Time and Angle correlations between muon tracks inside the Antares neutrino detector

Menno de Bell
Supervisor: Ana Carolina Assis Jesus
National institute for subatomic physics

September 25, 2008

Abstract

The Antares neutrino detector is built for neutrino astronomy, i.e. looking for neutrino sources in the Universe. Examples of possible neutrino sources are Gamma-ray Bursts. In the case of Chocked Gamma-ray Bursts only neutrinos can escape the source. In our research we studied the possibility to detect these 'hidden’ sources with Antares. Data from Antares were analysed to search for neutrinos which are correlated in time as well as in direction. After applying several data filters only eight events were found which could have been caused by neutrinos. Of these events none were so close together in time that they could have been caused by the same GRB.
Contents

1 Author’s contribution 4

2 Introduction 6

3 Neutrino Astronomy 7
  3.1 Neutrinos as astronomical probes 7
  3.2 Detecting Neutrinos 7
    3.2.1 Upgoing versus downgoing muons 9

4 Gamma-ray Bursts 11
  4.1 Choked GRB’s 11

5 Antares Neutrino Telescope 12
  5.1 Filtering out background noise 12
  5.2 Reconstruction Strategy - ScanFit 12

6 Muon Tracks 15
  6.1 Preparatory work 15
  6.2 Selecting Tracks 16
    6.2.1 Associated Hits 17
    6.2.2 Track length 17
    6.2.3 Likelihood/probability 18
    6.2.4 Misreconstructed Upgoing Events 20
  6.3 Time Correlation between Tracks 22
  6.4 Angle Correlation between Tracks 22
  6.5 Visualizing MuonTracks 24

7 Conclusion and discussion 25

8 Populaire Samenvatting 26

A List of arguments 29

B Track Filters 29
  B.1 Associated Hits 29
  B.2 Tracklength 30
  B.3 $\chi^2$ Probability 31
  B.4 Upgoing Tracks 31

C Time Sorting 32
D  Root macro to visualize tracks  33
E  Main function of the program  36
1 Author’s contribution

- Acquired knowledge about the Antares Neutrino Telescope
- Acquired knowledge about (choked) Gamma-ray Bursts
- Acquired knowledge about using the root analysis package
- Developed algorithm for the analysis
- Developed a tool to visualize the events
2 Introduction

Ever since men have been looking at the stars, the information gathered about the Universe has come from electro-magnetic radiation, in particular from photons. Whether they come in the form of high energy gamma-rays, or as the less enigmatic (but not less important!) optic light, the detection of photons has been essential in our understanding of the Universe. Unfortunately many photons are scattered or absorbed before they ever reach our telescopes. Furthermore, some astrophysical objects emit too few photons to be detected. Because of this, additional methods have been devised to perform astronomy with.

The last couple of decades, people have looked at neutrino detection as a way to see beyond what photons can tell us. Neutrinos travel with speeds comparable to that of photons, but unlike photons, they only interact through the weak force. Because of this, their interaction cross section is small and they can travel through intergalactic dust clouds and galaxies relatively unhindered. Inconveniently, their small interaction cross section acts as a double edged sword. Although it helps us to see objects that we could not see otherwise, it also makes the neutrinos very hard to detect and enormous neutrino detectors are needed.

One of these detectors is Antares, located on the bottom of the Mediterranean Sea. In this paper we will describe our analysis of data acquired by Antares, with the goal to find neutrinos which arrive within a short time interval and from the same direction. Finding such a pair of neutrinos would be an indication of a neutrino source in that direction. We will take special interest in the hypothetical Choked Gamma-ray Bursts, which differ from regular Gamma Ray Bursts (GRBs) because they do not emit any gamma-rays. They are expected to be luminous neutrino sources.
3 Neutrino Astronomy

3.1 Neutrinos as astronomical probes

Conventional astronomy uses photons as messengers to give us information about the Universe. Photons make good astronomical probes because of their abundancy and their velocity, but have the handicap that they can be absorbed by interstellar material through the electromagnetic force. An alternative to photon astronomy would be to use neutrinos as probes, since they share the same traits that make photons so useful (like photons, neutrinos come in great numbers and travel with speeds close to the speed of light), but unlike photons, they have a very small interaction cross section, which enable them to get past intergalactic material relatively unhindered. Moreover, neutrinos have zero charge, which keeps their directional information intact, even after traveling through magnetic fields.

3.2 Detecting Neutrinos

Although the small interaction cross section makes neutrinos excellent astronomical probes, it also has its drawbacks. It makes them very hard to detect. Antares, like most large neutrino detectors, uses an indirect method which is based on the detection of muons, induced by neutrino interactions in the sea bed or the sea water inside the detector (figure 1).

Neutrinos interact with nucleons through the weak force, in two different ways [1]. It can happen through the exchange of either an electrically neutral particle (Z) or a charged particle (W+ or W-). When a charged interaction occurs between a neutrino and a nucleon (figure 2), the associated lepton \( l \) is created, together with a hadronic component \( X \):

\[
\nu_l + N \rightarrow l + X
\]

The lepton created is a charged particle and travels inside the sea water in the detector, with a speed higher than the speed of light in sea water. Because of this, the leptons emit Cherenkov light, a phenomenon analogue to the sonic boom created by an aircraft going faster than the speed of sound (figure 3). Contrary to neutrinos, Cherenkov light is relatively easy to detect, and if it is detected in a sufficient number of locations with good time precision it is possible to reconstruct the path of the lepton.

Depending on the flavour of the neutrino, the lepton is either an electron, muon or tau lepton. Of these three only the muon is able to travel large distances through the water, which makes this detection method best for muon neutrinos. The scattering angle \( \theta_s \) between the direction of the neutrino
Figure 1: After the interaction with a nucleon inside the seabed a muon is created and detected by its Cherenkov cone in the water.

Figure 2: Charged interaction between a neutrino and a nucleon N.
and the originating muon depends on the energy of the neutrino. For energies in the order of TeVs, the relation between $\theta_s$ and the energy $E$ of the neutrino is:

$$\theta_s = \sqrt{\frac{1.5 \text{ deg}}{E[\text{TeV}]}}$$

(2)

Thanks to the small scattering angle at high energies the directional information of the neutrino is not lost.

### 3.2.1 Upgoing versus downgoing muons

It would be ideal if muons were only created during cosmic neutrino interactions. Unfortunately, many muons originate from hadronic showers, which are created when cosmic rays hit the Earth’s atmosphere. These muons are called atmospheric muons and are the majority of the events detected by Antares. Fortunately, hadronic showers cannot travel through the Earth, and are always directed downwards. Neutrinos however have no difficulty traveling through the Earth, and therefore come from all directions. As a consequence, if a muon is detected going downwards, it is most likely to be an atmospheric muon, but when a muon is detected going upwards, it must be coming from a cosmic neutrino interaction. This is shown in figure 4, where the flux of atmospheric muons and neutrinos is shown as function of the zenith angle.

![Cherenkov cone](image)

**Figure 3:** Charged particle going faster than the speed of light produces a Cherenkov cone of light
Figure 4: Muon flux as function of zenith angle. Solid line caused by atmospheric muons, dashed line by neutrino induced muons.
4 Gamma-ray Bursts

Gamma-ray bursts (GRBs) are observed on Earth as intense flashes of gamma-rays. Although the cause of these flashes is not yet certain, it has been proposed that they are related to the collapse of a massive rotating star under its own gravitational forces. The energy created during the collapse is in the form of jets which consist of a plasma of accelerating charged particles. Among these particles are protons, and when these protons collide with the protons and neutrons inside the stellar remnants surrounding the collapsed core, pions are produced. These pions subsequently decay into gamma-rays and neutrinos.

\[
\begin{align*}
\pi^0 & \rightarrow \gamma + \gamma \\
\pi^+ & \rightarrow e^+ + \nu_e + \bar{\nu}_\mu + \nu_\mu \\
\pi^- & \rightarrow e^- + \bar{\nu}_e + \nu_\mu + \bar{\nu}_\mu
\end{align*}
\]

Because the gamma-rays can not travel through the stellar remnants, we only detect those that are created on the outskirts of the shell. However, neutrinos are able to escape from deep within the shell and this results in a neutrino signal preceding the gamma-ray flash in time.

4.1 Choked GRB’s

Some Gamma-ray burst models predict the existence of Choked GRBs [2]. In this GRB variant, the jets do not have enough energy to get past the stellar material and consequently, there is no visible gamma-ray burst, but there still will be a neutrino signal. Furthermore, since all protons inside the jet were in the position to interact with the stellar remnants, even more neutrinos are produced than in a regular GRB. It is estimated that a 1 km$^3$ detector will be able to detect 3 neutrinos from a choked GRB 10 Mpc away [2].
5 Antares Neutrino Telescope

The Antares Telescope [3] consists of 12 lines submerged 2.4 kilometres below sealevel in the Mediterranean Sea. Each of these lines contains 75 Photomultiplier Tubes (PMT), distributed evenly in groups of three (a storey). The purpose of these PMTs is to detect the Cherenkov light produced by muons. When a photon is detected by one of the PMTs (a hit), a signal is sent to the shore. This signal contains the time, position and amplitude of the hit. To reconstruct a muon track from these hits, a very high time resolution is needed. Hence every storey in the detector has its own clock, which is reset and synchronized every 416 ms.

5.1 Filtering out background noise

On average, a single PMT in the Antares Telescope detects 70,000 hits per second, but the hit rate can go up to rates of several Mhz. Considering that one line in the detector has 75 PMTs, then with 12 lines these signals constitute a stream of information of 0.4 GB/s [1]. Most of the hits come from random background, like bioluminescense and the radioactive decay of elements like $^{40}\text{K}$ in the sea water.

Once the data reaches the shore the hits from the different PMTs are combined and the data stream is cut into manageable pieces called timeslices. These timeslices contain the information of all the hits during an interval of 13 ms. Commodity PCs are used to analyse timeslices and look for clusters of hits which may be causally related to each other. When such a group of hits is found, all data between $2\mu$s before the first hit and after the last one is stored. Together these hits form an event, and all hits that do not belong to an event are assumed to come from random background and are discarded.

5.2 Reconstruction Startegy - ScanFit

The muon tracks that will be used in this paper are reconstructed by ScanFit [4]. There are five parameters needed to define a track, in the case of ScanFit they are the $\theta$ (zenith) and $\phi$ (azimuth) angles for the direction (defined as in figure 5), $a$ and $b$ for the position and $t_0$ for time.

ScanFit basically works as follows: it starts with starting values for the track parameters depending on the causally correlated cluster of hits, and then scans over the parameter space to find those parameter values which result in a good track fit.

Figure 6 shows the result of ScanFit: the program has scanned over the two direction parameters keeping the value of the other parameters constant
Figure 5: Definition of the zenith ($\theta$) and the azimuth angle ($\phi$)

Figure 6: ScanFits scans over the azimuth and zenith angle of the direction of the muon to look for maxima
and calculated the number of associated hits, as it will be explained in section 6.2.1, and the log likelihood (section 6.2.3) of the tracks. For every maximum that ScanFit finds, a track is reconstructed, although only one of them can be the true muon track.
6 Muon Tracks

For our analysis we used data from the Antares Neutrino Telescope from the period 30th of April until 4th of May in 2007, when the detector was operating with only five lines. The total run time was 66 hours. The data consists of two different classes: the 

```
PhysicsEvent
```


class, which contains all the information about an event, and the 

```
RIO::FitWithHitsIO
```

class with the muon tracks reconstructed by ScanFit and their hits [5].

Below we will describe the data analysis procedure. The first step was to write a program that reads the events and tracks from a file (as explained in section 6.1), selects those muon tracks which are suitable for neutrino astronomy (section 6.2) and then look for time and angular correlations between these tracks (section 6.3 and section 6.4). There is also an option to create a macro, which can be used to visualise the tracks going through the detector (section 6.5).

6.1 Preparatory work

To fetch the data from the rootfile [6], 

```
Reco_Reader
```

and 

```
PhysicsEvent_Reader
```

were used [5]. The data is fetched in two steps. First, 

```
GetEvent()
```

is called which loads the data of a certain event in the memory. Next, a pointer is created to the correct memory address which is found with 

```
GetAddress()
```

. Although a loop over all events is sufficient if only the properties of individual events (and their associated tracks) are to be analyzed, there are several problems to be solved when quantities of two different events are to be compared.

Because we want to look at the time difference between successive events it was necessary to sort all events in time. Since the raw data are not always stored in time order. The second hurdle was the way the Reader handled the data read from the root file. Everytime 

```
GetEvent()
```

was called, the information about the event replaced that of the previous event, and the pointer which pointed to the previous event suddenly pointed to the next one. This proved to be very cumbersome when trying to compare an event with a priori unknown number of other events, which is necessary if you want to compare all events within a certain time interval.

To cope with these problems, it was decided to loop over all events and put them with their tracks into the 

```
myEvents
```

vector. This is a vector with an 

```
EventAndFits
```

class, a newly created class which contained a 

```
PhysicsEvent
```

and a std::vector with the results of the track fits.
6.2 Selecting Tracks

During the 66 hours of runtime 96,219 events were registered by the Antares data acquisition system. Only a couple of these events are caused by atmospheric neutrinos while the rest of them are caused by either atmospheric muons or other background. In figure 7 the zenith angle distribution of up-going tracks with the best fit is shown. The distribution shows that there are thousands of upgoing events, from which we need to select only the very best. Because a good estimate of the neutrino direction is vital in neutrino astronomy, it is necessary to focus only on high quality tracks.

![Image of zenith angle distribution]

Figure 7: Histogram with the zenith angle of the best track of all events. Only upgoing tracks with a zenith angle $\theta < 90$ deg are shown.

There are several quantities that can be used to determine the quality of a track. First, the reconstructed muon tracks are filtered using a set of filters. These filters are called in the event loop before the events are put into the myEvents vector, and they are called with an EventAndFits and filter specific arguments, returning the event with all tracks that passed the restrictions. Only events with at least one track left are put inside the myEvents vector for further analysis.
6.2.1 Associated Hits

When ScanFit reconstructs the muon tracks, it does not use all hits of an event. Tracks with a small number of associated hits are more likely caused by random background or bioluminescence. Therefore, tracks with less than 12 associated hits were removed.

Sourcecode in appendix B.1.

6.2.2 Track length

The track length is a measurement of how long the detector was able to follow a muon. It is the distance between the origin of the first produced photon and the origin of the last produced photon. To calculate these locations, the coordinates of the hits are transformed from detector coordinates (x,y,z) to track coordinates (x’,y’,z’). The track coordinate system is obtained by rotating the detector coordinate system until the z-axis is parallel to the muon track as in figure 8. In this rotated coordinate system the track is determined only by the point (a,b) where it crosses the x’y’ plane and the time \( t_0 \) when this occurred.

![Figure 8: The detector coordinate system is rotated until the z-axis lies parallel to the muon track.](image)

To transform the coordinates of the hits to the track coordinate system the rotation matrix \( \mathbf{R} \) is used:

\[
\mathbf{R} = \begin{pmatrix}
\cos \theta \cos \phi & \cos \theta \sin \phi & -\sin \theta \\
-\sin \phi & \cos \phi & 0 \\
\sin \theta \cos \phi & \sin \theta \sin \phi & \cos \theta
\end{pmatrix}
\] (6)
Using the hit coordinates \( x'_h, y'_h, z'_h \) and the known Cherenkov angle \( \theta_c \) (42 deg in sea water), the z coordinate \( z'_e \) of the location where the photon was produced on the track can be calculated with:

\[
    z'_e = z'_h - \frac{r_h}{\tan \theta_c}
\]

where

\[
    r_h = \sqrt{(a - x'_h)^2 + (b - y'_h)^2}
\]

The tracklength \( \Delta z \) is then found by subtracting the smallest value of \( z'_e \) from the largest value

\[
    \Delta z = z'_{e,min} - z'_{e,max}
\]

Since the muons travel distances much longer than the size of the detector, we should be able to follow them from the moment they enter the detector up until they leave it on the other side. All tracks with a tracklength shorter than 180m were removed from the data sample.

Sourcecode in appendix B.2.

### 6.2.3 Likelihood/probability

The likelihood is defined as the product of probabilities of the individual measurements. In our case, these individual measurements are the hits, and their probabilities are the probabilities that they were produced by the passing muon. The time at which a Cherenkov photon is predicted to hit a PMT depends on the track parameters. The uncertainty on the predicted time follows from a Gaussian probability density function, centered around the predicted hit time \( t^h \) and with a width of \( \sigma (1.75\text{ns}) \) [4]. The likelihood function, depending on \( n \) hits with a hit time \( t_i \) and predicted hit time \( t^h_i \), can thus be written as:

\[
    L = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{1}{2}(\frac{(t_i-t^h_i)}{\sigma})^2}
\]

With the logaritmic of this function:

\[
    \ln L = l = -\frac{n}{2} \ln 2\pi - \ln \sigma + \sum_{i=1}^{n} -\frac{(t_i-t^h_i)^2}{2\sigma^2}
\]

Because a logarithm is a strictly increasing function, we see that a large likelihood is achieved when the sum is small. This sum can be written as a \( \chi^2 \) quantity.
\[
\frac{\chi^2}{2} = \sum_{i=1}^{n} \frac{(t_i - t_{ih})^2}{2\sigma^2}
\] (12)

The terms \((t_i - t_{ih})^2\) in the sum of \(\chi^2\) are expected to be normal random variables with a zero mean and a variance of one. With a given number of degrees of freedom \(n_{dof}\), \(\chi^2\) has a known distribution. In our case, the \(n_{dof}\) are the number of hits minus the five track parameters.

The \(\chi^2\) of a track can be found by calling \(L()\) on RIO::FitwithHits. Together with \(n_{dof}\) as arguments, the NR::prob() function can then be used to compare the track’s \(\chi^2\) with the known distribution, and return the probability that the terms in the track’s \(\chi^2\) are caused randomly.

<table>
<thead>
<tr>
<th>Reconstructed, Probabilities</th>
<th>recoprob</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entries</td>
<td>10264</td>
</tr>
<tr>
<td>Mean</td>
<td>0.4869</td>
</tr>
<tr>
<td>RMS</td>
<td>0.3437</td>
</tr>
</tbody>
</table>

Figure 9: Histogram of the \(\chi^2\) probabilities

Very low probabilities mean that the difference between measured and expected hit times are too large to be caused randomly. The peak at high probabilities in figure 9 is an indication that \(\sigma\) might be too large, and the real variance is smaller than one. In order to ensure high quality, tracks with a probability lower than 0.1 are discarded.

After the quality cuts on track length, associated hits and probability roughly 75% of all events were cut from the selection, but as can be seen in figure 10 the number of upgoing events was reduced more drastically. Only 115 upgoing tracks were selected for the rest of the analysis.

Sourcecode in appendix B.3.
6.2.4 Misreconstructed Upgoing Events

As mentioned in section 3.2.1, only upgoing muons are of interest, because only these are certainly the product of a neutrino interaction. One could select upward going muons by only selecting events with a single track with a zenith angle larger than 90 deg. However, since Scanfit will reconstruct multiple tracks for a single event, it is quite possible that among them are both upgoing and downgoing tracks. Therefore a simple cut on zenith angle is not possible. Instead we look at the log likelihood, \( l \), divided by the number of degrees of freedom \( n_{dof} \).

But even in the case that an upgoing track has the highest \( l/n_{dof} \), we can not throw away all other solutions. Due to statistical errors or other factors there is a possibility that the upgoing track is not the true track (for more information see [4]). Because of this we want to make sure that the fit quality of the upgoing track is not just better than that of the other tracks, but that it is better by a fair margin. To determine which upgoing track is to be considered the true track, the log-likelihood per \( n_{dof} \) of the best fitted upgoing track is compared with that of the best fitted downgoing track. If the difference was found to be smaller than 1, the upgoing track was assumed to be a misreconstructed downgoing muon and the event was discarded.

After this cut only eight events remained; as we can see in figure 11.

![Figure 10: Zenith angle of tracks after quality cuts on tracklength, hits and probability](image.png)
Figure 11: Zenith angle of tracks after both quality and misreconstructed cuts

Sourcecode in appendix B.4.
6.3 Time Correlation between Tracks

With the Antares telescope data are taken in runs which take approximately 5 hours. The data of these runs consists of frames, which constitute 8 consecutive time slices (explained in section 5.1) with roughly 105 ms of data. Every event carries information about the frame and run it resides in. The hit times and the $t_0$ of the tracks are saved in nanoseconds, and are relative to the last clock reset (explained in section 5), which happens every four frames.

To sort the events inside the myEvents vector, the sort() function is called with timeSort() (app. C) as the comparison function. This function compares subsequently RunNumber(), FrameIndex() and MaxT() of two events, where MaxT() gives the time of the last hit inside an event.

Once the events are sorted in time, the time between two successive events inside the same run can be calculated from the difference between their MaxT() and the time between resets. This quantity can be found with the CLOCK::getTimeOfRTS() function, which gives us the time between the start of the run and the last reset, given the index number of a frame. For internal run comparisons there are several functions that return the Universal Time Coordinates of a run or the seconds passed since january 1st, 1970 [7].

The time and direction of the eight events are shown in table 1. Of these events, two pairs of events were found to happen inside an 30 minute interval.

<table>
<thead>
<tr>
<th>Date and Time</th>
<th>Zenith angle</th>
<th>Azimuth angle</th>
<th>Runnumber</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mon Apr 30 21:36:25 2007</td>
<td>1.33775</td>
<td>2.31815</td>
<td>27501</td>
</tr>
<tr>
<td>Mon Apr 30 23:04:22 2007</td>
<td>0.282669</td>
<td>2.34867</td>
<td>27501</td>
</tr>
<tr>
<td>Wed May 2 19:48:04 2007</td>
<td>0.832323</td>
<td>3.12447</td>
<td>27549</td>
</tr>
<tr>
<td>Wed May 2 20:11:46 2007</td>
<td>0.779913</td>
<td>3.0299</td>
<td>27549</td>
</tr>
<tr>
<td>Thu May 3 21:50:39 2007</td>
<td>0.775742</td>
<td>2.3151</td>
<td>27565</td>
</tr>
<tr>
<td>Fri May 4 01:09:07 2007</td>
<td>1.45621</td>
<td>3.13289</td>
<td>27566</td>
</tr>
<tr>
<td>Fri May 4 01:36:50 2007</td>
<td>1.46837</td>
<td>1.98103</td>
<td>27566</td>
</tr>
<tr>
<td>Fri May 4 16:28:51 2007</td>
<td>1.08991</td>
<td>2.64152</td>
<td>27570</td>
</tr>
</tbody>
</table>

Table 1: The times and directions of the eight neutrino tracks

6.4 Angle Correlation between Tracks

The angle between two tracks $\psi$ is calculated by transforming the spherical coordinates of the tracks $\theta$, $\phi$ and $r (= 1)$ into Cartesian coordinates:
\[ x = \sin \theta \cos \phi \quad y = \sin \theta \sin \phi \quad z = \cos \theta \]  

(13)

Then, the inproduct between the two track vectors is taken

\[ \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y + a_z b_z \]  

(14)

From which the \( \psi \) angle can be found

\[ \mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \psi \]  

(15)

The zenith and the azimuth angles \( \theta \) and \( \phi \) are fixed for the detector, the detector is located at a fixed position on the Earth, and the Earth rotates through the Galaxy, so we cannot compare just any tracks, but only those sufficiently close in time.

The location of a neutrino source relative to the detector changes most noticeably because of the Earth’s rotation around its own axis. The Earth rotates 360 deg every 24 hours, or 0.25 deg per minute. In the best case, when the source is located on the rotation axis, there is no movement and neutrinos coming from this source will always traverse the detector from the same direction. In the worst case, where the source is located somewhere on the plane perpendicular to the rotation axes and at the height of the detector, the neutrino source seems to move with 0.25 deg per minute.

Because of this movement, the program only compares tracks that are maximally 2 minutes apart. Unfortunately, none of the eight tracks were this close together in time. Therefore, we could not compare their directions.
6.5 Visualizing MuonTracks

During the analysis it appeared useful to visualise how muons moved through the detector, and how PMTs detected its Cherenkov cone. To accommodate this, a root macro was written that created a TView (figure 12), where TPolyLine3D and TPolyMarker3D [6] were used to draw the detector lines and PMTs. The location of the lines and PMTs were determined by looking at all possible hit locations.

The main program, run with -y as argument, outputs the position of the hits, the point where the photons were produced (as in section 6.2.2) and the path of every track. The output is formatted in root commands which could be copied and pasted right into the macro. When the macro is then run inside Root, the extra commands draw the muon track and the path of the Cherenkov photons in the figure. Using a special filter we were able to identify events with a misreconstructed upgoing track (explained in section 6.2.4). Figure 12 shows such an event.

Figure 12: Visualization of an upgoing and downgoing track, reconstructed from the same event
7 Conclusion and discussion

Of the 96,219 events, only eight had an upgoing muon. None of these events were closer than 25 minutes apart, which meant that they could not be caused by the same Gamma-ray Burst. However, this does not exclude that they emerged from the same neutrino source. Unfortunately it was not possible to directly compare the directions of the tracks because of the rotation of the Earth. There are a multitude of programs that transform the horizontal coordinate system used by the Antares telescope to the Galactic coordinate system, which could be easily incorporated in future research. Another option to improve results is to look at more events, by either using looser quality cuts or, preferably, by using more data. Instead of the 5 line data used in this paper, one could also look at more recent data from Antares with all 12 lines operational. The visualisation tool developed appeared to be usefull in identifying misreconstructed tracks.

Figure 13: One of the upgoing muons traveling through the detector
8 Populaire Samenvatting

Op de bodem van de middelandse zee staat de Antares Telescoop. In tegenstelling tot de meeste telescopen kijkt Antares niet naar licht dat uit de ruimte komt, maar naar neutrino’s. Neutrino’s zijn deeltjes die uitzonderlijk weinig interacteren met andere deeltjes. Dit komt omdat neutrino’s zich niets aantrekken van de elektromagnetische kracht, die er bijvoorbeeld voor zorgt dat protonen, elektronen en photonen elkaar kunnen voelen.

Het voordeel hiervan is dat neutrino’s met gemak door materie heen kunnen reizen. Zo kunnen we bijvoorbeeld aan de hand van neutrino telescopie informatie krijgen over wat er allemaal gebeurt achter interstellaire stofwolken, op plaatsen die we eigenlijk niet kunnen zien. Om dezelfde reden kunnen we ook geen normale telescopen gebruiken omdat neutrino’s er dwars door heen vliegen.

Van de ontelbare hoeveelheid neutrino’s die per seconde door de Aarde heen reizen botst er eens in de zoveel tijd (dankzij de zwakke kernkracht) een neutrino toch met een proton of een neutron. Bij deze botsing verdwijnt het neutrino en komt er een muon vrij die in dezelfde richting doorreist. Een muon is een deeltje dat in principe een zwaar electron is. De muons die vrijkomen bij een dergelijke botsing laten een spoor van licht achter in het water.

De Antares telescoop bestaat dan ook uit een groot aantal licht detectoren die gelijkmatig zijn verdeeld over een volume van 3 miljoen kubieke meter. Door het licht van het muonspoor te meten kunnen we erachter komen waar het muon, en dus ook het neutrino, vandaan is gekomen. Omdat neutrino’s zo zelden botsen zijn er maar enkele muons nodig om met zekerheid te kunnen zeggen dat er zich in die richting een neutrino bron bevindt. Een dergelijke neutrino bron is de Gamma Ray Burst waarbij een zware ster onder zijn eigen zwaartekracht ineenvalt en voor enkele minuten meer licht in zijn eentje uitzendt dan de rest van het zichtbare universum bij elkaar.

Om uit de data die Antares verzamelt (”er is op dit moment op deze plek licht gedetecteerd”) bruikbare informatie te krijgen (”er is een neutrino gevonden dat op dit moment uit deze richting kwam”) moeten er een aantal obstakels worden genomen. Zo wordt veel licht dat wordt gedetecteerd door Antares veroorzaakt door andere processen zoals bijvoorbeeld lichtgevende vissen en radioactief verval. Daarbij komt ook nog dat muons ook vrijkomen bij zogenaamde atmosferische showers.

Hierdoor zijn de muon sporen die worden gereconstrueerd niet altijd even betrouwbaar. Door te selecteren op de kwaliteit van de sporen kan een groot aantal worden weggegooid die niet overtuigend zijn veroorzaakt door neutrino’s. Muons die bijvoorbeeld alleen over een korte afstand konden wor-
den gevolgd worden niet geselecteerd, en hetzelfde geldt voor muons die van bovenaf komen, omdat de kans groot is dat we dan met muons uit een atmosferische shower te maken hebben.

Van de 90 duizend keer dat Antares in 66 uur dacht iets gevonden te hebben waren alleen acht gebeurtenissen zonder twijfel veroorzaakt door een neutrino. De tijd tussen opeenvolgende neutrino’s was echter niet kort genoeg om te kunnen zeggen of ze door dezelfde Gamma Ray Burst waren veroorzaakt.
References


[7] Mieke Bouwhuis Universal times in Antares - the utc class as part of the jeep package
A  List of arguments

These arguments can be used for the main program. Only \(-a\) and \(-f\) are mandatory.

<table>
<thead>
<tr>
<th>flag</th>
<th>meaning</th>
<th>input type</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
<td>calibration file</td>
<td>string</td>
</tr>
<tr>
<td>-f</td>
<td>path to inputfile</td>
<td>string</td>
</tr>
<tr>
<td>-o</td>
<td>path to outputfile</td>
<td>string</td>
</tr>
<tr>
<td>-u</td>
<td>url</td>
<td>string</td>
</tr>
<tr>
<td>-i</td>
<td>upgoing tracks filter</td>
<td>bool</td>
</tr>
<tr>
<td>-p</td>
<td>minimum probability filter</td>
<td>double</td>
</tr>
<tr>
<td>-t</td>
<td>minimum tracklength filter</td>
<td>double</td>
</tr>
<tr>
<td>-m</td>
<td>minimum hits filter</td>
<td>int</td>
</tr>
<tr>
<td>-b</td>
<td>only use the best track</td>
<td>bool</td>
</tr>
<tr>
<td>-y</td>
<td>output visualization tracks</td>
<td>bool</td>
</tr>
<tr>
<td>-c</td>
<td>compare directions</td>
<td>bool</td>
</tr>
</tbody>
</table>

B  Track Filters

B.1  Associated Hits

EventAndFits filterEventsHits (EventAndFits preEvent, unsigned int minHits)
{
    EventAndFits postEvent;
    postEvent.event = preEvent.event;

    for (unsigned int i = 0; i < preEvent.fits.size(); i++)
    {
        if (preEvent.fits[i].hits().size() > minHits)
        {
            postEvent.fits.push_back(preEvent.fits[i]);
        }
    }

    return postEvent;
}
B.2 Tracklength

EventAndFits filterEventsTrackL (EventAndFits preEvent, double minTrackL)
{
    EventAndFits postEvent;
    postEvent.event = preEvent.event;
    double theta;
    double phi;
    double zout, minzout, maxzout;
    double x, y, z;
    double xj, yj, zj, a, b;

    for(unsigned int i = 0; i < preEvent.fits.size(); i++)
    {
        theta = acos(cos(preEvent.fits[i].theta()));
        phi = acos(cos(preEvent.fits[i].phi()));
        a = preEvent.fits[i].a();
        b = preEvent.fits[i].b();
        minzout = 100000;
        maxzout = -100000;

        for(std::vector<RIO::HitIO>::const_iterator
            h = preEvent.fits[i].hits().begin();
            h!= preEvent.fits[i].hits().end();h++)
        {
            x = h->x();
            y = h->y();
            z = h->z();
            zj = sin(theta)*cos(phi)*x +
                sin(theta)*sin(phi)*y +
                cos(theta)*z;
            xj = cos(theta)*cos(phi)*x +
                cos(theta)*sin(phi)*y -
                sin(theta)*z;
            yj = sin(phi)*x + cos(phi)*y;

            zout = zj - sqrt(pow(a-xj,2)+pow(b-yj,2))/
                 tan(42.5*PI/180);
            if(zout > maxzout)
                maxzout = zout;
    }
if(zout < minzout)
    minzout = zout;
}
if(maxzout - minzout > minTrackL)
    postEvent.fits.push_back(preEvent.fits[i]);
}
return postEvent;
}

B.3 $\chi^2$ Probability

EventAndFits filterEventsHP (EventAndFits preEvent,
    double minProb)
{
    EventAndFits postEvent;
    postEvent.event = preEvent.event;
    double thisProb;
    for (unsigned int i = 0; i < preEvent.fits.size(); i++)
    {
        try {thisProb = NR::prob(preEvent.fits[i].L(),
            (preEvent.fits[i].hits().size()-5));}
        catch(std::exception& error) {thisProb = 0;}
        if (thisProb > minProb)
        {
            postEvent.fits.push_back(preEvent.fits[i]);
        }
    }
    return postEvent;
}

B.4 Upgoing Tracks

EventAndFits filterEventsUp (EventAndFits preEvent)
{
    EventAndFits postEvent;
    postEvent.event = preEvent.event;
double theangle;
double bestUpLikely = -9999;
double bestDownLikely = -9999;
double thisLikely;

for (unsigned int i = 0; i < preEvent.fits.size(); i++)
{
    // ln(2pi) ~ 1.83, sigma = 1.75, ln(1.75) ~ 0.243
    theangle = acos(cos(preEvent.fits[i].theta()));
    thisLikely = -(1.83*preEvent.fits[i].hits().size()/2 +
                   0.243*preEvent.fits[i].hits().size() +
                   preEvent.fits[i].L()/2)/
                 (preEvent.fits[i].hits().size()-5);
    if (theangle < PI/2)
    {
        postEvent.fits.push_back(preEvent.fits[i]);
        if(thisLikely > bestUpLikely)
            bestUpLikely = thisLikely;
    }
    else if(thisLikely > bestDownLikely)
    {
        bestDownLikely = thisLikely;
    }
}

if(bestUpLikely - bestDownLikely < 1 )
{
    postEvent.fits.clear();
}

return postEvent;

C Time Sorting

bool timeSort(EventAndFits t1, EventAndFits t2)
{
    if(t1.event.RunNumber() == t2.event.RunNumber())
    {
        if(t1.event.FrameIndex() == t2.event.FrameIndex())
        {
            if(t1.event.StartTime() == t2.event.StartTime())
            {
                if(t1.event.EndTime() == t2.event.EndTime())
                {
                    return true;
                }
            }
        }
    }
    return false;
}
if(t1.event.MaxT() < t2.event.MaxT())
    return true;
    if(t1.event.FrameIndex() < t2.event.FrameIndex())
    return true;
    return false;
}  
else if(t1.event.RunNumber() < t2.event.RunNumber())
    return true;
    return false;
}

D Root macro to visualize tracks
{
    gROOT->Reset();
    gBenchmark->Start("detector");
    double PI = 3.141592653;

    // create and open a canvas
    sea = new TCanvas( "sea", "Detector", 300, 10, 700, 500 );
    sea->SetFillColor(36);

    // creating view
    TView *view = TView::CreateView(1,0,0);
    view->SetRange( -80, -20, -150, 80, 100, 210 );

    // draw a line of the detector
    // (other 4 lines removed to save space)

    TPolyLine3D *l1 = new TPolyLine3D();
    l1->SetPoint(0,-67.5332,63.4929,210);
    l1->SetPoint(1,-67.5332,63.4929,-250);
    l1->SetLineWidth(2);
    l1->Draw();

    // create the bottoms of the lines
    TPolyMarker3D *pm3d = new TPolyMarker3D( );
pm3d->SetPoint(0,-67.5332,63.4929,-250);
pm3d->SetPoint(1,-4.9336,82.043,-250);
pm3d->SetPoint(2,-18.833,22.8528,-250);
pm3d->SetPoint(3,67.3467,43.3228,-250);
pm3d->SetPoint(4,36.3164,-4.80713,-250);

// set their marker size, color & style and draw
pm3d->SetMarkerSize( 2 );
pm3d->SetMarkerColor( 1 );
pm3d->SetMarkerStyle( 22 );
pm3d->Draw();

// set position of all storeys
TPolyMarker3D *pm3d = new TPolyMarker3D( );
pm3d->SetPoint(0,-67.5332,63.4929,30.8025);
pm3d->SetPoint(1,-67.5332,63.4929,45.3135);
// 119 more pmt's...
pm3d->SetPoint(122,-67.5332,63.4929,103.437);

// set their marker size, color & style and draw
pm3d->SetMarkerSize( 1 );
pm3d->SetMarkerColor( 1 );
pm3d->SetMarkerStyle( 8 );
pm3d->Draw();

// create some floats
TPolyMarker3D *pm3d = new TPolyMarker3D( );

pm3d->SetPoint(0,-67.5332,63.4929,210);
pm3d->SetPoint(1,-4.9336,82.043,210);
pm3d->SetPoint(2,-18.833,22.8528,210);
pm3d->SetPoint(3,67.3467,43.3228,210);
pm3d->SetPoint(4,36.3164,-4.80713,210);

/* what happens below is actually outputted by the main program */

// set their marker size, color & style and draw
pm3d->SetMarkerSize( 2 );
pm3d->SetMarkerColor( 5 );
pm3d->SetMarkerStyle( 20 );
pm3d->Draw();

// show the hits
TPolyMarker3D *pm3d = new TPolyMarker3D( );
pm3d->SetPoint(0,36.3164,-4.80713,1.79541);
pm3d->SetPoint(1,36.3164,-4.80713,1.79541);
// 12 more hits..
pm3d->SetPoint(14,36.3164,-4.80713,74.4446);
pm3d->SetMarkerSize( 2 );
pm3d->SetMarkerColor( 2 );
pm3d->SetMarkerStyle( 3 );
pm3d->Draw();

// and show where the photons were produced
// (other 14 photontracks removed to save space)
TPolyLine3D *ptrack = new TPolyLine3D();
ptrack->SetPoint(0,42.8193,-61.9674,-6.1725);
ptrack->SetPoint(1,36.3164,-4.80713,1.79541);
ptrack->SetLineStyle(3);
ptrack->SetLineColor(3);
ptrack->Draw();

// creat the muon track
TPolyLine3D *track = new TPolyLine3D();
track->SetPoint(0,139.315,-263.564,-50.4523);
track->SetPoint(1,-114.793,267.315,66.1526);
track->SetLineStyle(1);
track->SetLineColor(1);
track->Draw();

/* up to here */

char timeStr[60];
gBenchmark->Show("detector");

Float_t ct = gBenchmark->GetCpuTime("detector");
sprintf( timeStr, "Execution time: %g sec.", ct);

TPaveText *text = new TPaveText( 0.1, 0.81, 0.9, 0.97 );
Muon traveling through the detector with detected cherenkov photons

timeStr

text->Draw();
sea->Update();
}

E Main function of the program

int main(int argc, char** argv)
{

    std::string calibration;
    std::string inputFile;
    std::string outputFile;
    size_t numberOfEvents;
    std::string url;
    double minProb;
    double minTrackL;
    bool useUpEvents;
    bool pickSpecial;
    bool outputPoly;
    bool compare;
    bool useBestTrack;
    unsigned int minHits;

    Parser<> zap;

    // Get inputFile and calibration and stuff

    try {
        zap['a'] = make_field(calibration);
        zap['f'] = make_field(inputFile);
        zap['o'] = make_field(outputFile) = "analyse_new.root";
        zap['n'] = make_field(numberOfEvents) = 0;
        zap['u'] = make_field(url) = "ant_read/kyoread@antares";
zap['p'] = make_field(minProb) = 0;
zap['i'] = make_field(useUpEvents) = 0;
zap['m'] = make_field(minHits) = 0;
zap['s'] = make_field(pickSpecial) = 0;
zap['y'] = make_field(outputPoly) = 0;
zap['c'] = make_field(compare) = 0;
zap['b'] = make_field(useBestTrack) = 0;
zap['t'] = make_field(minTrackL) = 0;

if (zap.read(argc, argv) != 0)
    return 1;
}
catch(const std::exception &error) {
    std::cerr << error.what() << std::endl;
    return 2;
}

// Create a root file
TFile* out = new TFile(outputFile.c_str(), "recreate");
out->SetCompressionLevel(1);
out->cd();

// Create a couple of histograms
TH1F eventtimedif("eventtimedif","Title",200, -1e09, 4e09);
// Other histograms removed to save space

// Connect to the (UTC Time) database
otl_connect::otl_initialize();
otl_connect db;

try{db.rlogin(url.c_str());}
catch(otl_exception& error){
    otl_print_error(error);
    return -1;
}
// UTC object and function calls
utc my_utc(db);

// Read from the inputfile and do something with
// the calibration
PhysicsEvent_Reader.Add(inputFile.c_str());
Reco_Reader.Add(inputFile.c_str());

TriggerInterfaceHelper interface(calibration);

// Look at how many events there are
numberOfEvents = (int) Reco_Reader.GetEntries();

// Make pointers to the events
PhysicsEvent* pev = 0;
RIO::RecoEventIO* rev = 0;

// Vector with all events and their fits
std::vector<EventAndFits> myEvents;
EventAndFits thisEvent;

// Loop over events and put them with their
// fits in the mighty myEvents vector
int usedEvents = 0;
int runTime = 0;
int morethanone = 0;
double lambda;
double likendof;
int ncomp;
double timedifference = 0;
double lastTime = 0;

for (unsigned int eventCounter = 0;
    eventCounter < (unsigned int) numberOfEvents;
    ++eventCounter)
{
    Reco_Reader.GetEvent(eventCounter);
    rev = Reco_Reader.GetAddress();
    PhysicsEvent_Reader.GetEvent(eventCounter);
pev = PhysicsEvent_Reader.GetAddress();

thisEvent.event = (PhysicsEvent) *pev;
thisEvent.fits = (std::vector<RIO::FitWithHitsIO>) *rev;

// Is the event good enough?
if(useBestTrack)
    thisEvent = filterEventsBest(thisEvent);
if(useUpEvents)
    thisEvent = filterEventsUp(thisEvent);
if(minHits !=0)
    thisEvent = filterEventsHits(thisEvent, minHits);
if(minTrackL !=0)
    thisEvent = filterEventsTrackL(thisEvent, minTrackL);
if(useBestTrack)
    thisEvent = filterEventsBest(thisEvent);
if(minProb != 0)
    thisEvent = filterEventsHP(thisEvent, minProb);

// Are there any tracks good enough?
// put it in the myEvents vector
if(thisEvent.fits.size() > 0)
{
    count_lfits += thisEvent.fits.size();
    myEvents.push_back(thisEvent);
    usedEvents++;
}

// Sort events by time
std::sort(myEvents.begin(), myEvents.end(), frameSort);

// Now we can do stuff with events in the vector

//
for (std::vector<EventAndFits>::const_iterator citer =
     myEvents.begin(); citer != myEvents.end(); ++citer)
{
    // Get some info from current Event.
    double MaxT = citer->event.MaxT();
    double frameTime =
        CLOCK::getTimeOfFrame(citer->event.FrameIndex());

    if(compare)
    {
        std::vector<EventAndFits>::const_iterator next = citer;
        ++next;
        if(next == myEvents.end())
            continue;

        // Calculate the time between this event and the next one
        double difFrameT =
            CLOCK::getTimeOfRTS(next->event.FrameIndex()) -
            CLOCK::getTimeOfRTS(citer->event.FrameIndex());
        double difMaxT = next->event.MaxT() - MaxT;
        int difRunT =
            my_utc.getRunStartEpoch(next->event.RunNumber()) -
            my_utc.getRunStartEpoch(citer->event.RunNumber());
        double difTime = difMaxT + difFrameT + difRunT*1e9;

        // Calculate if there are any anglecorrelations between
        // this track and any tracks within two minutes
        while(difTime < 120e9 && next != myEvents.end())
        {
            double angledif;

            for(unsigned int i = 0 ; i < citer->fits.size(); ++i)
            {
                for(unsigned int j = 0 ; j < next->fits.size(); ++j)
                {
                    angledif = angleBetw(citer->fits[i], next->fits[j]);
                }
            }
            next++;
            if(next == myEvents.end())
                break;
        }
    }
}
difTime = next->event.MaxT() - MaxT +
       CLOCK::getTimeOfRTS(next->event.FrameIndex()) -
       CLOCK::getTimeOfRTS(citer->event.FrameIndex());
}
}
}

// write to root files
out->cd();

out->Write();
out->Close();

return 0;
}