Forward Veto MARLIN processor for ILC beam calorimeter [*]

Marc Montull, DESY summer student program

October 18, 2008

Abstract

This report describes the work I did for the FLC group during the eight weeks of my stay at DESY as a summer student. I wrote a MARLIN processor for the reconstruction of energetic electrons and photons for the beam calorimeter of the ILC. In this document I explain the motivations for making this processor and its features, explaining what it does and how does it do it. I also review the checks made to see if things were right, and I point out a couple of things that should be looked into in order for the processor to be ready to use. At the end I give some suggestions about simple modifications that could be done to improve it.

1 Introduction

My work as a summer student consisted in writing a MARLIN processor able to determine which electrons or photons would be detected in the beam calorimeter of the ILC detector assuming the typical radiation background around the beam pipe caused by Beamsstrahlung.

The *forwardveto* processor is meant to be a tool for vetoing $\gamma\gamma$ background when trying to detect $SUSY$ particles with a small mass difference to the lightest supersymmetric particle from an electron positron collision. That means that it will try to reproduce the detector behavior for the detection of electrons.

To understand this let's take a quick look to the physical processes involved. There are for example SUSY models that under some reasonable assumptions predict that $\tilde{\tau}$ leptons could be produced in the ILC and then would decay like is shown in (1) with a $\sigma \approx$ fb.

$$e^+e^- \to \tilde{\tau}^+\tilde{\tau}^- \to \tau^+\tau^-\tilde{\chi}_1^0\tilde{\chi}_1^0 \tag{1}$$

In this case the signature of this process would be like is shown in (2) since neutralinos can't be detected.

$$e^+e^- \to \tau^+\tau^- + missing\ energy \tag{2}$$

If this process was the only one with this kind of signature there wouldn't be a problem detecting the $\tilde{\tau}$ leptons. Unfortunately there is a Standard Model process with a much higher cross section ($\sigma \approx$ nb) that can give the same signature if its final electrons are not tagged. This process is what we call $\gamma\gamma$ background, and it shown in (3).

$$e^+e^- \to e^+e^-\tau^+\tau^- \tag{3}$$

It's easy to see that if we don't detect the outgoing electrons we will have the same signature for (1) and (3), and then we won't be able to distinguish them. To solve this we are going to try to tag all the outgoing electrons of (3).

[*]Supervisors: Jenny List, Mikael Berggren

If we look at the Feynman diagram of the $\gamma\gamma$ process (figure 1) we see that each electron emits a virtual photon doing it through a t-channel.

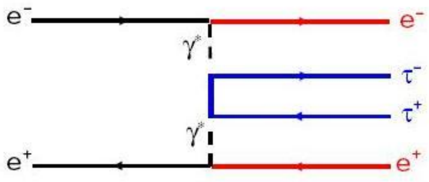

Figure 1: Feynman diagram for the $\gamma\gamma$ background process.

We know that when we have a t-channel usually the outgoing particle (in our case the electron) acquires very little transverse momentum. This means that the outgoing electrons will either hit the beam calorimeter or escape the detector through the beam pipe, and in both cases with a very high energy ($E_{electron} \geq 60$ GeV approximately). We can see now that in order to veto the $\gamma\gamma$ background we need to detect as many of this energetic electrons as possible, and that's why we need to make use of the beam calorimeter. To have an idea of what we mean, in figure 2 we can appreciate the difference between the signal we would get in the detector from the $e^+e^- \rightarrow \tau^+\tau^- + missing\ energy$ process and the signal result of applying different cuts and the electron veto to it (done with a very simplified simulation program).

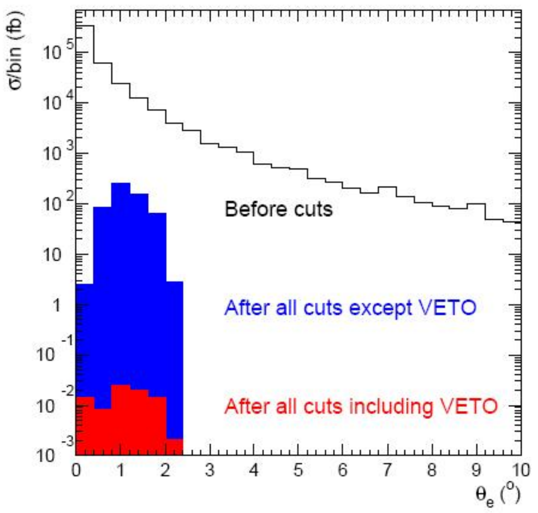

Figure 2: Angular distribution of the spectator electrons of $e^+e^- \rightarrow e^+e^-\tau^+\tau^-$ expressed in fb/bin. The shaded shaped show the distributions after applying various cuts, and after applying these cuts and the electron veto.[1]

2 Forwardveto processor

In this section I'll explain the *forwardveto* processor works, and how it should be used.

2.1 Forwardveto overview

The *forwardveto* processor reads a particle collection produced at the interaction point and gives a reconstructed particle collection with the electrons (or photons) that have been detected in the beam calorimeter. To do so it uses four different routines that serve different purposes. These are propagating each particle from the interaction point (IP) to the beam calorimeter, summing the energy density of beamsstrahlung radiation for each point were the particle goes through in the detector, parametrizing the efficiency of the detector and using a simple Monte

Carlo method to see whether the particle is detected or not. Some parameters used by these routines can be modified through the steering file.
In figure 3 we can see schematically the main steps that this processor follows to see if a particle is detected or not in the beamCal.

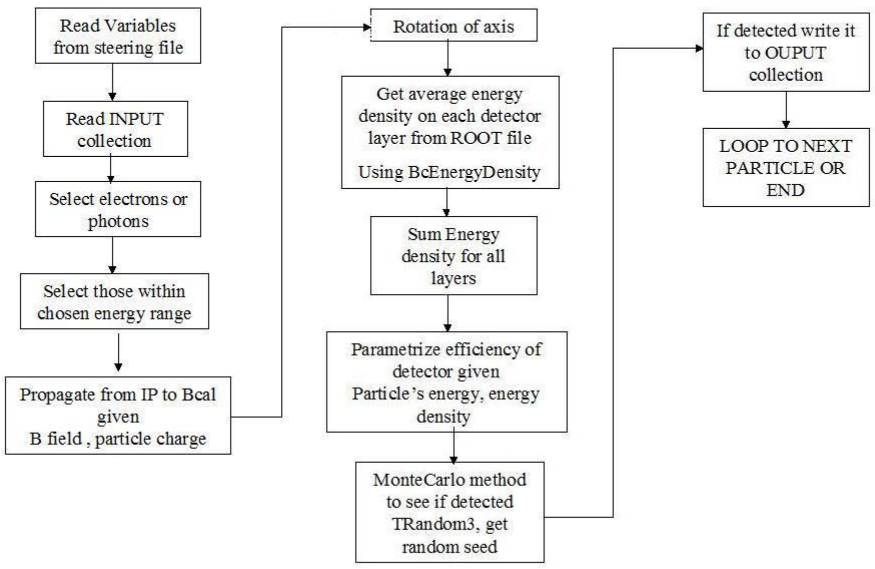
Read Variables from steering file
Read INPUT collection
Select electrons or photons
Select those within chosen energy range
Propagate from IP to Bcal given B field , particle charge
Rotation of axis
Get average energy density on each detector layer from ROOT file Using BcEnergyDensity
Sum Energy density for all layers
Parametrize efficiency of detector given Particle's energy, energy density
MonteCarlo method to see if detected TRandom3, get random seed
If detected write it to OUPUT collection
LOOP TO NEXT PARTICLE OR END

Figure 3: Schematic overview of the *forwardveto* processor flow.

Another feature of this processor is that by default it creates a *root* tree with the properties of each detected particle such as momentum, energy or position when hitting the beam calorimeter.

2.2 Processor parameters

Marlin applications are entirely configured through XML steering files. The XML files hold arbitrary named parameters - defined either for a particular processor, a group of processors or globally and also define the order in which the processors are called as well as optionally logical conditions assigned to some processors that are evaluated at runtime.

The *forwardveto* processor has some default parameters that can be modified from the steering file. These are shown in table 1.

Parameters	Default value	Units	Description
B	3.5	Tesla	B-field (direction along z axis)
EB	250	GeV	Energy of each electron beam
zbcal	3000	mm	Distance from IP to beamCal
thresholdMin	40	GeV	Minimum energy ”processed” particles can have
thresholdMax	10000000	GeV	Maximum energy ”processed” particles can have
detectAll	(not enabled)	-	Detect all electrons or photons within selected energy range that hit the beamCal
Electron/Photon	Electron	-	Detects electrons (if Electrons) or photons (if Photons)

Table 1: Table of the *forwardveto* parameters.

2.3 Methods of the processor

This processor was based on an example processor found at /afs/desy.de/group/it/ilcsoft/v01-03-06-p02/Marlin/v00-10-03/examples/mymarlin. This means that it uses the standard methods for a MARLIN processor.

virtual void init()
virtual void processRunHeader(LCRunHeader* run)
virtual void processEvent(LCEvent* evt)
virtual void check(LCEvent* evt)
virtual void end()

The only methods I've modified from the example are the init(), processRunHeader(LCRunHeader* run), processEvent(LCEvent* evt) and end(). The processEvent(LCEvent* evt) is the one where all the steps of the processor are made, so it was extensively modified. In the other three processes I only added what needed to create a *root* tree.

2.4 virtual void processEvent(LCEvent*)

We can see the flow diagram of how the processor works in figure 3, and besides the first box all the other ones happen when the processEvent(LCEvent*) is called. In order for any potential user to be able to know how things work the most important steps will be explained in detail.

2.4.1 Propagation of particles from IP to beamCal

bcalhit(pin,q,vin,B,EB,zbcal,pout,vout) is a FORTRAN routine made by Mikael Berggren. This routine tracks a particle with momentum $\vec{p} = (p_x, p_y, p_z)$ from an initial point (x, y, z), to the plane surface at the z desired coordinate. To do this it also uses the initial energy of the particle, the magnetic field in the detector and the particle's charge. In tables 2 and 3 we can see the input and output parameters of this routine.

Parameter	Units	Type	Description
pin	GeV	float[3]	Initial particle momentum $\vec{p} = (p_x, p_y, p_z)$
q	electron charge	float	Charge of the propagated particle
vin	mm	float[3]	Initial particle position set at (0,0,0) by default
B	Tesla	float	B-field (direction along z axis)
EB	GeV	float	Energy of each electron beam
zbcal	mm	float	Distance from IP to beamCal

Table 2: Table of the bcalhit routine input parameters.

Parameter	Units	Type	Description
pout	GeV	float[3]	Particle momentum at zbcal
vout	mm	float[3]	Particle position at zbcal

Table 3: Table of the bcalhit routine output parameters.

2.4.2 Rotation matrix

At the ILC, the beams will collide under a small crossing angle of 14mrad. The particle trajectories are extrapolated in the coordinate system of the main detector, who's z axis points in the middle between the incoming and outgoing beam. The beam calorimeter on the other hand is centered around the outgoing beam. The average energy density expected from beamstrahlung is given in the local coordinate system with a z axis parallel to the outgoing beam. In order to go from one coordinate system to another a rotation matrix is used. This matrix rotates the x and z axis around the y axis a certain angle. In my case I knew that I had to rotate it 7 mrad. I think that I did it correctly but as I'll say in a following section this should be double checked just in case.

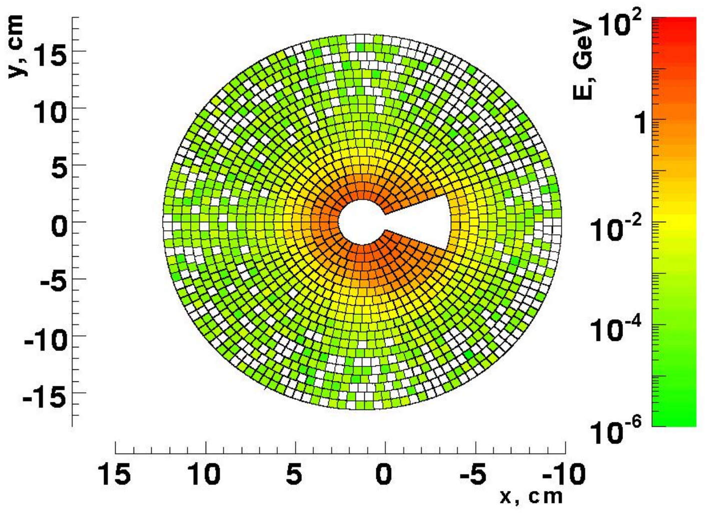

Figure 4: Plot of the distribution of energy density for an unknown layer of the beam calorimeter that was given as an example with the BcEnergyDensity class.

2.4.3 Get energy density in the beam calorimeter

BcEnergyDensity is a C++ class made by A. Sopranov. This class includes the method *bool GetEnergyDensity(layer,radius,phi,en_dens,en_dens_err)* that given the polar coordinates on the beam calorimeter and the layer it gives the energy density and its error in that location. *GetEnergyDensity* gets the energy density in the beam calorimeter from a root file called *bg_aver.root*, and using this estimates the energy density in any location of the beam calorimeter. If this operation is successful the value of *GetEnergyDensity* is 1, if it isn't, the value is 0 which usually means that the particle didn't hit the beam calorimeter. This class is meant to be used for an LDC magnetic field map.

The way this method is used in the processor is giving it the coordinates of each studied particle after being propagated from the IP with the *bcalhit* routine (and after changing the system of coordinates with the rotation matrix), and then getting the energy density in each layer at this coordinates and adding each of them up. This method supposes that the beam calorimeter has 30 layers but there's no shower in the first one. The method differentiates between the beam cal in the positive z direction and the one in the negative, and the way to tell it is assigning positive integers to the layers in the positive z direction and negative integers to the layers in the negative z direction.
In tables 4 and 5 we can see the input and output parameters of this method.

Parameter	Units	Type	Description
layer	-	int	Layer were the energy density is calculated ([1,30] or [-1,-30])
radius	mm	double	Radial position of the particle
phi	rad	double	Angular position of the particle [0,2ϕ]

Table 4: Table of the GetEnergyDensity method input parameters.

Parameter	Units	Type	Description
en_dens	GeV/mm^3	double	Energy density at given location in beamCal
en_dens_err	GeV/mm^3	double	Error of en_dens

Table 5: Table of the GetEnergyDensity method output parameters.

2.5 Electron recognition efficiency

float HBEER(Ebg,Eel) is a C function made by A. Drugakov and E. Kousnetzova. This function returns the efficiency of electron recognition of the beam calorimeter of the ILC given the total energy background in every layer the electron has gone through and the energy of this electron when first hit the beam calorimeter.

Parameter	Units	Type	Description
Ebg	GeV/mm^3	double	Energy density at given location in beamCal (summed over all layers)
Eel	GeV	float	Energy of the electron when hitting the beamCal

Table 6: Table of the GetEnergyDensity method output paremeters.

2.6 Determination of detection

In order to see if an electron (or a photon) has been detected we use a simple Monte Carlo method.
The HBEER function gives us a parametrization between 0 and 1 of the efficiency of electron recognition for each electron that goes through the beam calorimeter. This means that we can use a random generator to generate numbers between 0 and 1 every time an electron goes through the beam calorimeter to see if it is detected. The way to do this is comparing the value of number given by the random generator and the efficiency given by the HBEER function, and taking an electron as detected if the number given by the random generator is smaller than the one given by the HBBER function.

The random generator used has been the one given by the TRandom3() class of root. This class has a method called double_t Rndm(int_t i=0) and it produces uniformly distributed floating points in (0,1]. In order for this number to be different every time it's used we give it a random seed as well. We've chosen to give as seed the number created by the fourth to the eight digits of the energy of each electron.

2.7 Root trees

In order to know how the processor behaves I thought it would be a good idea that every time the processor runs it creates a tree to a *root* file. The processor creates two trees one called Treename (for electrons) and the other Treename2 (for photons). In these trees all the interesting to analyse the processor are stored. The values are shown in tables 7 and 8.

Name of variable	Units	Description
renergy	GeV	Energy of the electron when hitting the beamCal
rposx	mm	Position of the electron in x axis when hitting the beamCal
rposy	mm	Position of the electron in y axis when hitting the beamCal
rpos	mm	Position of the electron in z axis when hitting the beamCal
rRadius	mm	Radial position of the electron when hitting the beamCal
rPhi	radians	Angular position of the electron when hitting the beamCal angles from $[-\pi,\pi]$
rEfficiency	-	Electron efficiency parametrization of the HBEER function from [0,1]
rMC	-	Value of the TRandom3() method Rndm() from (0,1]

Table 7: Table of values stored in the *root* tree Treename (tree for electrons).

Name of variable	Units	Description
rpenergy	GeV	Energy of the electron when hitting the beamCal
rpposx	mm	Position of the electron in x axis when hitting the beamCal
rpposy	mm	Position of the electron in y axis when hitting the beamCal
rppos	mm	Position of the electron in z axis when hitting the beamCal
rpRadius	mm	Radial position of the electron when hitting the beamCal
rpPhi	radians	Angular position of the electron when hitting the beamCal angles from $[-\pi,\pi]$
rpEfficiency	-	Electron efficiency parametrization of the HBEER function from [0,1]
rpMC	-	Value of the TRandom3() method Rndm() from (0,1]

Table 8: Table of values stored in the *root* tree Treename2 (tree for photons).

3 Checks made

To see if the processor worked properly I made three checks to it, one of them was successful, one showed some problems, and the third one should be redone.

3.1 Output collection

I created a new MARLIN processor that would read the *reconstructed particle* collection from the *forwardveto* processor to see that the output collection it was giving could be read wihtout problems. This check worked properly.

3.2 Detector efficiency distribution

To see that the processor was doing things properly I plotted the efficiency of the detector against the radial position for particles with different energy ranges. I did this because the efficiency of a the beamCal has to go somehow as shown in figure 5.

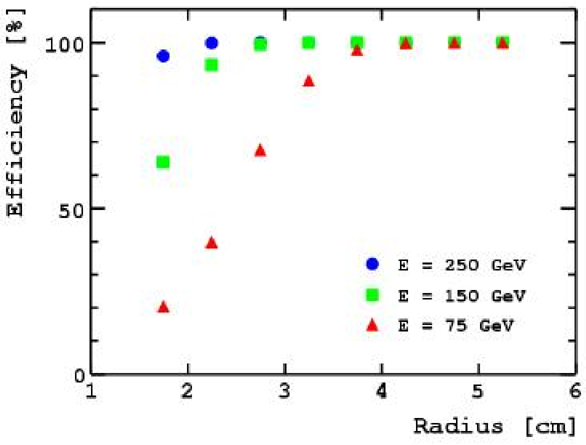

Figure 5: Scheme of how should approximately the efficiency of the detector vary for different radius and energies.[2]

When I performed this check I got that the efficiency for all the particles was around 0.98 no matter of its position or energy. This meant that there was something wrong either with the *GetEnergyDensity* (or the way I used it) or with the parametrization done by the *HBBER* function. No serious further checks were done to this, but just to see if the problem came from the energy density background given to the *HBBER* function I plotted the efficiency against the radius for the case where the energy density given to the *HBBER* was multiplied by 10 and also for the case it was multiplied by 100. At first sight the plot where the energy density was multiplied by 100 looked as it should for different energies. Even though some serious checks should be done to this. It could be that the a problem with the units of the energy background density.

3.3 Check change of coordinates

To see if the change of coordinates was done properly I tried to plot all the electrons hitting the beam calorimeter on the xy plane to see if they somehow would appear drawn in some "forbidden" region (like in the beam pipe). The problem with this check is that I didn't see any point close enough to the beam pipe to tell weather the change of coordinates was done properly or not. Another problem I had when doing this check is that I was trying to see this "forbidden" regions from a slice of the beam calorimeter (see figure 4) that was given in the same folder I received the *BcEnergyDensity*, but the problem was that I wansn't 100% sure that it was from the beamCal situated in the positive part of the z axis. That's why even though I think that the change of variables I used with the rotation matrix (section 2.4.2) was alright it'd be wise to double check it.

3.4 Problems to be solved

There are two main problems that need to be solved in order for this processor to be ready. The first one is concerning the problem I mentioned in section 3.2 concerning the detector efficiency distribution.
The second problem or thing that should be checked is if the HBBER function also applies to photons, so to be sure that it is can be used in the case the processor works detecting photons.

4 Suggestions

I think that after fixing these two problems the processor will be ready to be used. The only thing that could be changed is some variable names to be consistent with the coding conventions followed in MARLIN.

When creating this processor I linked it to many different libraries, and I don't know if all of them are needed. In order to make the processor "efficient" in this sense I'd recommend to check in the *CMakeLists.txt* file which of the are needed, and I'd modify it so when downloaded to use it doesn't complain about needing something unneeded.

5 Conclusions

The overall result of this project was positive even though I finished with some unsolved problems that I have to leave to my supervisors to fix. Even though I think that these won't require almost any further coding, and will only require the understanding of some of the routines involved. That's why I hope that without too much effort this processor will be able to run and help do some physics.

Aside from the job done I would like to thank my supervisors and all the people in the group that helped me when I had troubles with my work and helped me. I think this was a positive experience and I learned many valuable things. I'm very grateful to everybody in the FLC group and I hope for them the best luck with everything !!!
Best wishes !!

6 References

[1] P. Bambade, M. Berggren, F. Richard, Z. Zhang. arXiv:hep-ph/0406010v

[2] Bozovic-Zelisavcic, *The Forward Calorimetry at ILC* talk