there are both R and Phi hits on the same side of the Velo, i.e. that a track is not made with R on one half and Phi only on the other half. At this point, the track is kept and stored as space track. The loop continues for the next hits in s1, then for the next case except if a full length track has been found after case 1. There is no loop on the first sensor, as we know in which sensor the track should start, and case 4 recovers the case where the first sensor is missing.

3.4.4 Missed overlap tracks

If a RZ track in the overlap has not been upgraded as space track, a first recovery is performed by taking all the selected hits of all the stations, and trying to fit a track. For certain track, and with the alternated orientation of the Phi sensors, it may occur that a track has Phi hits only in every other station. The previous sequence will fail as there is no case like that. As the overlap has a small angular window, the hit multiplicity is small. The track is fitted, and hits removed until the highest χ^2 is below **MaxChi2PerHit**. It should be noted that the method to fit-and-remove first removes the worst hit in sensors with several hits, and only when all hits in multi-hit sensors have a good χ^2 it removes hits which are alone in their sensor. This helps in finding a track.

If this recovery fails also, this indicates that the 'overlap' track is in fact a coincidence of two tracks in the two halves having the same RZ projection. The input RZ track is then split in two RZ tracks, one with hits from the left half, one with hits from the right side, and these two tracks are space-extended as described before, as tracks not in the overlap.

3.4.5 Final cleanup

If several candidates are found for the same input RZ tracks, there is a possibility to have generated clones, even if the code tries to avoid it. A check is made to detect two space tracks sharing more than **FractionForMerge** = 0.70 of their Phi hits. If this is the case, the shortest track, or the one with the worse quality factor, is discarded.

Tracks with only 3 Phi hits, and where all these hits are already used, are also discarded.

If there are several candidates, a minimal number of Phi hits is computed: This is the largest number of Phi hits in the candidates, decremented by one if this number is higher than 4. The lowest quality factor is then searched within the track having enough phi hits. If the minimal length is higher than 3, then a tolerance of **DeltaQuality** = 0.5 is added to this minimal quality factor, to get the maximal acceptable quality for a track.

Tracks passing the criteria of length and quality are then filtered again: If the track has only 3 R and 3 Phi, the quality should be less than **MaxQFactor3** = 3. The track is refitted, eliminating R or Phi hits with a χ^2 greater than **MaxChi2PerHit**. If less than 6 hits are left, the track is discarded. Else, the track is stored as final track!

3.5 Phi tracking with unused hits

This search is performed only if the number of already found tracks is less than **MaxRZForExtra** = 200, if we have found already a number of space tracks higher than half the number of RZ tracks or if there are less than 20 RZ tracks. These tests are there to ensure that we will not have too high a combinatorics. Only forward tracks are searched for.

One searches for 3 R and 3 Phi hits in 3 consecutive stations making a segment. The seed sensor is taken in the last 8 stations, starting from the most downstream one. For each unused hit in s0, for each unused hit in s1 in a compatible ring (the track cannot go from the outside ring to the inside ring, while the reverse is true), a first compatibility check is made: The distance in the XY plane of the centre of strip in s0 to the strip in s1 should be less than **PhiUnusedPhiTol** = 5 mm. This tolerance is increased to 20 if the event is clean, i.e. has less than 10 RZ tracks. For each unused Phi hit in s3, the distance in the XY plane to the average (s0, s1) strip centre is computed and has to be less than **PhiUnusedSecondTol** = 10 mm, 20 mm for clean events. From these 3 Phi clusters, one computes the 45° zone in the first R sensor r0. Hits in this R zone compatible with the Phi ring are scanned, and for each a track candidate is built: There are 4 measurements, enough to compute a space line. The hit closest to this space line in r1 is selected, provided its χ^2 is below 400. The zone in this R sensor is selected according to the track extrapolation: This is how tracks crossing R zone boundaries can be found. The track

parameters are updated with this 5th hit, and the best hit in r2 is added, provided its χ^2 contribution is below **MaxChi2ToAdd**. If two or 3 of these R hits are already used, the candidate is ignored. The track is then extended, in R and Phi, until the extrapolation is outside the sensitive area, or until more than **MaxMissed** = 1 hits are missed. This is to recover tracks not found earlier, one can be quite strict! A full fit is performed, removing R and Phi clusters until the highest χ^2 contribution is below **MaxChi2PerHit**, and if there are 6 or more hits left, the track is stored and hits are tagged as used.

3.6 Final storage

Before converting the working tracks to final LHCb tracks, clone cleaning is performed on forward tracks. For each pair of tracks, the number of shared hits is computed. If it is higher than **FractionForMerge** = 0.70 times the shortest track, these tracks are clone candidates. The fraction of shared R hits is computed, if this is below **FractionForMerge** of the smallest number of R hits, both tracks are kept: Two space tracks sharing the same RZ were already compared as indicated in section 3.4.5 . Else the shortest one, or the one with worst quality factor, is removed.

Finally, the track is converted to an LHCb track, setting the proper parameters. A first state is provided at the point of closest approach to the beam line if **StateAtBeam** = true, else the first state is at first measurement. For forward tracks, a state is also provided at the end of the Velo, for extrapolation. Contrary to previous implementations, the track fit is not redone, as it was shown that the reweighting factor used previously generated no real improvement, while it had a cost in CPU.

3.7 Performance

The performance of the code is tested on Monte-Carlo simulation, where the efficiency and ghost rates are easy to measure. Tests were also done on real data, but there one needs a physics signal (e.g. J/Ψ→μμ) to measure the efficiency, and these signal events were selected using the previous version of the code. Tracks from a clean decay channel that weren't found by the old code can't be found by the new one: The events were not selected!

On 8000 event at ν = 3, which is close to the 2010 data taking conditions of μ≈2.5, the following performance is measured.

Item	Old code	FastVeloTracking	Difference
Ghost rate (track)	9.8 %	8.1 %	-1.7%
Efficiency long tracks	96.0 %	97.4 %	+1.4 %
Efficiency long tracks > 5 GeV	96.7 %	98.2 %	+1.5%
Eff. Long B daughters	96.1 %	97.2 %	+1.1%
Eff. Long B daughters > 5 GeV	96.8 %	98.1 %	+1.3%
Eff. long good B daughters	97.5 %	98.6 %	+1.1%
Eff. long good B daughters > 5 GeV	97.6 %	98.9 %	+1.3%
Eff. long Ks/Λ	87.8 %	90.4 %	+2.6%
Eff. long Ks/ Λ > 5 GeV	90.3 %	93.3 %	+3.0%
Clone rate all long tracks	1.7 %	1.7 %	-
Hit Purity all long tracks	99.4 %	99.6 %	+0.2%
Hit efficiency all long tracks	95.9 %	97.7 %	+1.8%
Total time per event	5.65 ms	3.77 ms	67%

Table 1 Performance of the FastVeloTracking

It is clear that there is a serious efficiency improvement, 1 to 1.5 %, with less ghost, better purity and efficiency, and in a time shorter by one third.

3.8 List of all properties of `FastVeloTracking`

Property name	Default value	Comment
OutputTrackName	"Rec/Track/Velo"	
OnlyForward	false	Don't search for backward tracks
OnlyBackward	false	Don't search for forward tracks
HLT1Only	false	Only quadruplets and space
HLT2Complement	false	Triplets and unused phi
MaxRZForExtra	200	Don't do unused phi if that many RZ tracks
StateAtBeam	true	
ResetUsedFlags	false	Clear used flags, for reprocessing the same event
ZVertexMin	-170 mm	
ZVertexMax	+120 mm	
ZVertexMaxBack	+1200 mm	Define the maximum angle for backward tracks
MaxRSlope	0.450	
rMatchTol4	1.0	
rMatchTol3	1.1	
rExtraTol	4.0	
rOverlapTol	1.0	
MaxMissed	1	
MinToTag	4	
PhiMatchZone	0.410	
PhiCentralZone	0.040	
MaxDelta2	0.05	
FractionFound	0.70	
MaxChi2PerHit	12.	
MaxChi2ToAdd	40.	
MaxQFactor	6.	
MaxQFactor3	3.	
DeltaQuality	0.5	
FractionForMerge	0.70	
PhiUnusedFirstTol	5.	
PhiUnusedSecondTol	10.	
DebugToolName	""	Use "PatVeloDebugTool" for debugging
WantedKey	-100	If ≥ 0, debug the MCParticle with that key
MeasureTime	false	Measure detailed timing

Table 2 List of job options for FastVeloTracking

4 Refitting the track: `FastVeloFitLHCbIDs`

In order to be able to reproduce the track fit in DaVinci, it is required to provide the same state on a track, starting from the list of LHCb Ids. This is implemented in this tool, which is used in the refit of tracks. Basically, it re-generates the **FastVeloTrack** and fits it, and then adds the states to the LHCb track.

When processing a track, the first step is to get the list of **FastVeloHit** from the LHCbID, using a dedicated method of the **FastVeloHitManager**. The R zone is computed by a majority vote of the R hits. The **FastVeloTrack** is built from the R clusters, the Phi clusters are added, R parameters updated and the track fitted. The track states are then created with the same code as in the original algorithm. It is assumed that the previous states have been removed from the track. It has been checked that this procedure give results essentially identical to the original ones. The source of difference is that the R parameters are updated from a different status of the track. These R parameters are essentially the equation of the tangent to the R strips once the space track is built. As in the original pattern code there can be extra hits that are

removed afterwards, the exact values are not guaranteed. But the differences are well below the accuracy of any measurement; the states have all their parameters equal to better than 10^{-3}.

5 Tracking when the Velo is open

It is important to be able to find primary vertices when the Velo is open, with the same algorithm as the HLT should not be reconfigured when the Velo is closed, this would lose precious luminosity. The **FastVeloTracking** code was tested in simulation and on real data with the Velo open. Compared to the previous code, the performance is better in all aspects.

Item	Old code	FastVeloTracking
Time per event (ms)	0.235	0.085
Ghost rate	0.8 %	0.3 %
Efficiency for long tracks	75.8 %	84.2 %
Efficiency for long tracks over 5 GeV	88.7 %	91.2 %

Table 3 Performace of FastVeloTracking with the Velo open

Tests inside the HLT for the Velo closing procedure are currently done by the expert, Malcolm John.

6 Tracking beam gas events

In the HLT, it is also important to select beam gas events, to measure the beam profile and normalize the luminosity. These events have vertices far from the luminous region. Due to the optics of the machine, the interesting vertices should not be too far, the zone ±1 meter around the nominal interaction point is enough. The use case in the HLT is to detect that an event has a good chance to have a beam gas interaction, and then run a dedicated instance of the Velo reconstruction with relaxed cuts (mainly the length of the luminous region) to get the maximal efficiency on these rare events, less than 1 percent of the interaction rate.

FastVeloTracking is efficient enough to get most of the tracks due to beam gas inside the standard reconstruction. These tracks come from the beam axis, so are found by the RZ tracking. The minimal track angle gives the maximal vertex distance. This is infinite for forward tracks as one accepts track parallel to the beam line. It is limited to **ZVertexMaxBack** = 1200 mm for backward tracks. Not all backward tracks (beam2-gas) are found with the standard setting, as we limit the range of Velo sensor looked at to the first few, up to the luminous region. But the code should find a few tracks, and then indicates that there is a potential beam2-gas interaction, thus triggering the execution of a dedicated version of **FastVeloTracking** looking at all backward tracks.

The exact implementation of this beam-gas processing in the HLT is still being worked out by the expert, Plamen Hopchev.

7 Summary

An entirely rewritten version of the Velo pattern recognition has been written and released in autumn 2010, and will be used for the 2011 run. It has better performance in all aspects, and supports Velo closing and beam gas search.

8 References

1. **Callot, Olivier.** *Velo tracking for the High Level Trigger.* LHCb 2003-027.

2. **Callot, Olivier**. *Online Pattern Recognition.* LHCb 2004-094.

3. **Hutchcroft, David.** *VELO Pattern Recognition.* LHCb 2007-013.