Bachelorarbeit

V0 Decays: Documentation of the C++ Program *AliESDv0KineCuts.cxx*

V0-Zerfälle: Dokumentation des C++-Programms *AliESDv0KineCuts.cxx*

David Baumeier

2011

Bachelorarbeit

im

Institut für Kernphysik

V0 Decays: Documentation of the C++ Program *AliESDv0KineCuts.cxx*

V0-Zerfälle: Dokumentation des C++-Programms *AliESDv0KineCuts.cxx*

David Baumeier

12.05.2011 bis 01.09.2011

Erster Gutachter: Prof. Dr. Johannes P. Wessels

Zweiter Gutachter: Dr. Christian Klein-Bösing

Contents

1 Introduction **1**

2 Standard Model of particle physics **3**

3 V^0 particles and decays **5**

3.1 What are V^0 particles? . . . 5

3.1.1 Overview about V^0 decays . . . 5

3.1.2 Weak interaction . . . 7

3.1.3 γ conversion . . . 9

4 Identification of V^0s **11**

4.1 V^0 particles as a tool . . . 11

4.2 Reconstruction of the daughter tracks . . . 11

4.2.1 Vertex . . . 12

4.2.2 Pointing angle . . . 12

4.2.3 ξ_{pair} and Ψ_{pair} . . . 12

4.2.4 Invariant mass . . . 13

4.2.5 χ^2 / NDF . . . 13

4.3 Monte Carlo simulation . . . 14

5 Design of the V^0 analysis software **15**

5.1 How to use . . . 15

5.1.1 An example . . . 15

5.1.2 User execution . . . 17

5.2 General overview . . . 18

5.3 Initialization . . . 19

5.4 V^0 common cuts . . . 20

5.5 Single track cuts . . . 21

5.6 Armenteros preselection . . . 21

5.7 Decision . . . 25

6 Summary **30**

References **31**

A Appendix **33**

A.1 Figures 33

A.2 Code excerpts 36

List of Figures

1 AliRoot user interface . . . 2
2 Two examples of V^0 decays in a bubble chamber . . . 6
3 Weak decay of K^0_S, $\overline{K^0_S}$, Λ^0, and $\overline{\Lambda^0}$ illustrated in the Feynman diagram . . . 9
4 Topology of a V^0 decay . . . 11
5 Illustration of the angles ξ_{pair} and Ψ_{pair} . . . 12
6 ROOT Object Browser . . . 16
7 Scheme of the implementation . . . 18
8 Armenteros plot of all found V^0 candidates with different number of entries . . . 22
9 Armenteros plot with applied cuts . . . 23
A.1 Plots of the DCA of the V^0 daughter tracks . . . 33
A.2 Plots of the radius of the vertex of the V^0 daughter tracks . . . 34
A.3 Plots of the invariant mass of the V^0 daughter tracks . . . 35

List of Tables

1 Organization of fermions . . . 3
2 List of interaction types and their properties . . . 4
3 V^0 particles with their branching ratio, their daughter particles, and the daughters' daughter particles . . . 7
4 Attributes of V^0 particles . . . 8
5 Weak decay in quark notation and hadron notation . . . 9
6 Default skip values for the cuts . . . 19

List of C++ code extracts

1 runV0task . . . 15
2 Main user function: ProcessV0 . . . 17
3 Example values for cuts . . . 23
4 Differentiation of V^0 cases . . . 24
5 γ verification: Check sign of the charge . . . 25
6 γ verification: Check the daughter tracks . . . 25
7 γ verification: Apply the cuts . . . 26

8 γ verification: All cuts passed 26
9 K^0_S verification: All cuts passed 26
10 Creating the mother particle of $p^{\pm}$ and $\pi^{\mp}$ 27
11 Checking momentum and mass of daughter particles 28
12 Λ^0 or $\overline{\Lambda^0}$ verification: All cuts passed 28
13 Default constructor 36
14 Method `SingleTrackCuts` 37
15 Method `V0CutsCommon` 38
16 Method `Armenteros` 39
17 Method `CreateMotherParticle` 40
18 Method `PsiPair` 41
19 Method `GetConvPosXY` 43
20 Method `GetHelixCenter` 44

1. Introduction

My physics teacher in school told me once: Physics is the fundamental natural science. In fact, it asks questions about nature and tries to find out how nature works. Physics ranges from the huge size of the universe in astrophysics to the incredibly small size of elementary particles in nuclear physics. There are two extreme dimensions of 10^9 light years ($\approx 10^{25}$ meters) and above and 10^{-15} meters and below.
Nuclear physics is engaged in atomic nuclei and their constituents with the aid of particle accelerators in which various particles are colliding. Emerging particles can be analysed and hypothesised particles can be corroborated with detectors and computer programs.
Those particles are summed up in the so-called Standard Model of particle physics.

One research establishment is CERN[1] in Geneva, Switzerland. A ring-shaped particle accelerator, the **L**arge **H**adron **C**ollider (LHC), belongs to CERN, at which six detector experiments are installed. One of them is ALICE[2]. ALICE is an acronym for "**A** **L**arge **I**on **C**ollider **E**xperiment". ALICE is optimized to study heavy ion collisions, for example Pb-Pb.
The ALICE TPC (abbreviation for ***T**ime **P**rojection **C**hamber*) is the "main device for tracking of charged particles and particle identification" [TPC1]. It enables the reconstruction of a three-dimensional picture of tracks from charged particles [TPC2].
The ALICE TPC at CERN is the world's largest TPC with a radius of 2.5 m, a length of about 5 m, and a volume of 88 m^3.

The detector which is closest to the collision point is the ITS (abbreviation for ***I**nner **T**racking **S**ystem*). It detects the formation of particle showers which arise shortly after a collision. Besides, it shall locate and identify low momentum particles together with the TPC [ITS].

This bachelor thesis deals with the function of a C++ program which is used for example at CERN to analyse a certain group of particles, so-called $V0$ or V^0 particles[3], and can be used by people in ALICE, as an overview, manual, and documentation of the program `AliESDv0KineCuts.cxx`.

[1] originally for "**C**onseil **E**uropéen pour la **R**echerche **N**ucléaire" (European Council for Nuclear Research), today: "European Organization for Nuclear Research"
[2] besides ATLAS, CMS, TOTEM, LHCb, and LHCf
[3] Both spellings are common. I personally prefer and use the spelling V^0 in my thesis.

The implementation of the program takes place in AliRoot (see figure 1), which is based on object-oriented techniques for programming and on the ROOT system, which was developed by CERN.
The AliRoot framework is used for simulation, alignment, calibration, reconstruction, visualisation, and analysis of the experimental data [Gus].

V^0 names the secondary vertices from certain particle decays. The mother particles of these decays are called V^0 particles. In our case, the analysis of these particles is not a physical problem, but is primarily a question of gaining pure samples of electrons, pions, and protons, which arise from V^0 decays. For this purpose, Dr. Matus Kalisky developed a method in AliRoot, `AliESDv0KineCuts.cxx` [AOff], which identifies the V^0 tracks with purely kinematical cuts.

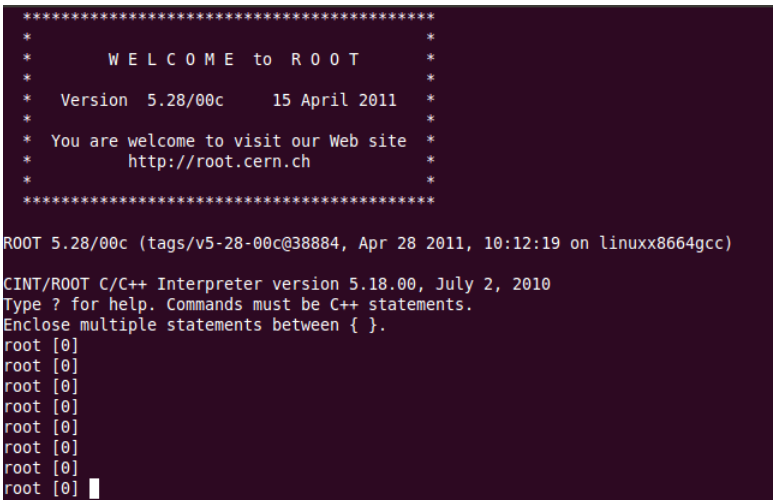


Figure 1 *– AliRoot user interface after the login*

The structure of my bachelor thesis is the following. Chapter 2 explains the Standard Model of particle physics. Chapter 3 deals with the explanation of the V^0 particles. The topology of a V^0 decay is illustrated in chapter 4. The main part of this thesis is chapter 5, which explains the C++ program in detail.
The objective is to describe the program `AliESDv0KineCuts.cxx` so precisely that someone who works with the program can understand it without any difficulty.

2. Standard Model of particle physics

The Standard Model of particle physics is a theory of all known particles and their interaction amongst each other [SM].
It contains many different elementary particles: six quarks, six leptons, and gauge bosons. Particles with half-integer spin ($\frac{1}{2}$, $\frac{3}{2}$, $\frac{5}{2}$, ...) are called fermions, including leptons, quarks, and all baryons. Baryons (or anti-baryons) consist of three quarks (or three antiquarks) like protons (*uud*) and neutrons (*udd*). A quark-quark pair generates so-called mesons. Unlike baryons, mesons have an integer spin. Therefore they belong to the bosons. Altogether, particles composed of quarks are called hadrons.

	Charge	First generation	Second generation	Third generation
Quarks	+ $\frac{2}{3}$	Up (u)	Charm (c)	Top (t)
	- $\frac{1}{3}$	Down (d)	Strange (s)	Bottom (b)
Leptons	- 1	Electron (e^-)	Muon (μ^-)	Tauon (τ^-)
	0	Electron neutrino (ν_e)	Muon neutrino (ν_μ)	Tauon neutrino (ν_τ)

***Table 1** – Organization of fermions*

Quarks are color charged particles. In analogy to the additive color model, there are six color charges: red, blue, green for particles as well as anti-red, anti-blue, anti-green for antiparticles. The addition of three colors or three anticolors as well as the addition of one color-anticolor pair result in white, color charge neutral. Color charged particles cannot be isolated singularly. They appear always in bound states with white color charge. This phenomenon is called confinement.

The gauge bosons are the carriers of the fundamental interactions of nature. Beyond the Standard Model, there are new gauge bosons needed, for example the graviton[4], which should be responsible for gravitation. There are three kinds of gauge bosons within the Standard Model, which are listed in table 2 with their corresponding interaction.

[4]It is a hypothetical particle. Experiments could not confirm its existence yet.

interaction	**gauge boson**	**interaction between**	**range** (m)	**relative strength**	**typical life time** (s)
strong	8 gluons	quarks	10^{-15}	1	10^{-23}
weak	W^+-, W^--, Z^0-boson	quarks, leptons	10^{-18}	10^{-5}	10^{-8}
electro-magnetic	photon	quarks, leptons (without neutrinos)	∞	10^{-2}	10^{-20}

Table 2 *– List of interactions types and their properties, [Kol]*

There are three types of interaction:

- **strong interaction**

  The strong interaction is responsible for the binding between nucleons in the atomic nucleus and between quarks in hadrons (confinement).

- **weak interaction**

  The weak interaction takes effect between quarks and leptons as well as antiquarks and antileptons. Moreover it arranges for the exchange of energy and momentum and takes part in weak decays (more on that in chapter 3).

- **electromagnetic interaction**

  The electromagnetic interaction is responsible for light, electricity, magnetism, chemistry, and several sorts of solid state properties.

Though gravity is a fundamental interaction, it is not implemented in the quantum field theory yet.

3. V^0 particles and decays

3.1. What are V^0 particles?

In the 1940s and 1950s, scientists published many papers about a new group of particles. One article from Leighton, Wanlass and Anderson, published in the scientific journal *Physical Review* in 1953, showed several photos of bubble chambers, where decays of those particles were visible due to the appearance of two tracks at the same point. Two photos are shown in figure 2.

3.1.1. Overview about V^0 decays

These particles are some heavy, unstable subatomic particles, which decay into a pair of daughter particles. In a bubble chamber or another particle detector, the tracks of the decay products can be seen as the characteristic letter V. Since the particles are charge neutral, they are called V^0 particles.

The most frequent particles are: Λ^0, $\overline{\Lambda^0}$, K_S^0, γ. In the strict sense, γ is not a V^0 particle, because it does not decay weakly, but it converts into new particles. Here it is treated in the same way as the others because of its similar behaviour. The detector is not able to detect the V^0 particles themselves because of their short life time (see table 4) on the one hand. And on the other hand V^0s are charge neutral, so they do not interact electromagnetically. Instead of that, we detect their daughter particles which are charged and stable or at least long-living enough to hit the detector.

The Λ^0 particle decays into p^+ and π^- while $\overline{\Lambda^0}$ decays into p^- and π^+. A K_S^0 as a mother particle decays into a π^+ and a π^-. There may also be $\overline{K_S^0}$, which has the same daughter particles as K_S^0, so after the detection of the daughter particles, it is not possible to decide whether there was a K_S^0 or a $\overline{K_S^0}$ decay. There is one more possibility for K_S^0 and Λ^0 to decay. 30.7 % of the K_S^0 decays and 35.8 % of the Λ^0 decays create two π^0. Therefore those decays are not very interesting, because the program cannot determine the mother particle (either Λ^0 or K_S^0). The last mother particle is the γ, which nearly solely converts into an e^- and an e^+.

Table 4 shows an overview about some attributes of the mother and the daughter particles.

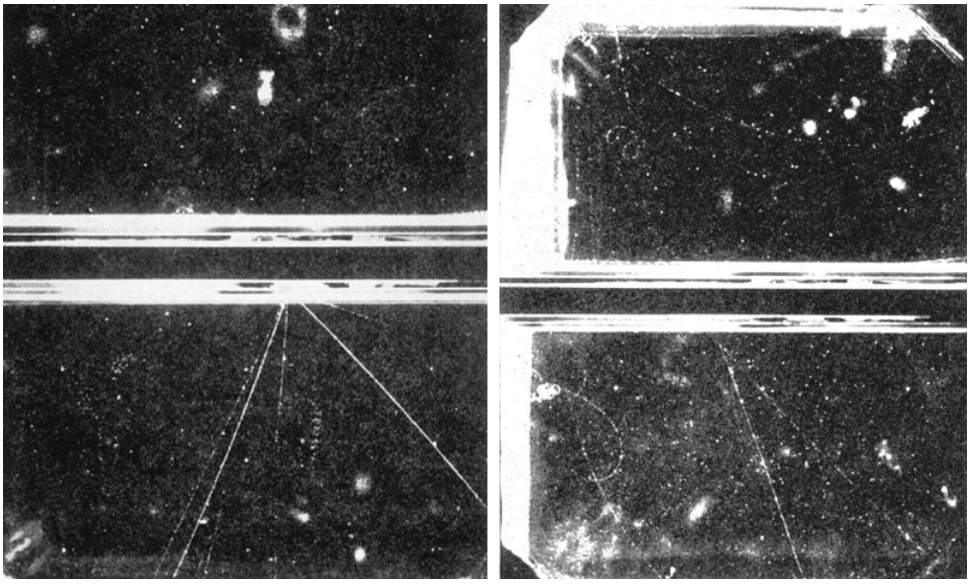


Figure 2 – *Two examples of V^0 decays visualized in a bubble chamber:*

Left picture:

"Example of the production of a V^0 particle by a single charged particle. The initiating particle enters the lead plate between the chambers almost vertically from above and slightly to the right of the center of the chamber. In the lead plate it undergoes a nuclear collision, from which six charged particles, the V^0 particle, and an unknown number of neutral particles emerge. The V^0 particle decays near the bottom of the lower chamber into a heavily-ionizing particle and a negative particle of about minimum ionization. These products are probably a proton and a π^- meson, respectively. The shortness of the tracks does not permit accurate measurements of the momenta."

Right picture:

"The decay occurs near the top center of the lower chamber. The positive particle is heavily ionizing, and moves downward and slightly toward the right from the decay point, while the negative particle is near minimum ionization and travels diagonally downward and to the right from the decay point.", [LWA]

To generate the mother particle like K^0_S or Λ^0, a proton beam or a lead beam hits another beam. This process generates some more particles but the program only looks at the V^0 particles. The interaction between the beam and the target or, in our cases, between colliding protons causes the emission of hadrons.

γ-rays and their interaction with matter are the reason for the appearance of e^+ and e^- via pair production.

mother particle	daughter particles and branching ratio	
Λ^0, $\overline{\Lambda^0}$	$p^+ \pi^-$, $p^- \pi^+$	$\pi^0 \pi^0$
	(63.9 ± 0.5) %	(35.8 ± 0.5) %
K_S^0, $\overline{K_S^0}$	$\pi^+ \pi^-$	$\pi^0 \pi^0$
	(69.20 ± 0.05) %	(30.69 ± 0.05) %
γ	$e^+ e^-$ (in matter)	
	$\sim 100.0\%$ (via pair production)	
π^0	$\gamma \gamma$	$e^+ e^- \gamma$
	(98.823 ± 0.034) %	(1.174 ± 0.035) %
π^+	$\mu^+ \nu_\mu$	$e^+ \nu_e$
	(99.98770 ± 0.00004) %	$(1.230 \pm 0.004) \cdot 10^{-4}$ %
π^-	$\mu^- \overline{\nu}_\mu$	$e^- \overline{\nu}_e$
	(99.98770 ± 0.00004) %	$(1.230 \pm 0.004) \cdot 10^{-4}$ %

Table 3 *– V^0 particles with their branching ratio, their daughter particles, and the daughters' daughter particles, [PDG1]*

Table 3 gives a short overview about the most important V^0 decays and γ conversion.

3.1.2. Weak interaction

A decay or a conversion of a particle occurs via the weak interaction. There are two different types of weak interaction, which are distinguished by the type of the mediating boson. The W^+- and W^--boson refer to the *charged current interaction* while the Z-boson refers to the *neutral current interaction*.

The short range of the weak interaction is based on the heavy mass of the bosons ((80.399 ± 0.023) GeV/c^2 for W^+ and W^-, (91.1876 ± 0.0021) GeV/c^2 for Z^0, [PDG3]). The neutral V^0 particles decay into one positively charged and one negatively charged particle. Since a boson "moves" the charge to convert the neutral charge into a positive and a negative part, it is the charged current interaction.

Such a decay can be written in two different ways: in quark notation or hadron notation. The quark notation only refers to the quark which changes, while the hadron notation describes

particle name	quark content	particle ID	rest mass (MeV/c^2)	mean life time (s)	max. reach ($c \cdot \tau$) (cm)
Lambda (Λ^0)	uds	3122	$1,115.683 \pm 0.006$	$(2.631 \pm 0.020) \cdot 10^{-10}$	7.9
Antilambda ($\overline{\Lambda^0}$)	$\overline{u}\overline{d}\overline{s}$	-3122	$1,115.683 \pm 0.006$	$(2.631 \pm 0.020) \cdot 10^{-10}$	7.9
Kaon (K_S^0)	$\frac{d\overline{s}-s\overline{d}}{\sqrt{2}}$	310	497.614 ± 0.024	$(8.953 \pm 0.005) \cdot 10^{-11}$	2.7
Antikaon ($\overline{K_S^0}$)	$\frac{s\overline{d}-d\overline{s}}{\sqrt{2}}$	-310	497.614 ± 0.024	$(8.953 \pm 0.005) \cdot 10^{-11}$	2.7
Gamma (γ)	-	22	0	varying	∞
Proton (p^+)	uud	2212	$938.272 \pm 2.3 \cdot 10^{-5}$	stable	∞
Antiproton (p^-)	$\overline{u}\overline{u}\overline{d}$	-2212	$938.272 \pm 2.3 \cdot 10^{-5}$	stable	∞
Pion (neutral) (π^0)	$\frac{u\overline{u}-d\overline{d}}{\sqrt{2}}$	111	$134.977 \pm 6.0 \cdot 10^{-4}$	$(8.4 \pm 0.6) \cdot 10^{-17}$	$2.5 \cdot 10^{-6}$
Pion (π^+)	$u\overline{d}$	211	$139.570 \pm 3.5 \cdot 10^{-4}$	$(2.6033 \pm 0.0005) \cdot 10^{-8}$	780
Antipion (π^-)	$\overline{u}d$	-211	$139.570 \pm 3.5 \cdot 10^{-4}$	$(2.6033 \pm 0.0005) \cdot 10^{-8}$	780
Electron (e^-)	-	11	$0.5110 \pm 1.3 \cdot 10^{-11}$	stable	∞
Positron (e^+)	-	-11	$0.5110 \pm 1.3 \cdot 10^{-11}$	stable	∞

Table 4 – *Attributes of mother and daughter particles in the V^0 decays. The particle ID is given by the PDG (particle data group). The minus sign marks the antiparticles. These numbers are used in the program, [PDG2]*

all particles in this process (see table 5).

particle	quark notation	hadron notation
Λ^0	$s^{-\frac{1}{3}} \rightarrow u^{+\frac{2}{3}} + d^{-\frac{1}{3}} + \overline{u}^{-\frac{2}{3}}$	$\Lambda^0 \rightarrow p^+ + \pi^-$
$\overline{\Lambda^0}$	$\overline{s}^{+\frac{1}{3}} \rightarrow \overline{u}^{-\frac{2}{3}} + \overline{d}^{+\frac{1}{3}} + u^{+\frac{2}{3}}$	$\overline{\Lambda^0} \rightarrow p^- + \pi^+$
K_S^0	$\overline{s}^{+\frac{1}{3}} \rightarrow \overline{u}^{-\frac{2}{3}} + \overline{d}^{+\frac{1}{3}} + u^{+\frac{2}{3}}$	$K_S^0 \rightarrow \pi^+ + \pi^-$
$\overline{K_S^0}$	$s^{-\frac{1}{3}} \rightarrow u^{+\frac{2}{3}} + d^{-\frac{1}{3}} + \overline{u}^{-\frac{2}{3}}$	$\overline{K_S^0} \rightarrow \pi^+ + \pi^-$

Table 5 *– Weak decay in quark notation and hadron notation*

Besides the written reaction equations, the decays can be illustrated by the Feynman diagrams (see figure 3).

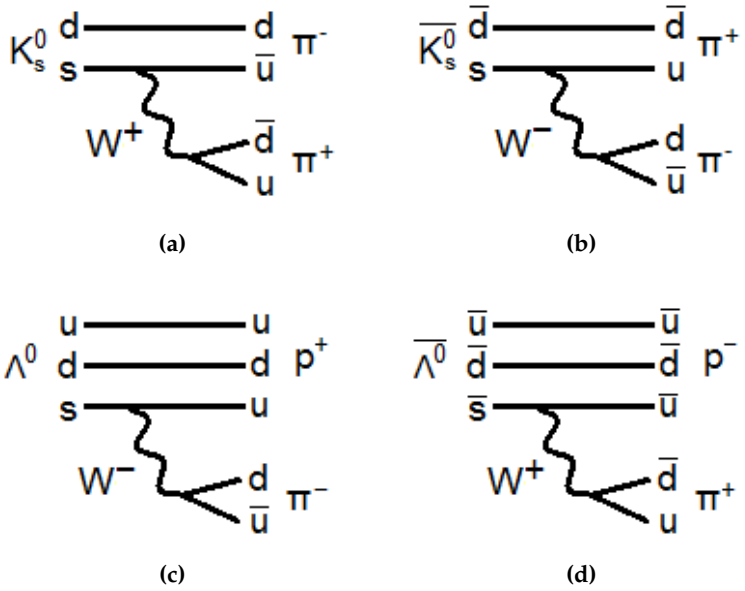


Figure 3 *– Weak decays illustrated in the Feynman diagram:*
(a) K_S^0*, (b)* $\overline{K_S^0}$*, (c)* Λ^0*, (d)* $\overline{\Lambda^0}$

Due to the quark transformation, caused by the weak decay, the V^0 particles have a long life time compared to other instable particles (see table 2 for reference values).

3.1.3. γ conversion

Pair production is a process where a particle and its antiparticle come into existence. The necessary energy of the mother photon must be at least the sum of the rest mass energy

of the produced particles. The best-known pair production is the conversion of a γ into an electron-positron pair:

$$\gamma \rightarrow e^- + e^+$$

The γ quantum must reach an energy of at least 1022 keV while hitting any material for the production of an electron-positron pair ($2 \cdot 511$ keV). Excessive energy converts into kinetic energy of the produced electron and positron.
The reverse process, when a particle comes into contact with its antiparticle, for example

$$e^- + e^+ \rightarrow \gamma + \gamma$$

is called “pair annihilation”.

4. Identification of V^0s

4.1. V^0 particles as a tool

The particle identification (PID) of the daughter particles of the V^0 decays is important for several applications. Thus it is possible to gain a pure sample of protons, electrons, pions, and their corresponding antiparticles, which can be used for calibration purposes.
To gain pure particles, the PID software has to analyse the detected particles for different properties.

4.2. Reconstruction of the daughter tracks

The reconstruction of the tracks of the daughter particles of the V^0s is already part of the identification. It is done by cutting on the following values [AKal]:

- "The distance of the daughter tracks to the primary vertex."
- "The distance of closest approach (DCA) between the daughter tracks."
- "The pointing angle θ. The momentum of the mother particle should point to the primary vertex."
- "A causality check is performed. This means the absence of space points in forbidden ITS layers can be required if the decay takes place far enough from the vertex."

Figure 4 shows the topology of a V^0 decay with important variables.

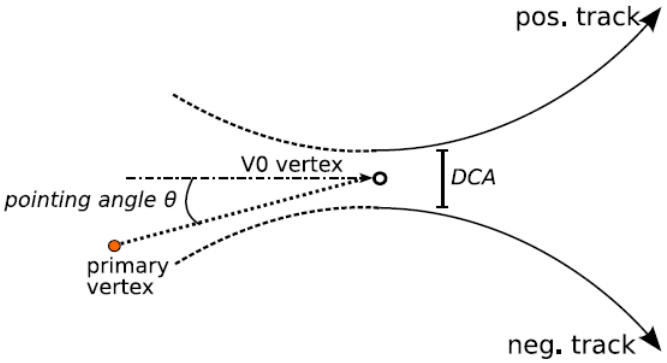


Figure 4 *– Topology of a V^0 decay, [AKal]*

4.2.1. Vertex

Vertex means:

- **Event vertex** or **Primary vertex**
  This is the location where two accelerated protons or lead nuclei collide and where other new particles come into existence.
- **Secondary vertex** or **V^0 vertex**
  This is the location where instable particles which arise from the collision or another process decay into new particles.
  The point of conversion of a photon into an electron-positron pair is also considered as secondary vertex.

4.2.2. Pointing angle

The straight line between the primary vertex and the V^0 vertex and the reconstructed mother track define the pointing angle (see figure 4).

4.2.3. ξ_{pair} and Ψ_{pair}

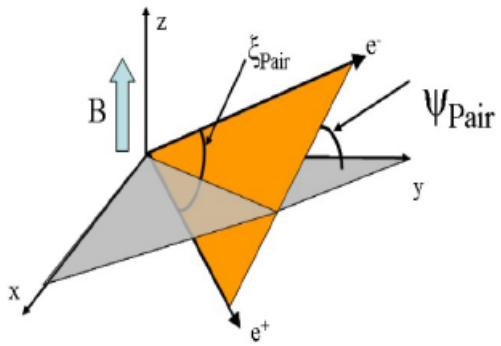


Figure 5 *– Illustration of the angles ξ_{pair} and Ψ_{pair}, example shown with e^+ and e^-, [Dah]*

In these V^0 decays, two decisive angles are defined [Dah] (see figure 5): ξ_{pair} and Ψ_{pair}. ξ_{pair} is the angle between the daughter tracks and is given by:

$$\xi_{\text{pair}} = \arccos\left(\frac{\vec{p_{i^-}} \cdot \vec{p_{i^+}}}{||\vec{p_{i^-}}|| \cdot ||\vec{p_{i^+}}||}\right) ,$$

in which $i^{\pm}$ is the negative daughter particle (e^-, p^-, π^-) or the positive daughter particle (e^+, p^+, π^+), $\vec{p_{i^{\pm}}}$ is the corresponding momentum.
Ψ_{pair} is the angle between the plane the daughter tracks span and the plane which is orthogonal to the magnetic field. Ψ_{pair} is given by:

$$\Psi_{\text{pair}} = \arcsin\left(\frac{\Delta\vartheta_0}{\zeta_{\text{pair}}}\right)$$

$\Delta\vartheta_0$ is the difference between the angles of the two daughter particles regarding the magnetic field direction $\vec{z}$:

$$\Delta\vartheta_0 = \vartheta_0(i^-) - \vartheta_0(i^+)$$

4.2.4. Invariant mass

The relativistic energy-momentum relation of a single particle [Rey] is defined as:

$$E^2 = m^2c^4 + p^2c^2$$

with energy E, mass m, momentum p, and speed of light c. In a decay process (or another isolated system) the total energy and momentum of all particles is constant at every time. The invariant mass M of a decay into two particles is given by:

$$\begin{aligned} M^2 &= (E_1 + E_2)^2 - (\vec{p_1} + \vec{p_2})^2 \\ &= m_1^2 + m_2^2 + 2E_1E_2 - 2\vec{p_1}\vec{p_2} \\ &= m_1^2 + m_2^2 + 2E_1E_2 - 2p_1p_2\cos\varphi \end{aligned}$$

with angle φ between the vectors of the daughter momenta.

4.2.5. χ^2 / NDF

In probability theory and statistics exists the term χ^2. One can hypothesise a certain mother particle to a vertex. Here it is a measure for the probability of that hypothesis. The higher this value is, the lower is the probability that the hypothesis is correct.

Another depending parameter is the number of degrees of freedom (NDF), which is the number of parameters that are free to vary. Typically, NDF is 3.
So there is a maximal accepted value of χ^2 / NDF (often called "reduced χ^2"). Every higher value is omitted.

4.3. Monte Carlo simulation

Monte Carlo simulation (MC simulation) is a stochastic method used for problems which are difficult to solve analytically or not solvable at all [PYT]. They can be solved numerically because of the *law of large numbers*. A high number of adequate events is simulated with computer programs to calculate the probability from the relative frequencies. One simulation program for particle physics is **PYTHIA**, which is used to simulate collisions at particle accelerators like the LHC.

5. Design of the V^0 analysis software

The following extracts of the C++ program `AliESDv0KineCuts.cxx` are from the version of April 20, 2011 [AOff].

5.1. How to use

5.1.1. An example

To run the PID, enter *AliRoot*. Load and start a task by giving it a list of data, for example:

```
root [0] .L V0/runV0task.C
root [1] runV0task(''lhc10d4_1.txt'',kTRUE,100,0)
```

The task `runV0task` is called via

C++ code extract 1 – *runV0task*

```
void runV0task(const char *treelist = 0x0, Bool_t hasMC = kTRUE, Int_t
    nFiles = 100, Int_t nSkip = 50)
```

This method needs a file with useful data. In the example the file name is `lhc10d4_1.txt`. The boolean variable `hasMC` transfers the information whether Monte Carlo simulated data are included or not. `nFiles` is a parameter to choose a number of data and `nSkip` is the number of skipped files, for example `nFiles = 100` and `nSkip = 50` skips the files up to and including position 50 and takes the next 100 files.
In this example, `runV0task` calls the method `AliTRDpidReferenceTask.cxx`, which is responsible for diverse histograms. Finally this method calls the method `AliESDv0KineCuts.cxx`, which shall be examined in this thesis.
After running the program (whose duration depends on the number of files (`nFiles`)), the command

```
root [2] new TBrowser()
```

opens a new window, a new ROOT Object Browser like in figure 6. In the user folder, for

example,

`/ALICE/HFE/V0task.root/V0cuts;1` *(The number can be 2, 3, 4, ..., depending on how many tasks are run.)*

there are numerous plots, located in three folders:

- `/ALICE/HFE/V0task.root/V0cuts;1/V0qa/V0cuts`
  This folder contains control distributions for the different cut variables. (see chapter 5.3)
- `/ALICE/HFE/V0task.root/V0cuts;1/V0qa/V0cutsMC`
  This folder contains distributions of all particles within the V^0 decays, based in the Monte Carlo PID.
- `/ALICE/HFE/V0task.root/V0cuts;1/V0qa/V0pid`
  This folder contains plot about the momenta of the particles and so-called Armenteros-Podolanski plots (see chapter 5.6).

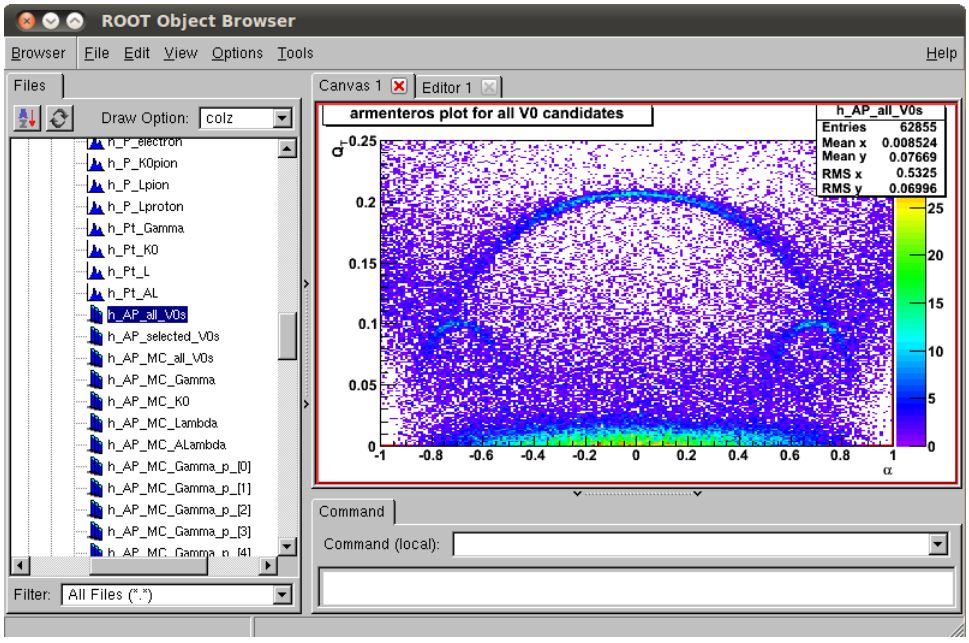


Figure 6 *– ROOT Object Browser where to find the designated plots*

5.1.2. User execution

The user has the possibility to develop his own task for his purpose. `AliESDv0KineCuts.cxx` inherits some elements (variables, constants, functions) from the header files:

- `AliESDv0KineCuts.h`, `AliESDv0.h`, `AliESDtrack.h`, `AliESDEvent.h`
- `AliVTrack.h`, `AliVEvent.h`
- `AliLog.h`, `AliKFParticle.h`, `AliKFVertex.h`

To run his own method, a method - which have to be called `UserExec` within the ALICE analysis framework - must exist, where the following methods can be set:

- `fV0cuts->SetEvent(AliESDEvent* const event);`
- `fV0cuts->SetPrimaryVertex(AliKFVertex* const v);`
- `fV0cuts->SetMode(Int_t mode, const char* type);`

`SetEvent` and `SetPrimaryVertex` sets the V^0 analysis. `SetMode` sets either higher purity (`mode = 0`) or higher efficiency (`mode = 1`), and selects either proton-proton (`type = pp`) or lead-lead (`type = PbPb`) mode.
If necessary, the user can change the selection cuts. Here are some examples:

- `fV0cuts->SetGammaCutChi2NDF(Float_t val);`
- `fV0cuts->SetK0CutVertexR(Float_t * const val);`
- `fV0cuts->SetLambdaCutInvMass(Float_t * const val);`

More commands to set the cuts can be seen in the header file `AliESDv0KineCuts.h`. Alternatively standard cut values are existing (see chapter 5.3).
The main user function is

C++ code extract 2 *– Main user function: ProcessV0*

```
Bool_t ProcessV0(AliESDv0* const v0, Int_t &pdgV0, Int_t &pdgP, Int_t &
    pdgN)
```

to run the PID via `AliESDv0KineCuts.cxx`.

5.2. General overview

Before the explanation of the program, figure 7 shows a scheme of the implementation of how the program works roughly. During a proton-proton or lead-lead collision, new particles come into existence. In the first instance, there are potential V^0 candidates. These candidates have been pre-selected during the event reconstruction, as described in chapter 4.2. In our analysis software, they run through the following checks:

- V^0 common cuts

  Until now, here the charges of the daughter particles are checked. Due to charge conservation, the detected daughter particles may not have the same charge.

- Single track cut

  This method checks the daughter particle tracks for kinks, TPC refit, and χ^2 per TPC cluster.

- Armenteros preselection

  Here the topology of the decay is checked with the help of the Armenteros-Podolanski plots.

- γ, K^0_S, Λ^0, $\overline{\Lambda^0}$

  Then the program distinguishes between the different cases and can finally give the information which particles are detected for the clean sample.

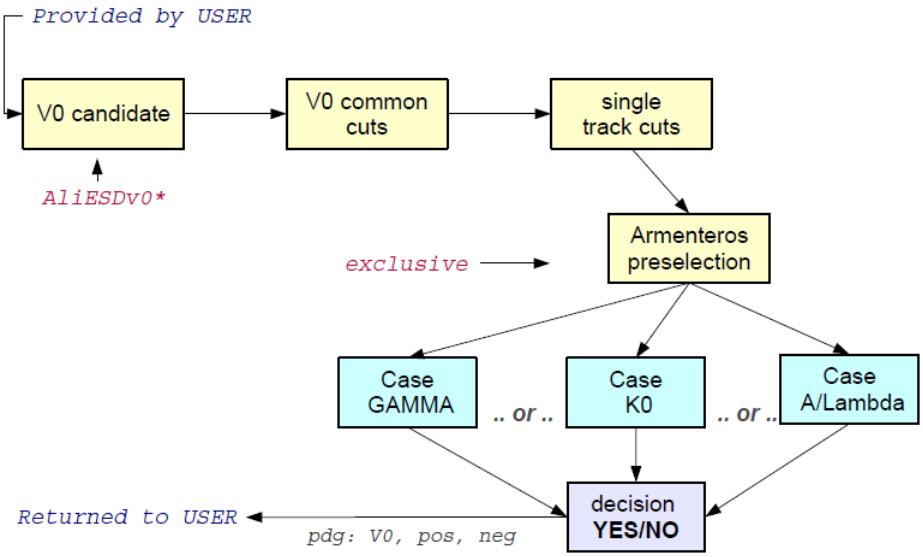


Figure 7 *– Scheme of the implementation, [MKal]*

5.3. Initialization

As a characteristic of a C++ program, the initialization takes place in the constructor, in which some initial values of the cuts will be defined. The cuts must be set for all V^0 types (γ, K^0_S, Λ^0) and the tracks in general. The following table 6 shows the default cuts for the V^0 particles. For more detailed settings, see code extract 13 in the appendix.

value	γ	K^0_S	Λ^0, $\overline{\Lambda^0}$
χ^2 / NDF	< 10	< 10	< 10
pointing angle	< 0.02	< 0.02	< 0.02
DCA of the daughter tracks [cm]	< 0.25	< 0.2	< 0.2
radius of the vertex [cm]	3 - 90	2 - 30	2 - 40
value of Ψ_{pair}	< 0.05		
invariant mass [GeV/c^2]	< 0.05	486 - 508	1.11 - 1.12

Table 6 *– Default skip values for the cuts*

All these values define the cuts to distinguish between the V^0 particles. The set values are based on measurements, which can be seen as plots for example in the folder:
`/ALICE/HFE/V0task.root/V0cuts;1/V0qa/V0cuts`.
The cuts are defined in such a way that the user gets as little contamination as possible, but as many tracks as possible at the same time.

- χ^2 **divided by the number of degrees of freedom (NDF)**, reduced χ^2, χ^2_{red}
  The cut value for the class `AliKFParticle` [ROOT], which is based on the Kalman Filter. The Kalman Filter is a mathematical procedure to estimate an unknown state vector (here: of the mother particle) as optimal as possible, based on measurements and calculation. According to [AKal], the AliKF vertexing package in AliRoot is precisely described in [GoKi].

- **pointing angle**
  It is reasonable to set a small value near zero. The smaller the pointing angle, the better the reconstruction line of the mother track and the vertex line match. Moreover we assume that the V^0 mother particles come from very close to the primary vertex. If the reconstructed mother does not match, it is likely a false combination of two daughters, which do not arise from a real decay.

- **DCA of the daughter tracks**

  Both the daughter particles have a minimum of distance to each other called distance of closest approach, DCA. See figure 4.

  Figures A.1 (a) to (c) show the number of counts depending on the DCA. One recognizes a high number of counts at few millimetres, so the limit was set at 0.2 cm for K_S^0 and Λ^0, and at 0.25 cm for γ.

- **radius of the vertex**

  This is the radial distance of the reconstructed decay from the event's primary vertex. This distance depends on the different life times of Λ^0 and K^0 respectively on the conversion of γ which cannot occur without material. Figures A.2 (a) to (c) show the number of counts depending on the radius of the vertex. Compared to K_S^0 and Λ^0, the conversion of γ quanta is not regular, but completely varying.

- **value of $\Psi_{\mathbf{pair}}$**

  See figure 5.

- **invariant mass**

  Sets the interval of the measured invariant mass of the particle, which is different in all three cases. Figures A.3 (a) to (c) show the number of counts depending on the invariant mass. According to table 3, the masses of the particles differ, which can be verified by figure A.3.

5.4. V^0 common cuts

At first, the program filters those tracks out which do not come from V^0 decays in the first place. This works with the method `V0CommonCuts` (see C++ code extract 15 in the appendix), where the program checks if a detected pair of particles can be daughter particles of a V^0 particle in general. For that this method tests the charge conservation of both particles by reason that a neutral V^0 particle always decays into one positive charged and one negative charged particle (or more unusually into two uncharged particles, which we do not analyse).

5.5. Single track cuts

The method `SingeTrackCuts` checks for several attributes. First it checks whether the TPC refit of the daughter particle track is alright "in order to get the track parameters at the vertex" [AOff2]. Then a float variable `chi2perTPCcluster`, which is χ^2 divided by the number of TPC clusters, is checked whether it is greater than 4 or not. A cluster is a "set of adjacent digits that were presumably generated by the same particle crossing the sensitive element of a detector" [AOff2]. So if a fit is not good enough (the smaller χ^2 is, the better the fit is), it is probably a V^0 candidate, if a great number of clusters is available. Another point are kinks in a track. Tracks must be a straight line (or a curve because of the magnetic field). If a kink is detected in any way, then this is an evidence for a collision of that particle with another particle or a decay. In both cases the detected particle is not useful.
In case, that the detected particles are no V^0s or only with little probability, the program returns `kFALSE`, else it returns `kTRUE`.

5.6. Armenteros preselection

The Armenteros-Podolanski plot (often abbreviated with "Armenteros plot") is the name of a special two-dimensional plot, in which the transverse momentum Q_T (in $\frac{\text{GeV}}{\text{c}}$) of the positive daughter particle with respect to the reconstructed mother particle's momentum is plotted against the longitudinal momentum asymmetry α, to illustrate the kinematic properties of the V^0 candidates. Q_T and α are defined by:

$$Q_T = \frac{|\vec{p_T} \times \vec{p_m}|}{|\vec{p_m}|}$$

$$\alpha = \frac{p_L^+ - p_L^-}{p_L^+ + p_L^-}$$

p_L^+ and p_L^- are the momenta of the daughter particles.
With the knowledge about the V^0 particles, the obtained picture can be explained in a simple way. The K_S^0 decays into two particles with the same mass. Their momenta are distributed symmetrically. In the decay of Λ^0 ($\overline{\Lambda^0}$) the p^+ (p^-) takes a larger part of the momentum than the π^- (π^+). The distribution is asymmetric. The e^+ and e^-, which result from the γ conversion are light particles compared to π or p particles, so the symmetric part is near zero momentum.

The more files `nFiles` are used, the more entries can be used to construct an Armenteros plot. The figures 8 (a) - (f) show how the plot develops with increasing number. There are all found V^0 candidates illustrated.

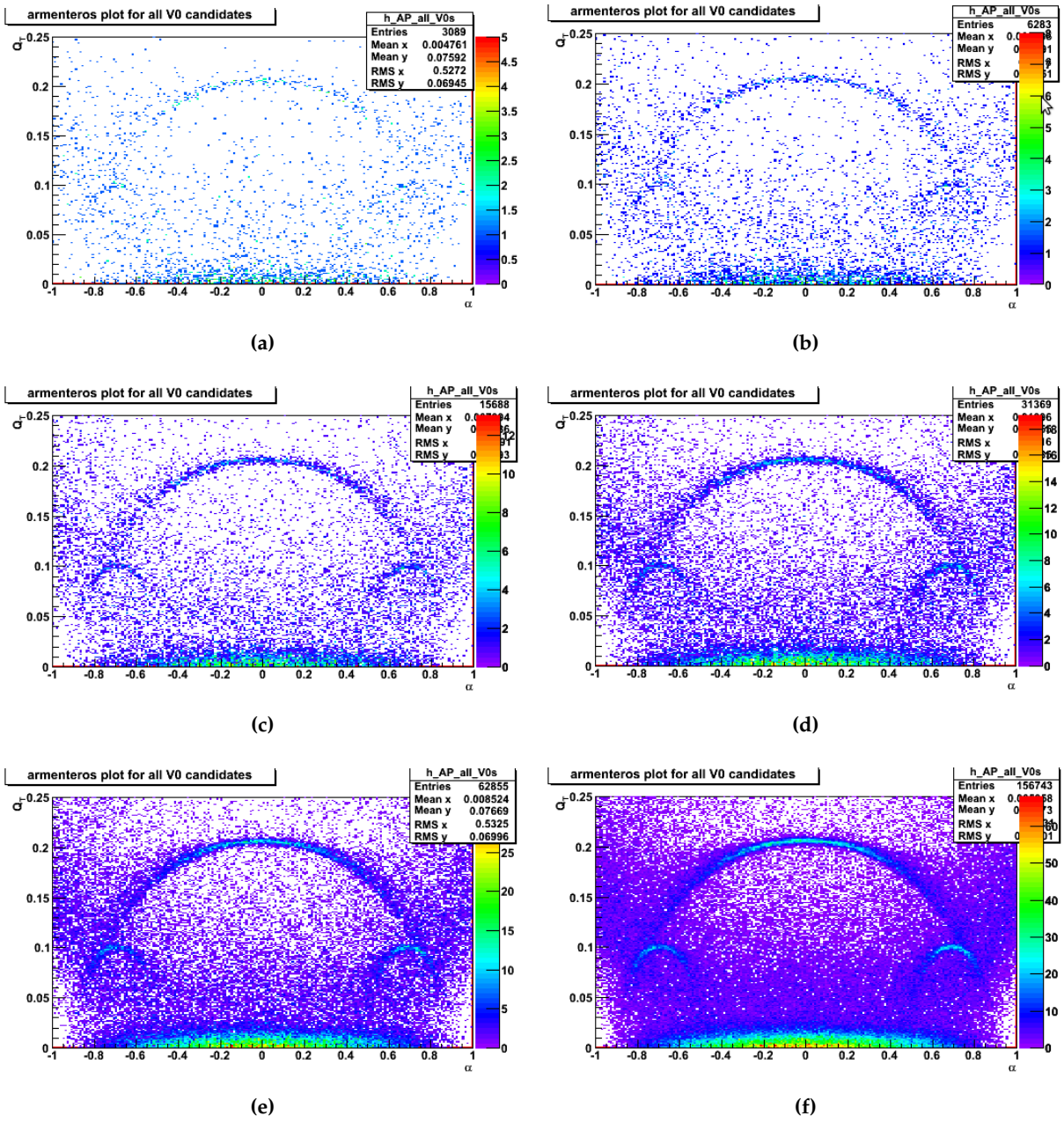


Figure 8 *– Armenteros plot of all found V^0 candidates with different numbers of entries:* `nFiles`*: (a) 10, (b) 20, (c) 50, (d) 100, (e) 200, (f) 500*

These plots show the different particles. The correct particles are identified by adjusted cuts. The values of the cuts are chosen in that way to assign particles clearly. Figure 9 shows the plot with applied cuts. Now the plot shows identified V^0 particles with low contamination.

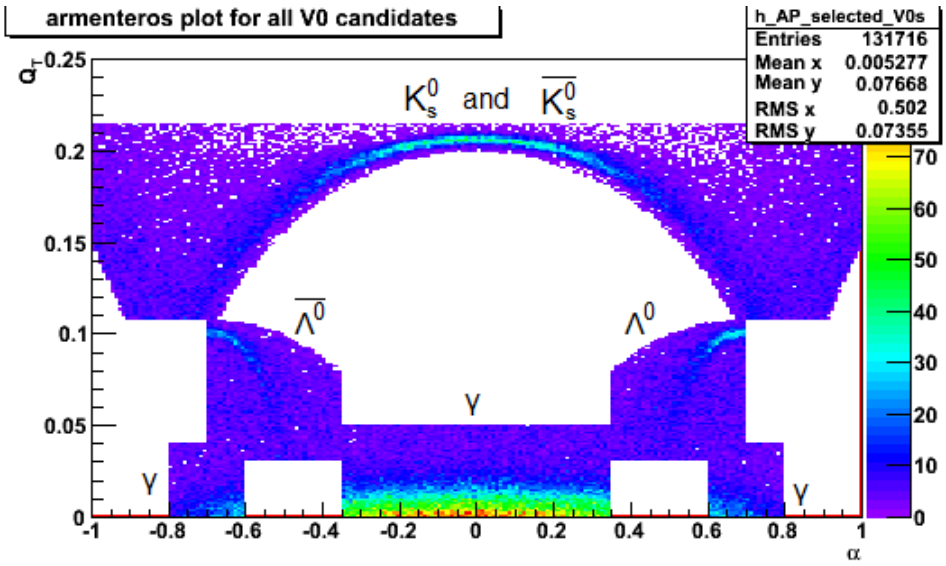


Figure 9 – *Armenteros plot (`nFiles` = 700) with applied cuts*

Next the track pairs are compared to the Armenteros simulated results with the help of the method `Armenteros` (see the code excerpt 16 in the appendix). In that method, some fix values are set as Armenteros preselection (see figure 9). Important for this selection is that the regions for the different candidates must not overlap, so detected vertices can be assigned clearly.

C++ code extract 3 – *Example values for cuts*

```
// Gamma cuts
 const Double_t cutAlphaG = 0.35;
 const Double_t cutQTG = 0.05;
 const Double_t cutAlphaG2[2] = {0.6, 0.8};
 const Double_t cutQTG2 = 0.04;

 // K0 cuts
 const Float_t cutQTK0[2] = {0.1075, 0.215};
 const Float_t cutAPK0[2] = {0.199, 0.8};

 // Lambda & A-Lambda cuts
 const Float_t cutQTL = 0.03;
 const Float_t cutAlphaL[2] = {0.35, 0.7};
 const Float_t cutAlphaAL[2] = {-0.7, -0.35};
 const Float_t cutAPL[3] = {0.107, -0.69, 0.5};
```

These constant variables set the cuts to the transverse momentum Q_T and the longitudinal momentum asymmetry α for every single V^0. Additionally K^0_S, Λ^0 and $\overline{\Lambda^0}$ gets one more cut value, which is a parameter for the curved Q_T.
After this, the program checks either for γ, K^0_S, Λ^0 or $\overline{\Lambda^0}$ candidates and returns one of the following values to the integer constant `id`: `kGamma`, `kK0`, `kLambda`, `kALambda` or `kUndef`. Depending on the V^0 candidate the variable `id` calls a specific method via distinction of cases.

C++ code extract 4 – *Differentiation of V^0 cases*

```
switch(id){
case kUndef:
  return kFALSE;
case kGamma:
  return CaseGamma(v0, pdgV0, pdgP, pdgN);
case kK0:
  return CaseK0(v0, pdgV0, pdgP, pdgN);
case kLambda:
  return CaseLambda(v0, pdgV0, pdgP, pdgN, 0);
case kALambda:
  return CaseLambda(v0, pdgV0, pdgP, pdgN, 1);
}
```

The variables describe the V^0 candidate by `pdgV0`, which is the PDG number (see table 4). `pdgP` and `pdgN` give information about the sign of the charge (+ or -, for example p^+ or p^-). This is especially important for Λ^0 and $\overline{\Lambda^0}$. The method `CaseLambda` gets one more value to work with: 0 or 1. This number differentiates between Λ^0 and its antiparticle.

Every single method analyses the detected particle regarding several aspects and reads exact values for comparing values like cosinus of the pointing angle, DCA between the daughter tracks, production vertex, reduced χ^2 and the angle Ψ_{pair} with the cut conditions. In addition, one virtual mother particle is created to check if the detected daughter particles fit to the virtual mother particle.
If any of these cases is not fulfilled the program returns the value `kFALSE`. Else the V^0 candidate passes all the cuts and the program returns the value `kTRUE`.

5.7. Decision

The three method cases are basically the same. In the following, all code excerpts are excerpts from the method `CaseGamma`.

In the γ and K^0_S case the charge of the daughter particles are unimportant, so the program checks the charge and switches them for further consistent working if necessary.

C++ code extract 5 – *γ verification: Check sign of the charge*

```
Bool_t sign = CheckSigns(v0);
 if(sign){
   pIndex = v0->GetPindex();
   nIndex = v0->GetNindex();
 }
 else{
   pIndex = v0->GetNindex();
   nIndex = v0->GetPindex();
 }
```

Next the program checks whether the tracks are available or not. It could be that one or both tracks are not correct due to bugs. If this is the case, the value `kFALSE` is returned.

C++ code extract 6 – *γ verification: Check the daughter tracks*

```
daughter[0] = dynamic_cast<AliVTrack *>(fEvent->GetTrack(pIndex));
daughter[1] = dynamic_cast<AliVTrack *>(fEvent->GetTrack(nIndex));
if(!daughter[0] || !daughter[1]) return kFALSE;
```

If two daughter tracks are found, another sub-program `CreateMotherParticle` (see C++ code extract 17 in the appendix) creates a virtual mother particle to check if the detected daughter particles fit to the virtual mother particle. This is done via the Kalman filter method.

In the next step the program reads the values of cosinus of the pointing angle, Ψ_{pair} (see C++ code extract 18 in the appendix), DCA between the daughter tracks, reduced χ^2 and point of production vertex (see C++ code extract 19 in the appendix) and compares these values to the Armenteros plot or rather to the adjusted cuts. If any of these criteria is not fulfilled the value `kFALSE` is returned.

C++ code extract 7 – *γ verification: Apply the cuts*

```
if(iMass > fGcutInvMass) return kFALSE;

 if(chi2ndf > fGcutChi2NDF) return kFALSE;

 if(cosPoint < fGcutCosPoint[0] || cosPoint > fGcutCosPoint[1]) return
    kFALSE;

 if(dca < fGcutDCA[0] || dca > fGcutDCA[1]) return kFALSE;

 if(r < fGcutVertexR[0] || r > fGcutVertexR[1]) return kFALSE;

 if(psiPair < fGcutPsiPair[0] || psiPair > fGcutPsiPair[1]) return
    kFALSE;
```

In those cases there is no V^0 particle detected. But if a detected particle passes all cuts, this particle is specified by a certain number within a variable. All these particle numbers were given by the Particle Data Group (PDG) [PDG2]. An overview of the numbers concerning the V^0 particles is shown in table 3. At the end of this method the value `kTRUE` is returned.

C++ code extract 8 – *γ verification: All cuts passed*

```
pdgV0 = 22;
if(sign){
  pdgP = -11;
  pdgN = 11;
}
else{
  pdgP = 11;
  pdgN = -11;
}

return kTRUE;
```

C++ code extract 9 – *K^0_S verification: All cuts passed*

```
pdgV0 = 310;
if(sign){
  pdgP = 211;
```

```
    pdgN = -211;
  }
  else{
    pdgP = -211;
    pdgN = 211;
  }

  return kTRUE;
```

The other methods for K^0_S and Λ^0 ($\overline{\Lambda^0}$) differ in the values and cut settings. Apart from that the methods are basically the same.

Only the method for `CaseLambda` differs in some parts, differentiating between Λ^0 particles and $\overline{\Lambda^0}$ particles.
Creating the mother particle checks the daughter particles. A p^+ particle and a π^- particle indicates a Λ^0 while a p^- particle and a π^+ particle indicates a $\overline{\Lambda^0}$.

C++ code extract 10 – *Creating the mother particle of $p^\pm$ and $\pi^\mp$*

```
AliKFParticle *kfMother[2] = {0x0, 0x0};

// Lambda
kfMother[0] = CreateMotherParticle(daughter[0], daughter[1], TMath::Abs
    (kProton), TMath::Abs(kPiPlus));
if(!kfMother[0]) return kFALSE;

// Anti-Lambda
kfMother[1] = CreateMotherParticle(daughter[0], daughter[1], TMath::Abs
    (kPiPlus), TMath::Abs(kProton));
if(!kfMother[1]) return kFALSE;
```

A proton is nearly seven times as heavy as a charged pion, so it is more probable that the momentum of a proton is greater than the pion's momentum. To differentiate the daughter particles the program checks their momentum, their mass, and the correlation of these comparisons.

C++ code extract 11 – *Checking momentum and mass of daughter particles*

```
// check the 3 lambda - antilambda variables
Int_t check[2] = {-1, -1};   // 0 : lambda, 1 : antilambda
// 1) momentum of the daughter particles - proton is expected to have
    higher momentum than pion
check[0] = (p[0] > p[1]) ? 0 : 1;
// 2) mass of the mother particle
check[1] = (dMass[0] < dMass[1]) ? 0 : 1;

// require positive correlation of (1) and (2)
if(check[0] != check[1]){
  if(kfMother[0]) delete kfMother[0];
  if(kfMother[1]) delete kfMother[1];
  return kFALSE;
}

// now that the check[0] == check[1]
const Int_t type = check[0];

// require that the input armenteros preselection agree:
if(type != id) return kFALSE;
```

After applying all cuts, the program gives values of the PDG numbers to the variables.

C++ code extract 12 – Λ^0 *or* $\overline{\Lambda^0}$ *verification: All cuts passed*

```
if(0 == type){
  pdgV0 = 3122;
  if(sign){
    pdgP = 2212;
    pdgN = -211;
  }
  else{
    pdgP = -211;
    pdgN = 2212;
  }
}
else{
```

```
    pdgV0 = -3122;
    if(sign){
      pdgP = 211;
      pdgN = -2212;
    }
    else{
      pdgP = -2212;
      pdgN = 211;
    }
  }

  return kTRUE;
```

So after the run through the program the be user now gets information whether the detector was hit by daughter particles of V^0 particles or not. This information can used for further analyses. But more relevant is the information of the samples, because now the user has clean samples of protons, electrons, or pions with less contamination.

6. Summary

In this thesis I dealt with the functionality of the C++ program `AliESDv0KineCuts.cxx`, which is used to analyse daughter particles of V^0 particles.
The program works well and is able to identify and to filter requested particles with good precision. The contamination is between 0.1 % and 10 %, depending on the particle momentum [MKal].
Since a decay is always a statistical process, every detected particle with all its values (transverse momentum, longitudinal momentum asymmetry, angles, vertices, and so on) is subject to statistical variations. So it is difficult to improve the cut values.
Nevertheless the cut program serves the purpose to gain pure samples of the V^0 daughters without accessing on the PID information of the detectors. In this way, it is possible to calibrate the detectors.
Of course, this program is not perfect due to statistic but prospectively the program will be improved and expanded by the ALICE members to improve the efficiency of the particle identification, in particular for momenta higher than 3 to 5 GeV. Furthermore cuts for high multiplicity environment must be prepared.

Betone den Hauptzweck dieser Cut Klasse: Sie dient dazu möglichst reine samples von V0 Töchtern (und damit identifizierten e/p/pi) zu bekommen ohne auf die PID Informationen der Detektoren zurückzugreifen. Damit man damit kalibrieren kann.

References

[AKal] Alexander Philipp Kalweit: *Energy Loss Calibration of the ALICE Time Projection Chamber*, pp. 29 - 30, Master Thesis, Technische Universität Darmstadt, May 2008

[AOff] ALICE@CERN: *ALICE Offline Pages*, `http://aliceinfo.cern.ch/Offline/`, ../AliRoot/trunk/ANALYSIS/, `http://alisoft.cern.ch/viewvc/trunk/ANALYSIS/?root=AliRoot`, 31 July 2011

[AOff2] ALICE@CERN: *Reconstruction | ALICE Offline Pages*, `http://aliceinfo.cern.ch/Offline/Activities/Reconstruction/index.html`, 29 August 2011

[Dah] Torsten Dahms: PDF foil about *Test of "New" Cut for Identifying Conversion*

[GoKi] S. Gorbunov and I. Kisel: *Reconstruction of decayed particles based on the Kalman filter*, CBM-SOFT-note-2007-003, 2007

[Gus] Hans-Ake Gustafsson (corresponding author) et al.: *The ALICE experiment at the CERN LHC*, published by Institute of Physics Publishing and SISSA, 2008; first published electronically as *2008 JINST 3 S08002*: `http://iopscience.iop.org/1748-0221/3/08/S08002/`

[ITS] *LHC - ALICE Detektor*, `http://www.lhc-facts.ch/index.php?page=alice`, 31 July 2011

[Kol] Hermann Kolanosk: *Einführung in die Kern- und Elementarteilchenphysik*, lecture at Humboldt Universität zu Berlin, winter semester 2007/08, `http://www-zeuthen.desy.de/~kolanosk/ket0708/skript/hist01.pdf`, 16 August 2011

[LWA] R.B. Leighton, S. D. Wanlass and C. D. Anderson: *The Decay of V^0 Particles*, Physical Review, Volume 89, Number 1, January 1, 1953, pp. 148 - 167

[MKal] Matus Kalisky: Speech about *V0 - 101, .. or what you never wanted to know about V0s but was given no other choice ..*, WWU Münster

[PDG1] Particle Data Group:
Strange mesons,
`http://pdg.lbl.gov/2011/tables/rpp2011-tab-mesons-strange.pdf`, 31 July 2011
Λ *baryons*,
`http://pdg.lbl.gov/2011/tables/rpp2011-tab-mesons-strange.pdf`, 31 July 2011

Light unflavored mesons,
http://pdg.lbl.gov/2011/tables/rpp2011-tab-mesons-light.pdf, 31 July 2011

[PDG2] Particle Data Group: *Masses, widths, and MC ID numbers from 2010 edition of RPP*, http://pdg.lbl.gov/2010/mcdata/mass_width_2010.mcd, 20 May 2011

[PDG3] Particle Data Group:
W-boson,
http://pdg.lbl.gov/2011/listings/rpp2011-list-w-boson.pdf, 17 August 2011
Z-boson,
http://pdg.lbl.gov/2011/listings/rpp2011-list-z-boson.pdf, 17 August 2011

[PYT] *Official PYTHIA page*: http://home.thep.lu.se/~torbjorn/Pythia.html, 18 July 2011

[Rey] Klaus Reygers: Speech about *Ultrarelativistische Schwerionenphysik - Quarks, Gluonen und Quark-Gluon-Plasma*, Institut für Kernphysik, Universität Münster, 2006

[ROOT] The ROOT Team: *ROOT | A Data Analysis Framework*, http://aliceinfo.cern.ch/static/aliroot-pro/html/roothtml/AliKFParticle.html, ../AliRoot/STEER/AliKFParticle/, 01 August 2011

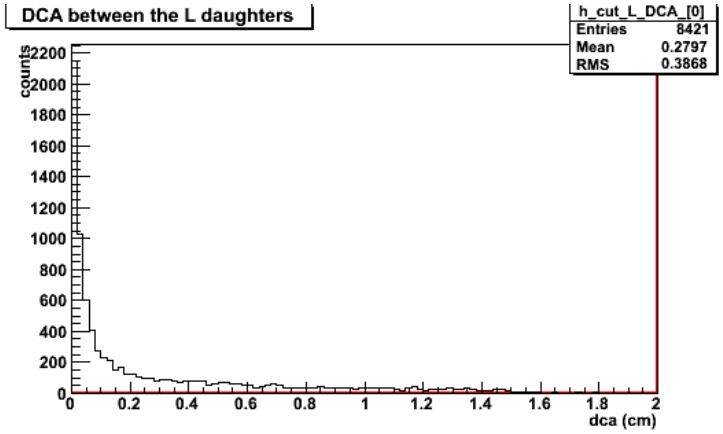
[SM] *The standard package*, http://public.web.cern.ch/public/en/science/StandardModel-en.html, CERN 2008, 31 July 2011

[TPC1] ALICE@CERN: *ALICE Time Projection Chamber*, http://aliweb.cern.ch/TPC/, 28 July 2011

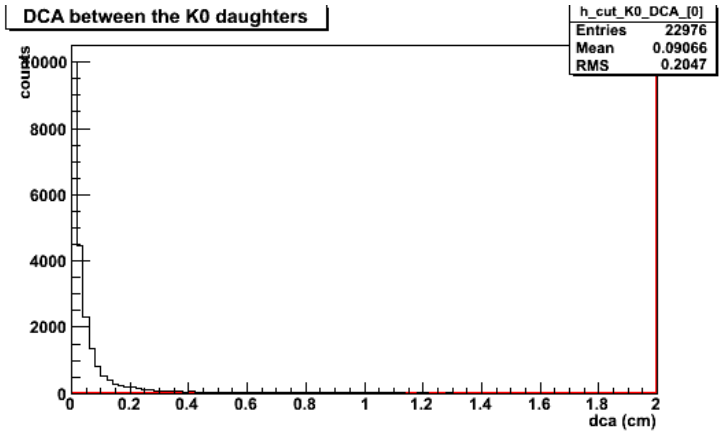
[TPC2] ALICE Collaboration: *The ALICE Time Projection Chamber*, http://aliceinfo.cern.ch/Public/en/Chapter2/Chap2_TPC.html, 26 May 2011

A. Appendix

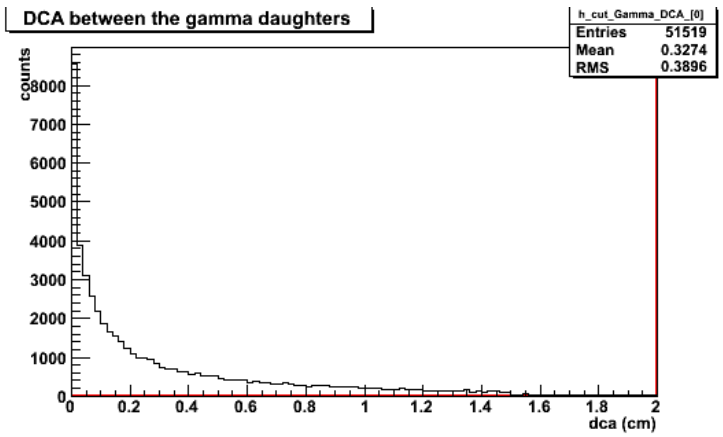
A.1. Figures



(a)

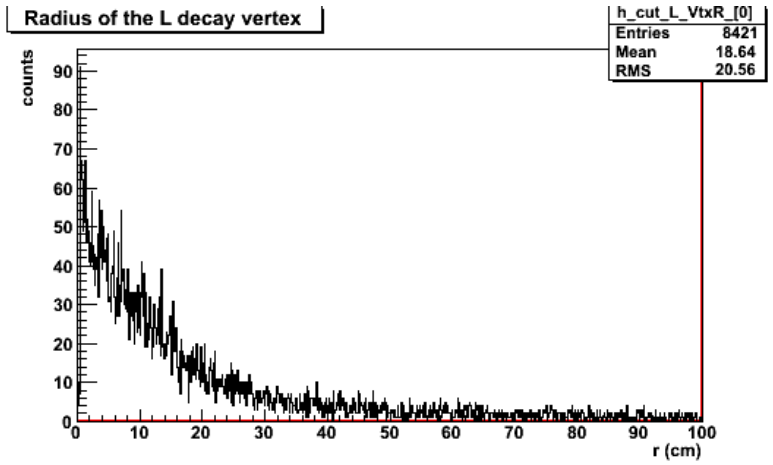


(b)

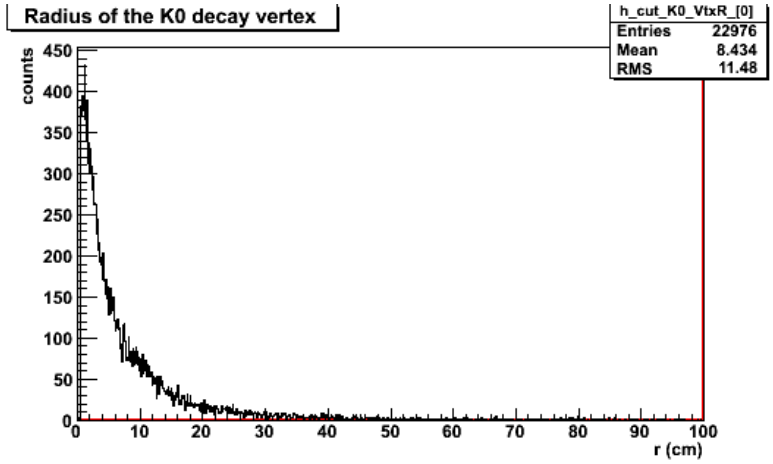


(c)

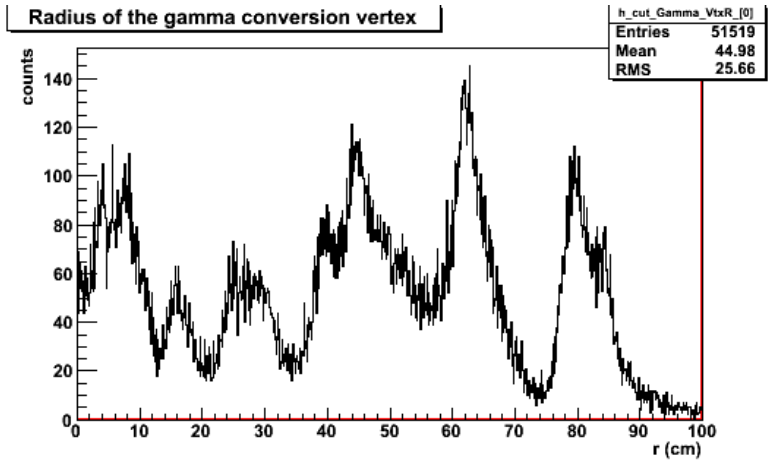
Figure A.1 – *Plots of the DCA of the V^0 daughter tracks: (a) Λ^0, (b) K^0_S, (c) γ*



(a)

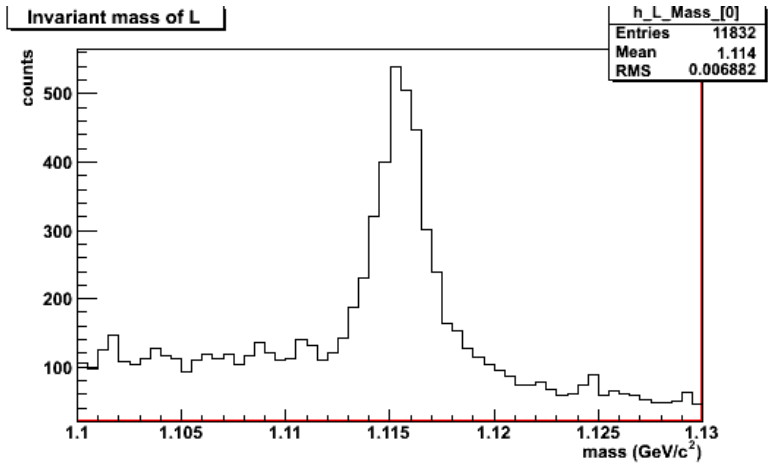


(b)

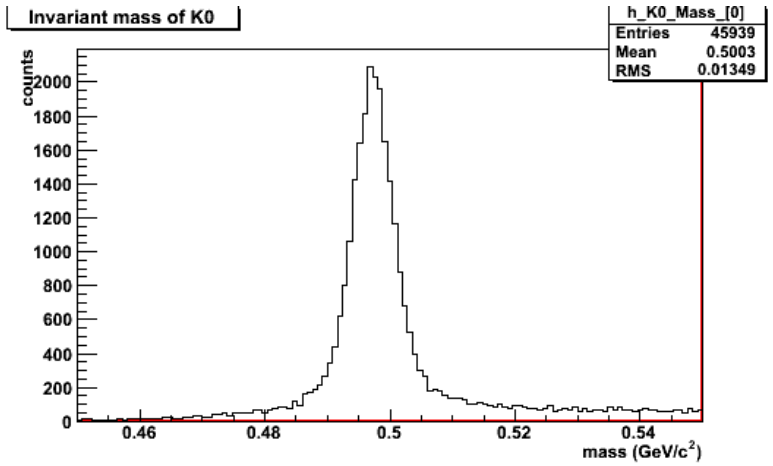


(c)

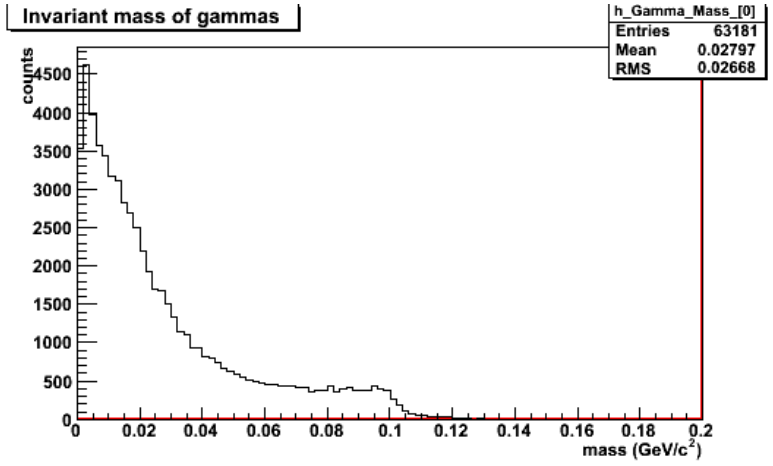
Figure A.2 *– Plots of the radius of the vertex of the V^0 daughter tracks: (a) Λ^0, (b) K^0_S, (c) γ*



(a)



(b)



(c)

Figure A.3 *– Plots of the invariant mass of the V^0 daughter tracks: (a) Λ^0, (b) K^0_S, (c) γ*

A.2. Code excerpts

C++ code extract 13 – *Default constructor*

```
// default single track cuts
fTPCNcls = 1;
fTPCrefit = kTRUE;
fTPCchi2perCls = 4.0;
fTPCclsRatio = 0.6;
fNoKinks = kTRUE;

// default gamma cuts values
fGcutChi2NDF = 10;
fGcutCosPoint[0] = 0;
fGcutCosPoint[1] = 0.02;
fGcutDCA[0] = 0.;
fGcutDCA[1] = 0.25;
fGcutVertexR[0] = 3.;
fGcutVertexR[1] = 90.;
fGcutPsiPair[0] = 0.;
fGcutPsiPair[1] = 0.05;
fGcutInvMass = 0.05;

// default K0 cuts
fK0cutChi2NDF = 10;
fK0cutCosPoint[0] = 0.;
fK0cutCosPoint[1] = 0.02;
fK0cutDCA[0] = 0.;
fK0cutDCA[1] = 0.2;
fK0cutVertexR[0] = 2.0;
fK0cutVertexR[1] = 30.0;
fK0cutInvMass[0] = 0.486;
fK0cutInvMass[1] = 0.508;

// Lambda & anti-Lambda cut values
fLcutChi2NDF = 10;
fLcutCosPoint[0] = 0.;
fLcutCosPoint[1] = 0.02;
```

```
  fLcutDCA[0] = 0.;
  fLcutDCA[1] = 0.2;
  fLcutVertexR[0] = 2.0;
  fLcutVertexR[1] = 40.0;
  fLcutInvMass[0] = 1.11;
  fLcutInvMass[1] = 1.12;
```

C++ code extract 14 – *Method* `SingleTrackCuts`

```
Bool_t const AliESDv0KineCuts::SingleTrackCuts(AliESDv0 * const v0){
  //
  // apply single track cuts
  // correct sign not relevat here
  //

  if(!v0) return kFALSE;

  Int_t pIndex = 0, nIndex = 0;
  pIndex = v0->GetPindex();
  nIndex = v0->GetNindex();
  AliESDtrack* d[2];
  d[0] = dynamic_cast<AliESDtrack*>(fEvent->GetTrack(pIndex));
  d[1] = dynamic_cast<AliESDtrack*>(fEvent->GetTrack(nIndex));

  for(Int_t i=0; i<2; ++i){
    if(!d[i]) return kFALSE;

    // status word
    ULong_t status = d[i]->GetStatus();

    // No. of TPC clusters leave to the users
    if(d[i]->GetTPCNcls() < 1) return kFALSE;

    // TPC refit
    if(!(status & AliESDtrack::kTPCrefit)) return kFALSE;

    // Chi2 per TPC cluster
    Int_t nTPCclusters = d[i]->GetTPCNcls();
```

```
    Float_t chi2perTPCcluster = d[i]->GetTPCchi2()/Float_t(nTPCclusters);
    if(chi2perTPCcluster > 4) return kFALSE;

    // TPC cluster ratio
    Float_t cRatioTPC = d[i]->GetTPCNclsF() > 0. ? static_cast<Float_t>(d
        [i]->GetTPCNcls())/static_cast<Float_t> (d[i]->GetTPCNclsF()) :
        1.;
    if(cRatioTPC < 0.6) return kFALSE;

    // kinks
    if(d[i]->GetKinkIndex(0) != 0) return kFALSE;

  }

  return kTRUE;
}
```

C++ code extract 15 – *Method* `V0CutsCommon`

```
Bool_t const AliESDv0KineCuts::V0CutsCommon(AliESDv0 * const v0){
  //
  // V0 cuts common to all V0s
  //

  AliESDtrack* dN, *dP;

  dP = dynamic_cast<AliESDtrack *>(fEvent->GetTrack(v0->GetPindex()));
  dN = dynamic_cast<AliESDtrack *>(fEvent->GetTrack(v0->GetNindex()));

  if(!dN || !dP) return kFALSE;

  Int_t qP = dP->Charge();
  Int_t qN = dN->Charge();

  if((qP*qN) != -1) return kFALSE;

  return kTRUE;
}
```

C++ code extract 16 – *Method* `Armenteros`

```
void const AliESDv0KineCuts::Armenteros(AliESDv0* const v0, Float_t val
    [2]){
  //
  // computes the Armenteros variables for given V0
  // fills the histogram
  // returns the values via "val"
  //

  Double_t mn[3] = {0,0,0};
  Double_t mp[3] = {0,0,0};
  Double_t mm[3] = {0,0,0};

  if(CheckSigns(v0)){
    v0->GetNPxPyPz(mn[0],mn[1],mn[2]); //reconstructed cartesian momentum
        components of negative daughter
    v0->GetPPxPyPz(mp[0],mp[1],mp[2]); //reconstructed cartesian momentum
        components of positive daughter
  }
  else{
    v0->GetPPxPyPz(mn[0],mn[1],mn[2]); //reconstructed cartesian momentum
        components of negative daughter
    v0->GetNPxPyPz(mp[0],mp[1],mp[2]); //reconstructed cartesian momentum
        components of positive daughter
  }
  v0->GetPxPyPz(mm[0],mm[1],mm[2]); //reconstructed cartesian momentum
      components of mother

  TVector3 vecN(mn[0],mn[1],mn[2]);
  TVector3 vecP(mp[0],mp[1],mp[2]);
  TVector3 vecM(mm[0],mm[1],mm[2]);

  Double_t thetaP = acos((vecP * vecM)/(vecP.Mag() * vecM.Mag()));
  Double_t thetaN = acos((vecN * vecM)/(vecN.Mag() * vecM.Mag()));

  Double_t alfa = ((vecP.Mag())*cos(thetaP)-(vecN.Mag())*cos(thetaN))/
    ((vecP.Mag())*cos(thetaP)+(vecN.Mag())*cos(thetaN)) ;
```

```
  Double_t qt = vecP.Mag()*sin(thetaP);

  val[0] = alfa;
  val[1] = qt;
}
```

C++ code extract 17 – *Method* `CreateMotherParticle`

```
AliKFParticle *AliESDv0KineCuts::CreateMotherParticle(const AliVTrack*
    const pdaughter, const AliVTrack* const ndaughter, Int_t pspec, Int_t
    nspec){
  //
  // Creates a mother particle
  //
  AliKFParticle pkfdaughter(*pdaughter, pspec);
  AliKFParticle nkfdaughter(*ndaughter, nspec);


  // Create the mother particle
  AliKFParticle *m = new AliKFParticle(pkfdaughter, nkfdaughter);
  m->SetField(fEvent->GetMagneticField());
  if(TMath::Abs(kElectron) == pspec && TMath::Abs(kElectron) == nspec) m
      ->SetMassConstraint(0, 0.001);
  else if(TMath::Abs(kPiPlus) == pspec && TMath::Abs(kPiPlus) == nspec) m
      ->SetMassConstraint(TDatabasePDG::Instance()->GetParticle(kK0Short)
      ->Mass(), 0.);
  else if(TMath::Abs(kProton) == pspec && TMath::Abs(kPiPlus) == nspec) m
      ->SetMassConstraint(TDatabasePDG::Instance()->GetParticle(kLambda0)
      ->Mass(), 0.);
  else if(TMath::Abs(kPiPlus) == pspec && TMath::Abs(kProton) == nspec) m
      ->SetMassConstraint(TDatabasePDG::Instance()->GetParticle(kLambda0)
      ->Mass(), 0.);
  else{
    AliErrorClass("Wrong daughter ID - mass constraint can not be set");
  }

  AliKFVertex improvedVertex = *fPrimaryVertex;
  improvedVertex += *m;
```

```
  m->SetProductionVertex(improvedVertex);

  // update 15/06/2010
  // mother particle will not be added to primary vertex but only to its
     copy
  // as this confilcts with calling
  // m->SetPrimaryVertex() function and
  // subsequently removing the mother particle afterwards
  // Source: Sergey Gorbunov

  return m;
}
```

C++ code extract 18 – *Method* `PsiPair`

```
Double_t const AliESDv0KineCuts::PsiPair(AliESDv0* const v0) {
  if(!fEvent) return -1.;

  Float_t magField = fEvent->GetMagneticField();

  Int_t pIndex = -1;
  Int_t nIndex = -1;
  if(CheckSigns(v0)){
    pIndex = v0->GetPindex();
    nIndex = v0->GetNindex();
  }
  else{
    pIndex = v0->GetNindex();
    nIndex = v0->GetPindex();
  }



  AliESDtrack* daughter[2];

  daughter[0] = dynamic_cast<AliESDtrack *>(fEvent->GetTrack(pIndex));
  daughter[1] = dynamic_cast<AliESDtrack *>(fEvent->GetTrack(nIndex));

  Double_t x, y, z;
```

```
v0->GetXYZ(x,y,z);//Reconstructed coordinates of V0; to be replaced by
    Markus Rammler's method in case of conversions!

Double_t mn[3] = {0,0,0};
Double_t mp[3] = {0,0,0};



v0->GetNPxPyPz(mn[0],mn[1],mn[2]);//reconstructed cartesian momentum
    components of negative daughter;
v0->GetPPxPyPz(mp[0],mp[1],mp[2]);//reconstructed cartesian momentum
    components of positive daughter;



Double_t deltat = 1.;
deltat = TMath::ATan(mp[2]/(TMath::Sqrt(mp[0]*mp[0] + mp[1]*mp[1])+1.e
    -13)) -  TMath::ATan(mn[2]/(TMath::Sqrt(mn[0]*mn[0] + mn[1]*mn[1])
    +1.e-13));//difference of angles of the two daughter tracks with z-
    axis

Double_t radiussum = TMath::Sqrt(x*x + y*y) + 50;//radius to which
    tracks shall be propagated

Double_t momPosProp[3];
Double_t momNegProp[3];

AliExternalTrackParam pt(*daughter[0]), nt(*daughter[1]);

Double_t psiPair = 4.;

if(nt.PropagateTo(radiussum,magField) == 0)//propagate tracks to the
    outside
  psiPair =  -5.;
if(pt.PropagateTo(radiussum,magField) == 0)
  psiPair = -5.;
pt.GetPxPyPz(momPosProp);//Get momentum vectors of tracks after
    propagation
nt.GetPxPyPz(momNegProp);
```

```
  Double_t pEle =
    TMath::Sqrt(momNegProp[0]*momNegProp[0]+momNegProp[1]*momNegProp[1]+
        momNegProp[2]*momNegProp[2]);//absolute momentum value of negative
         daughter
  Double_t pPos =
    TMath::Sqrt(momPosProp[0]*momPosProp[0]+momPosProp[1]*momPosProp[1]+
        momPosProp[2]*momPosProp[2]);//absolute momentum value of positive
         daughter

  Double_t scalarproduct =
    momPosProp[0]*momNegProp[0]+momPosProp[1]*momNegProp[1]+momPosProp
        [2]*momNegProp[2];//scalar product of propagated positive and
        negative daughters' momenta

  Double_t chipair = TMath::ACos(scalarproduct/(pEle*pPos));//Angle
      between propagated daughter tracks

  psiPair =  TMath::Abs(TMath::ASin(deltat/chipair));

  return psiPair;
}
```

C++ code extract 19 – *Method* `GetConvPosXY`

```
Bool_t const AliESDv0KineCuts::GetConvPosXY(AliESDtrack * const ptrack,
    AliESDtrack * const ntrack, Double_t convpos[2]){
  //
  // Recalculate the gamma conversion XY position
  //

  const Double_t b = fEvent->GetMagneticField();

  Double_t helixcenterpos[2];
  GetHelixCenter(ptrack,b,ptrack->Charge(),helixcenterpos);

  Double_t helixcenterneg[2];
  GetHelixCenter(ntrack,b,ntrack->Charge(),helixcenterneg);
```

```
  Double_t  poshelix[6];
  ptrack->GetHelixParameters(poshelix,b);
  Double_t posradius = TMath::Abs(1./poshelix[4]);

  Double_t  neghelix[6];
  ntrack->GetHelixParameters(neghelix,b);
  Double_t negradius = TMath::Abs(1./neghelix[4]);

  Double_t xpos = helixcenterpos[0];
  Double_t ypos = helixcenterpos[1];
  Double_t xneg = helixcenterneg[0];
  Double_t yneg = helixcenterneg[1];

  convpos[0] = (xpos*negradius + xneg*posradius)/(negradius+posradius);
  convpos[1] = (ypos*negradius+  yneg*posradius)/(negradius+posradius);

  return 1;
}
```

C++ code extract 20 – *Method* `GetHelixCenter`

```
Bool_t const AliESDv0KineCuts::GetHelixCenter(AliESDtrack * const track,
    Double_t b,Int_t charge, Double_t center[2]){
  //
  // computes the center of the track helix
  //

  Double_t pi = TMath::Pi();

  Double_t  helix[6];
  track->GetHelixParameters(helix,b);

  Double_t xpos =  helix[5];
  Double_t ypos =  helix[0];
  Double_t radius = TMath::Abs(1./helix[4]);
  Double_t phi = helix[2];
```

```
  if(phi < 0){
    phi = phi + 2*pi;
  }

  phi -= pi/2.;
  Double_t xpoint =  radius * TMath::Cos(phi);
  Double_t ypoint =  radius * TMath::Sin(phi);

  if(b<0){
    if(charge > 0){
      xpoint = - xpoint;
      ypoint = - ypoint;
    }

    if(charge < 0){
      xpoint =  xpoint;
      ypoint =  ypoint;
    }
  }
  if(b>0){
    if(charge > 0){
      xpoint =  xpoint;
      ypoint =  ypoint;
    }

    if(charge < 0){
      xpoint = - xpoint;
      ypoint = - ypoint;
    }
  }
  center[0] =  xpos + xpoint;
  center[1] =  ypos + ypoint;

  return 1;
}
```

Danksagung

An dieser Stelle danke ich allen Leuten, die zur Entstehung dieser Arbeit beigetragen haben.
Zunächst danke ich Prof. Dr. Johannes P. Wessels für das interessante Thema der Teilchenidentifikation und die Aufnahme in seine Arbeitsgruppe während der Anfertigung meiner Bachelorarbeit.
Des Weiteren danke ich herzlich Dr. Christian Klein-Bösing, Dr. Matus Kalisky und Markus Heide für ihre sehr gute und freundliche Betreuung, für ihre stetige Hilfe bei allen Fragen, sowie für das Korrekturlesen.
Meinen weiteren Korrekturlesern Ulrich Baumeier, Linus Feldkamp, Paul Popp und Kai Sparenberg möchte ich zusätzlich danken. Eine physikalische Arbeit auf Englisch zu lesen und zu korrigieren ist gewiss nicht immer einfach.
Weiteren Dank gebührt Paul Popp, mit dem ich seit Beginn meines Studiums sämtliche Kurse, Übungsgruppen und Praktika geteilt und gemeinsam auch die stressigsten Zeiten des Studium gut überstanden habe.
Außerdem danke ich meinem Vater Ulrich Baumeier für die finanzielle Unterstützung, die mir erst mein Studium ermöglichte.
Und natürlich möchte ich noch meiner Freundin Nele Kieseler danken, für all die Zeit, die in keinsterweise irgendetwas mit Physik zu tun hatte und für das stetige Aufmuntern, wenn es mal nicht so lief, wie es eigentlich sollte.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe, und dass ich keine anderen Quellen und Hilfsmittel als die angegebenen benutzt habe und dass die Stellen der Arbeit, die anderen Werken - auch elektronischen Medien - dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Münster, 01. September 2011

David Baumeier