V0Finder

V0s

- In the release 01-00-00 there will be Λ in V0

- I would like to have an opinion from experts about the current strategy (is it the best one or can we do something different?)

- these slides: only cut and paste of the code, to have a starting point for the discussion

- For V0s: 2 main parts
  - V0Fitter
  - V0Finder (module that use the fitter)

V0FinderModule

software/tracking/modules/V0Finder/src/**V0FinderModule.cc**

```
129   // Group tracks into positive and negative tracks.
130   std::vector<const Track*> tracksPlus;
131   tracksPlus.reserve(nTracks);
132
133   std::vector<const Track*> tracksMinus;
134   tracksMinus.reserve(nTracks);
135
136   for (const auto& track : tracks) {
137     const TrackFitResult* tfr = track.getTrackFitResult(Const::pion);
138
139     if (!tfr) {
140       B2WARNING("No TrackFitResult for track");
141       continue;
142     }
143
144     const double charge = tfr->getChargeSign();
145     if (charge == +1) {
146       tracksPlus.push_back(&track);
147     } else {
148       tracksMinus.push_back(&track);
149     }
150   }
151
152   // Reject boring events.
153   if (tracksPlus.size() == 0 || tracksMinus.size() == 0) {
154     B2DEBUG(200, "No interesting tracks.");
155     return;
156   }
157
158   V0Fitter v0Fitter(m_TFRColName, m_V0ColName, m_V0ValidationVertexColName, m_RecoTrackColName);
159   v0Fitter.initializeCuts(m_beamPipeRadius, m_vertexChi2CutInside, m_massWindowKshortInside, m_vertexChi2CutOutside);
160   if (m_validation) {
161     v0Fitter.enableValidation();
162   }
```

1

2

3

1. loop on tracks
2. divided according to their sign
3. call the fitter

V0FinderModule

software/tracking/modules/V0Finder/src/**V0FinderModule.cc**

```
162   }
163
164   // Pair up each positive track with each negative track.
165   for (auto& trackPlus : tracksPlus) {
166     for (auto& trackMinus : tracksMinus) {
167       v0Fitter.fitAndStore(trackPlus, trackMinus, Const::Kshort);
168       v0Fitter.fitAndStore(trackPlus, trackMinus, Const::photon);
169     }
170   }
171 }
```

4

4. fitAndStore method from V0Fitter is called for different types of V0

V0Fitter

software/tracking/v0Finding/fitter/src/**V0Fitter.cc**

```
152 bool V0Fitter::fitAndStore(const Track* trackPlus, const Track* trackMinus,
153                            const Const::ParticleType& v0Hypothesis)  // TODO: Give Particle Types
154 {
155   const auto trackHypotheses = getTrackHypotheses(v0Hypothesis);
156
157   RecoTrack* recoTrackPlus = trackPlus->getRelated<RecoTrack>(m_RecoTrackColName);
158   if (not recoTrackPlus) {
159     B2ERROR("No RecoTrack for Belle2::Track");
160     return false;
161   }
162   genfit::Track& gfTrackPlus = RecoTrackGenfitAccess::getGenfitTrack(*recoTrackPlus);
163
164   RecoTrack* recoTrackMinus = trackMinus->getRelated<RecoTrack>(m_RecoTrackColName);
165   if (not recoTrackMinus) {
166     B2ERROR("No RecoTrack for Belle2::Track");
167     return false;
168   }
169   genfit::Track& gfTrackMinus = RecoTrackGenfitAccess::getGenfitTrack(*recoTrackMinus);
170
171   const genfit::MeasuredStateOnPlane* stPlusPtr = nullptr;
172   try {
173     stPlusPtr = &recoTrackPlus->getMeasuredStateOnPlaneFromFirstHit(
174                   TrackFitter::getTrackRepresentationForPDG(trackHypotheses.first.getPDGCode(), *recoTrackPlus));
175   } catch (genfit::Exception) {
176     B2DEBUG(100, "Hypotheses " << trackHypotheses.first.getPDGCode() << " not available. Taking default instead.");
177     try {
178       stPlusPtr = &recoTrackPlus->getMeasuredStateOnPlaneFromFirstHit(
179                     TrackFitter::getTrackRepresentationForPDG(Const::pion.getPDGCode(), *recoTrackPlus));
180     } catch (genfit::Exception) {
181       B2ERROR("Default track hypothesis not available. Should never happen, but I can continue savely anyway.");
182       return false;
183     }
```

1

2

3

1. Input: 2 track, 1 v0 hypothesis
2. (2x) Track → RecoTrack → genfit::Track
3. (2x) RecoTrack → MSOP

V0Fitter

software/tracking/v0Finding/fitter/src/**V0Fitter.cc**

```
152 bool V0Fitter::fitAndStore(const Track* trackPlus, const Track* trackMinus,
153                            const Const::ParticleType& v0Hypothesis)  // TODO: Give Particle Types
154 {
155   const auto trackHypotheses = getTrackHypotheses(v0Hypothesis);
```

```
135 std::pair<Const::ParticleType, Const::ParticleType> V0Fitter::getTrackHypotheses(const Const::ParticleType& v0Hypothesis)
136 {
137   if (v0Hypothesis == Const::Kshort) {
138     return std::make_pair(Const::pion, Const::pion);
139   } else if (v0Hypothesis == Const::photon) {
140     return std::make_pair(Const::electron, Const::electron);
141   } else if (v0Hypothesis == Const::Lambda) {
142     return std::make_pair(Const::proton, Const::pion);
143   } else if (v0Hypothesis == Const::antiLambda) {
144     return std::make_pair(Const::pion, Const::proton);
145   } else {
146     B2FATAL("Given V0Hypothesis not available.");
147     return std::make_pair(Const::invalidParticle, Const::invalidParticle);
148   }
149
150 }
```

V0Fitter

software/tracking/v0Finding/fitter/src/**V0Fitter.cc**

```
186  const genfit::MeasuredStateOnPlane* stMinusPtr = nullptr;
187  try {
188    stMinusPtr = &recoTrackMinus->getMeasuredStateOnPlaneFromFirstHit(
189                   TrackFitter::getTrackRepresentationForPDG(trackHypotheses.second.getPDGCode(), *recoTrackMinus));
190  } catch (genfit::Exception) {
191    B2DEBUG(100, "Hypotheses " << trackHypotheses.second.getPDGCode() << " not available. Taking default instead.");
192    try {
193      stMinusPtr = &recoTrackMinus->getMeasuredStateOnPlaneFromFirstHit(
194                     TrackFitter::getTrackRepresentationForPDG(Const::pion.getPDGCode(), *recoTrackMinus));
195    } catch (genfit::Exception) {
196      B2ERROR("Default track hypothesis not available. Should never happen, but I can continue savely anyway.");
197      return false;
198    }
199  }
200
201  if (not stPlusPtr or not stMinusPtr) {
202    return false;
203  }
204
205  genfit::MeasuredStateOnPlane stPlus = *stPlusPtr;
206  genfit::MeasuredStateOnPlane stMinus = *stMinusPtr;
207
208  if (rejectCandidate(stPlus, stMinus)) {
209    return false;
210  }
211
212  genfit::GFRaveVertex vert;
213  if (not fitVertex(gfTrackPlus, gfTrackMinus, vert)) {
214    return false;
215  }
```

3

4

4. 2x genfit::Track → vertex (GFRaveVertex)

V0Fitter

software/tracking/v0Finding/fitter/src/**V0Fitter.cc**

```
217   const genfit::GFRaveTrackParameters* tr0 = vert.getParameters(0);
218   const genfit::GFRaveTrackParameters* tr1 = vert.getParameters(1);
219
220   const TVector3& posVert(vert.getPos());
221   TLorentzVector lv0, lv1;
222
223   // Reconstruct invariant mass.
224   lv0.SetVectM(tr0->getMom(), trackHypotheses.first.getMass());
225   lv1.SetVectM(tr1->getMom(), trackHypotheses.second.getMass());
226
```

5

```
227   // Apply cuts.  We have one set of cuts inside the beam pipe,
228   // the other outside.
229   if (posVert.Perp() < m_beamPipeRadius) {
230     if (v0Hypothesis == Const::photon) {
231       return false;  // No converions inside beampipe
232     }
233
234     if (vert.getChi2() > m_vertexChi2CutInside) {
235       B2DEBUG(200, "Vertex inside beam pipe, chi^2 too large.");
236       return false;
237     }
238
239     const double mReco = (lv0 + lv1).M();
240     if (v0Hypothesis == Const::Kshort and fabs(mReco - Const::K0Mass) > m_massWindowKshortInside * Unit::MeV) {
241       B2DEBUG(200, "Vertex inside beam pipe, outside Kshort mass window.");
242       return false;
243     }
244   } else {
245     if (vert.getChi2() > m_vertexChi2CutOutside) {
246       B2DEBUG(200, "Vertex outside beam pipe, chi^2 too large.");
247       return false;
248     }
```

6

5. vertex → GFRaveTrackParameters (used only in the V0Validation)

6. cuts

V0Fitter

software/tracking/v0Finding/fitter/src/**V0Fitter.cc**

```
251   B2DEBUG(200, "Vertex accepted.");
252
253   // Assemble V0s.
254   if (not extrapolateToVertex(stPlus, stMinus, posVert)) {
255     return false;
256   }
257
258   const double Bz = getBzAtVertex(posVert);
259
260   TrackFitResult* tfrPlusVtx = buildTrackFitResult(gfTrackPlus, stPlus, Bz, trackHypotheses.first);
261   TrackFitResult* tfrMinusVtx = buildTrackFitResult(gfTrackMinus, stMinus, Bz, trackHypotheses.second);
262
263   B2DEBUG(100, "Creating new V0.");
264   m_v0s.appendNew(std::make_pair(trackPlus, tfrPlusVtx),
265                   std::make_pair(trackMinus, tfrMinusVtx));
266
267   if (m_validation) {
268     B2DEBUG(300, "Create StoreArray and Output for validation.");
269     m_validationV0s.appendNew(
270       std::make_pair(trackPlus, tfrPlusVtx),
271       std::make_pair(trackMinus, tfrMinusVtx),
272       vert.getPos(),
273       vert.getCov(),
274       (lv0 + lv1).P(),
275       (lv0 + lv1).M(),
276       vert.getChi2()
277     );
278
279   }
280   return true;
```

7

8

9

7. (2x) new TFR (using Track Hypothesis)
8. V0
9. V0Validation

V0s

- Problem I found:

  (from slide 4) since the “fitAndStore” method is called for each different V0 type, the same vertex is saved more than one time

- Does the mass fit hypothesis enter in the vertex fit? Where? (it is not so clear to me if it is included somehow in the GFRaveVertex/ genfit::Track)

- Is it useful to call the method more than once? (4 times, considering also Λ and anti-Λ)
  Is there a more efficient way to proceed?

Backup

V0fitter

software/tracking/v0Finding/fitter/src/**V0Fitter.cc**

```
62 bool V0Fitter::fitVertex(genfit::Track& trackPlus, genfit::Track& trackMinus, genfit::GFRaveVertex& vertex)
63 {
64   VertexVector vertexVector;
65   std::vector<genfit::Track*> trackPair {&trackPlus, &trackMinus};
66
67
68   try {
69     IOIntercept::OutputToLogMessages
70     logCapture("V0Fitter GFRaveVertexFactory", LogConfig::c_Debug, LogConfig::c_Debug);
71     logCapture.start();
72
73     genfit::GFRaveVertexFactory vertexFactory;
74     vertexFactory.findVertices(&vertexVector.v, trackPair);
75   } catch (...) {
76     B2ERROR("Exception during vertex fit.");
77     return false;
78   }
79
80   if (vertexVector.size() != 1) {
81     B2DEBUG(150, "Vertex fit failed. Size of vertexVector not 1, but: " << vertexVector.size());
82     return false;
83   }
84
85   if ((*vertexVector[0]).getNTracks() != 2) {
86     B2DEBUG(100, "Wrong number of tracks in vertex.");
87     return false;
88   }
89
90   vertex = *vertexVector[0];
91   return true;
92 }
```

Outline

software/genfit2/code2/GFRave/src/**GFRaveVertexFactory.cc**

```
66  void
67  GFRaveVertexFactory::findVertices ( std::vector <  genfit::GFRaveVertex* > * GFvertices,
68      const std::vector < genfit::Track* > & GFTracks,
69      bool use_beamspot ){
70
71    clearMap();
72
73    try{
74      RaveToGFVertices(GFvertices,
75                       factory_->create(GFTracksToTracks(GFTracks, nullptr, IdGFTrackStateMap_, 0),
76                                        use_beamspot),
77                       IdGFTrackStateMap_);
78    }
79    catch(Exception & e){
80      clearMap();
81       std::cerr << e.what();
82    }
83
84    clearMap();
85  }
```

software/genfit2/code2/GFRave/src/**GFRaveConverters.cc**

```
209 void
210 RaveToGFVertices(std::vector<GFRaveVertex*> * GFVertices,
211     const std::vector<rave::Vertex> & raveVertices,
212     const std::map<int, genfit::trackAndState>& IdGFTrackStateMap){
213
214   unsigned int nVert(raveVertices.size());
215
216   GFVertices->reserve(nVert);
217
218   for (unsigned int i=0; i<nVert; ++i){
219     GFVertices->push_back(RaveToGFVertex(raveVertices[i], IdGFTrackStateMap));
220   }
221 }
```

Outline

software/genfit2/code2/GFRave/src/**GFRaveConverters.cc**

```
140 GFRaveVertex*
141 RaveToGFVertex(const rave::Vertex & raveVertex,
142     const std::map<int, genfit::trackAndState>& IdGFTrackStateMap){
143
144   if (!(raveVertex.isValid())) {
145     Exception exc("RaveToGFVertex ==> rave Vertex is not valid!",__LINE__,__FILE__);
146     throw exc;
147   }
148
149   std::vector < std::pair < float, rave::Track > > raveWeightedTracks(raveVertex.weightedTracks());
150   std::vector < std::pair < float, rave::Track > > raveSmoothedTracks(raveVertex.weightedRefittedTracks());
151
152   int id;
153   unsigned int nTrks(raveWeightedTracks.size());
154
155   // check if rave vertex has  refitted tracks
156   bool smoothing(true);
157   if (! (raveVertex.hasRefittedTracks()) ) {
158     smoothing = false;
159   }
160
161   // check numbers of tracks and smoothed tracks
162   if (smoothing && nTrks != raveSmoothedTracks.size()){
163     Exception exc("RaveToGFVertex ==> number of smoothed tracks != number of tracks",__LINE__,__FILE__);
164     throw exc;
165   }
166
167   // (smoothed) track parameters
168   std::vector < GFRaveTrackParameters* > trackParameters;
169   trackParameters.reserve(nTrks);
170
```

Outline

software/genfit2/code2/GFRave/src/**GFRaveConverters.cc**

```
171   // convert tracks
172   for (unsigned int i=0; i<nTrks; ++i){
173     id = raveWeightedTracks[i].second.id();
174
175     if (IdGFTrackStateMap.count(id) == 0){
176       Exception exc("RaveToGFVertex ==> rave track id is not present in IdGFTrackStateMap",__LINE__,__FILE__);
177       throw exc;
178     }
179
180     GFRaveTrackParameters* trackparams;
181
182     if(smoothing) {
183       // convert smoothed track parameters
184       trackparams = new GFRaveTrackParameters(IdGFTrackStateMap.at(id).track_, //track
185           IdGFTrackStateMap.at(id).state_, //state
186           raveWeightedTracks[i].first, //weight
187           Vector6DToTVectorD(raveSmoothedTracks[i].second.state()), //smoothed state
188           Covariance6DToTMatrixDSym(raveSmoothedTracks[i].second.error()), //smoothed cov
189           true);
190     }
191     else {
192       // convert track parameters, no smoothed tracks available
193       trackparams = new GFRaveTrackParameters(IdGFTrackStateMap.at(id).track_, //track
194           IdGFTrackStateMap.at(id).state_, //state
195           raveWeightedTracks[i].first, //weight
196           Vector6DToTVectorD(raveWeightedTracks[i].second.state()), //state
197           Covariance6DToTMatrixDSym(raveWeightedTracks[i].second.error()), //cov
198           false);
199     }
200     trackParameters.push_back(trackparams);
201   }
202
203   return new GFRaveVertex(Point3DToTVector3(raveVertex.position()),
204                           Covariance3DToTMatrixDSym(raveVertex.error()),
205                           trackParameters,
206                           raveVertex.ndf(), raveVertex.chiSquared(), raveVertex.id());
207 }
```