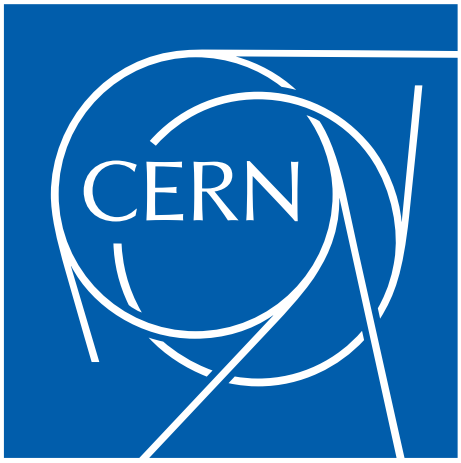
TU/e technische universiteit eindhoven



CERN project Sjoerd Loenen - CTPPS

Project Report

August 30, 2017

Student: S.J.H. Loenen
Supervisor: Dr. Martijn Mulders
Supervisor: Dr. Laurent Forthomme

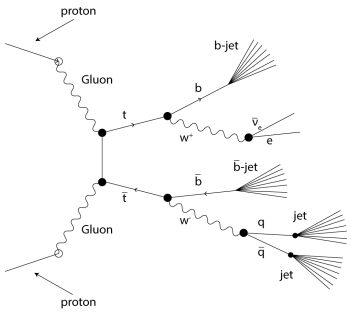
Contents:
-) Report on project (Ch. 1-7)
-) Documentation of main script used to analyze data (Ch. 8)

1 Introduction

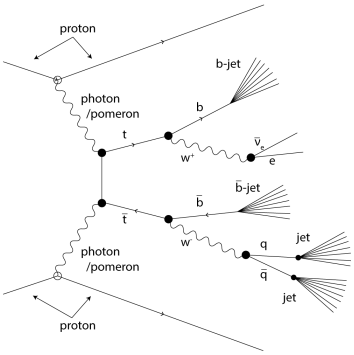
The main goal of this project has been to do a feasibility study on the usage of forward protons in combination with the central system of a $t\bar{t}$ decay to achieve the same information by two different measurements. One of these measurements is CMS, the other is TOTEM. TOTEM roughly consists of two arms approximately 200 meter on both sides along the beam line of the CMS detector. Both arms contain a number of Roman Pots, which are able to detect forward protons that meet certain requirements. The experimental setup discussed in this report contains two roman pots.
Since TOTEM only detects the forward protons it does not suffer from as much background as CMS does. This leads to cleaner measurements, which is one of the motivations for using TOTEM. Important to stress is that the events that it measures are exclusive events, which means that the protons stay intact after the collision. If after the feasibility study it seems that TOTEM is working properly, verifications of the Standard Model (SM) or anomalous models that explain phenomena beyond the SM can be performed. One of the examples could be the profile of the invariant mass of the $t\bar{t}$ system, where one is able to compare the SM prediction to experiments. Another promise is the possibility to determine the cross section of exclusive $t\bar{t}$ production.

2 Event production

This report elaborates on the study of the invariant mass of a $t\bar{t}$ system produced by the collision of protons in the LHC. The productions of particles in proton-proton collisions can be categorized in two groups. One of them is non-exclusive production and the other is exclusive production. In non-exclusive production at least one of the protons falls apart after creating the $t\bar{t}$. In the exclusive production both the protons stay intact after their collision.
The most well known form of $t\bar{t}$ production is the non-exclusive production. An example is shown in Figure 1a, where the protons exchange a gluon whereby color is not conserved. Therefore the proton will decay as well and cannot be measured in the end.



(a) Non-exclusive production of $t\bar{t}$ production



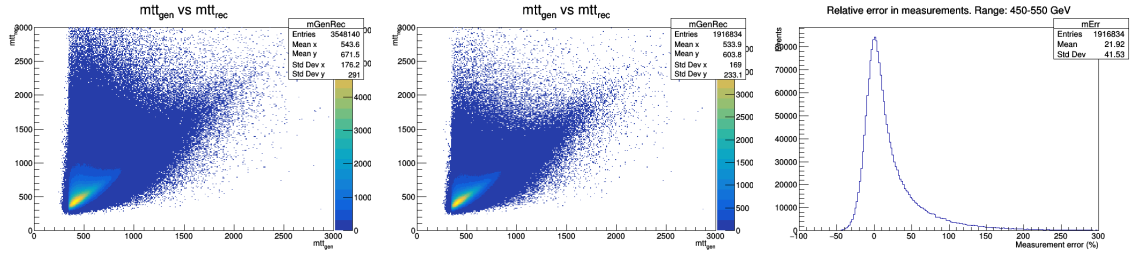
(b) Exclusive production of $t\bar{t}$ production

The other way of producting a $t\bar{t}$ system is the exclusive production, see figure 1b. In this case the protons exchange a photon or a pomeron wherefore the color is conserved and the protons stay intact. This assures that only a $t\bar{t}$ system is created. Therefore there is much less background compared to the non-exclusive decay in which the protons decay to jets. Exclusive production allows the protons that produced the $t\bar{t}$ system to be measured. This

is exactly where TOTEM relies. This project is aimed at the exclusive production of a $t\bar{t}$ system.

3 MC and CMS comparison

In order to test the degree to which the roman pots measure the $M_{t\bar{t}}$ correctly, their measurements will be compared with the measurements done in CMS. This means CMS also needs to be tested whether it measures correctly. To that end Monte Carlo simulations have been performed. In these simulations protons are collided which produce $t\bar{t}$ systems, these systems decay to their final state products (for example as in Figure 1b). Next to the collision of the protons the measurements of CMS are also included in the simulations. So by using the measured final decay products in the simulation one is able to reconstruct the $M_{t\bar{t}}$ as measured in CMS and compare this with the $M_{t\bar{t}}$ as initially defined in the Monte Carlo simulation. If CMS would be perfect, there would be a one to one correlation between MC and CMS and when plotting the invariant masses measured in both of them against each other their should be a linear line with slope 1 through the origin.
After performing the simulations and comparing the $M_{t\bar{t}}$ of CMS to $M_{t\bar{t}}$ of MC, there is a slight indication of the expected linear line, however, there still is a lot of deviation from that, see figure 2a.



(a) $M_{t\bar{t}}$ of CMS to MC without any event selection

(b) $M_{t\bar{t}}$ of CMS to MC with an event selection

(c) $M_{t\bar{t}}$ of CMS to MC with an event selection

This indicates CMS is not performing perfectly and improvements need to be made. One possible way of improving is using a stricter event selection, so that not only certain events are taken into account that might have less deviation from the expected behavior. The event selection that has been applied here is to only use these events that have 4 jets, whereof 2 b-jets, and 1 lepton as final state product (as shown in figure 1b). Results of this method are shown in figure 2b, which shows improvements compared to figure 2a. However, still more research is needed to make further improvements. What is also to be seen is that the deviation is different in different energy regions of $M_{t\bar{t}}$, this is shown in figure 2c which shows the relative deviation in the energy region of 450-550 GeV. From this a mean deviation and a rms in this deviation can be determined, which can be used to "de-bias" the CMS measurements and apply an uncertainty on them.
In this project no further research on this aspect has been performed. However, suggestions could be made. Some things one could consider is using a stricter P_t and η cuts. At the moment there are no cuts applied to the measured leptons. To the jets a minimal P_t cut of 30 GeV is and a maximum η cut of 2.4 is applied. Besides one could look at which jets have been used for the reconstruction of the $t\bar{t}$ system. Since in this research the four most energetic jets have been used, but it could be that one of them did not come from the $t\bar{t}$ system, which could explain the overshoot in the measurement of CMS compared to MC. The way to do this is to make advantage of the capabilities of CMS to track back from which

vertex the jets came. By looking at figure 1b, one sees that all final state products need to come from two main vertices.

4 From TOTEM to invariant mass

As explained TOTEM measures the forward protons. Currently TOTEM only measures the positions of the protons. A time detector is scheduled for the future, but is not in operation yet. From these positions one can calculate the longitudinal momentum losses the protons have experienced. This momentum loss is defined as $\xi = \frac{\Delta P_z}{P_{tot}}$, where ΔP_z is the difference in longitudinal momentum before and after the collision and P_{tot} is the total momentum before the collision.
This ξ parameter allows to calculate the invariant mass of a produced $t\bar{t}$ system. Therefore one needs three main ingredients, equation (4.1), (4.2) and (4.3);

$$E^2 = P^2 + m^2 \tag{4.1}$$

$$P_z \gg P_x, P_y \tag{4.2}$$

$$\xi \ll 1 \tag{4.3}$$

$$E_{t\bar{t}} = \frac{1}{2}\sqrt{s}\,(\xi_0 + \xi_1) \tag{4.4}$$

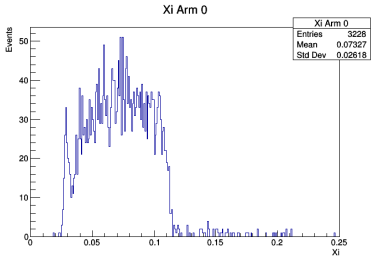
$$P_{t\bar{t}} = \frac{1}{2}\sqrt{s}\,(\xi_0 - \xi_1) \tag{4.5}$$

Equation (4.1) is exact, whereas the other two are approximations. Since the protons in the LHC are very forward, equation (4.2) is satisfied. Besides it is expected that equation (4.3) is satisfied. This will either be verified or denied by the experiments. As it turns out ξ is around 10%, which is assumed to be small enough. Combining these three equations and performing some algebra equations (4.4) and (4.5) could be derived. In these equations ξ_0 is the momentum loss of the proton moving one direction and ξ_1 is the one of the proton moving in the opposite direction. By inserting these back into equation (4.1) it is derived that

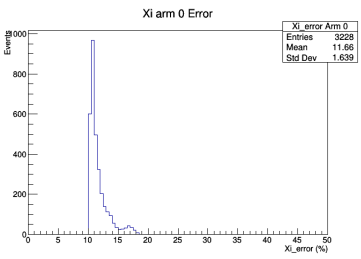
$$M_{t\bar{t}} = \sqrt{s\xi_0\xi_1} \tag{4.6}$$

5 TOTEM Measurements

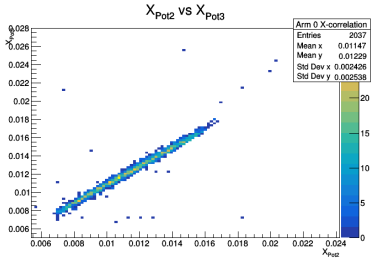
From the previous section it is known that xi, the momentum loss, is the main quantity that needs to be extracted from the roman pots in TOTEM. By measuring the positions of the forward protons, this quantity is calculated. Results are shown in figure 4a. It shows that the momentum loss is between 5 to 15 percent, which was assumed to be small enough for equation (4.3) to be valid. While the acceptance of the roman pots is at maximum around 15%, some measurements of more than 20% have been measured. These measurements cannot be real. What possibly explains them is a large error in the measurement. But if we show the error in the measurements, figure 4b, one sees that these errors are too small to explain the xi values above 20%.



(a) ξ values measured in one arm of TOTEM



(b) Relative error in ξ measurements

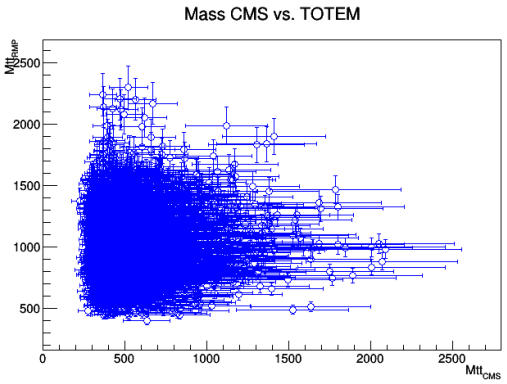


(c) x-position of track in pot one vs x-position in pot two

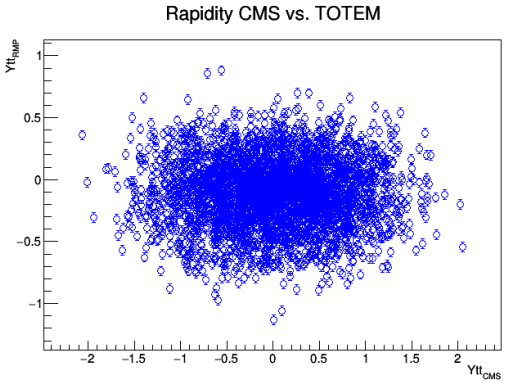
Another possible explanation for the unreal measurements are possible tracks not coming from forward protons. Whether these occur can be investigated by plotting the measured horizontal positions of the proton in the first roman pot versus the second roman pot. Since these pots are very close to each other, these positions should be (nearly) the same. This measurement obviously requires that the proton leaves a track in the first as well as the second pot. Results have been shown in figure 3c, which shows that there are some tracks not coming from protons. Ideally one would like to remove all these tracks from the measurements. This is however not possible, since for tracking non-proton tracks measurements of the first as well as the second pot are required, while in the measurement of xi also protons only leaving tracks in one pot are taken into account. Thus, part of them can be removed, but not all of them.

6 CMS and TOTEM

Since the xi-values have been measured, it is possible to construct the invariant mass of a $t\bar{t}$ system. Since also the uncertainties in the xi measurements are known, they can be taken into account. Also taking into account the mean deviation and the rms in this mean, as calculated from figure 2c, figure **??** is produced. Note that here symmetric error bars have been used, while in reality the error in the CMS measurements are far from symmetric around their mean (see 2c).



(a) $M_{t\bar{t}}$ of CMS vs. TOTEM



(b) Rapidity of CMS vs. TOTEM

Still there is a general overshoot to be seen. This suggest further research in the reconstruction of TOTEM. It might be that two tracks in different arms are assumed to come from the same collision. However, if the do not come from the same collision a large error can be made. This error is however expected to give both undershoots and overshoots in

the invariant mass of TOTEM compared to CMS. Still it is wise to be able to distinguish whether protons came from the same collision or not. With the introduction of the timing detector on TOTEM, this will become possible to a certain extend.
Besides the invariant mass of the $t\bar{t}$ system, one could also look at the rapidity of TOTEM vs CMS and CMS vs MC. In this project the main focus has been on the invariant mass, but some research has been done to rapidity. The general idea (event selection, unreal tracks in roman pots, etc) is the same. Results are shown in figure 4b. This figure does not take into account the uncertainty in the CMS measurements. This is because the same procedure as in figure 2c has been done, has not been done yet for the rapidity. This is still open to do in the future.
What could be done in the future is to look at data points that agree with CMS both in mass and in rapidity and only take these data points into consideration for experimental purposes.

7 Conclusion and Outlook

It has become clear that TOTEM is not perfectly able to reconstruct the same as what CMS does. This might be explained from tracks in roman pots that do not come from protons or from protons that do not originate from the same interaction point. Although it is possible to remove some tracks that did not come from protons from our data, it is not possible to do it for all of them. Up till now it is not possible to make a statement whether measured protons in different arms came from the same interaction point. However, by the introduction of the timing detector in TOTEM improvements can be made upon this. What could be done as a follow up is investigate the antisymmetric error bars in the CMS measurements and look more closely into the rapidity as has been done for the invariant mass.
Next to this improvements need to be made on how well CMS reconstructs the invariant mass. Hereby one could for example think of event selection, jet selection and P_t and η cuts applied to the final state products. From the current measurements, CMS shows an obvious overshoot to the invariant mass as inserted by the Monte Carlo methods.
If in the end TOTEM seems very promising in measuring the invariant mass of the $t\bar{t}$ system, it prosperous to perform verifications of the SM or anomalous models in particle physics.

8 XiMassAndRap_ErrCorrection.C

In this section the "XiMassAndRap_ErrCorrection.C" script will be elucidated. This script is included after this section. The general structure of the script is given in Figure 5, which shows that the script is devided up in two parts. One is the "main" function, the other is the assistant functions. The main function contains the general outline and procedures the script does. We can basically define 4 main tasks in the "main" function. For some of these tasks long parts of code are required. To keep the script clear, these parts are casted into an external function defined outside main: the assistant functions. In this case there are three assistant functions, which will be treated later.

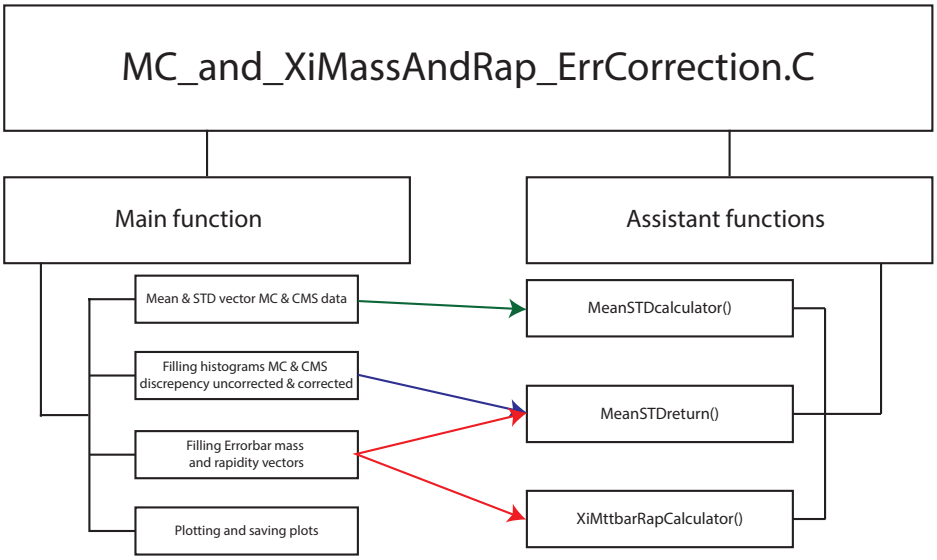


Figure 5: Structure of script

All the assistant functions are already called at the top of the script. (At this point I am not sure whether this is highly necessary, but one can try to remove them from the top part and see whether it still works).

8.1 Main

Underneath the 4 main tasks and the assumptions/errors made in them will be explained. Note that in this function we only take events with in total 4 jets of which 2 bjets into account.

1) Defining the vector containing the mean and STD:
In this part a simulation file is taken as an input file. In this MC simulations have been done whereby tt_{bar} systems are created and it has been simulated how CMS detects them. Part 1 will calculated what the mean relative deviation of CMS is compared to MC $\left(\frac{\text{Mass}_{\text{CMS}} - \text{Mass}_{\text{CM}}}{Mass_{\text{CMS}}}\right)$. This will be done for discrete CMS energy regimes, since the mean relative deviation depend on the energy of the detected tt_{bar} system. The size of the regimes can be altered in the script. Besides the mean, also the standard deviation of the mean is calculated. Both of them are done by MeanSTDcalculator(). Then for every regime, the lower mass value, the higher mass value, the mean relative deviation and the relative STD of this mean are stored

into a dynamic matrix (called MErrInfo in the script). The number of matrix elements could be calculated on beforehand ($\frac{M_{high}-M_{low}}{spacing}$)), which allows for static matrix usage. But that has not been done here. ROOT/C++ requires the user to enlarge the matrix every time the user adds something to it. This is done by the "MErrInfo[i].resize(...)" command in the for loop.

2) Filling the histograms showing the MC mass vs. the CMS mass.
In this part the "not bias corrected" values of the MC and the CMS mass are plotted against each other in a histogram. If CMS would perfectly reconstruct MC, this would obtain a straight line. It seems that it is not a straight line. Therefore we need to compensate for the bias CMS applies. The bias depends on the energy that CMS measures as we saw in part 1. To cope with this bias, the mean relative deviation (calculated in part 1) per energy regime is subtracted from all energies in that regime. The function MeanSTDreturn() takes care of finding the correct Mean and the STD in each regime

3) Preparing to plot the mass measured in the Roman Pots against the mass measured in CMS:
Here a file containing real data is used as an input. The data entails the mass measured in CMS and the mass measured in the Roman Pots. To store the values of CMS mass and rapidity and the Roman Pot mass and rapidity 8 dynamic floating point vectors (TVectorF) are defined. They are dynamic since it is unknown on beforehand how many events will be measured. By every measurement the size of these vectors is increased by 1 using the ".ResizeTo" command (see the for loop). For each event that fullfills the selection criterion (njets == 4 and nbjets == 2) the mass, rapidity and corresponding errors are calculated by the external function XiCalculator(). Next to the erros in the mass and rapidity from the Roman Pot measurements, the errors of the CMS measurement is needed. This is again calculated by MeanSTDreturn(). So note that the errorcorrection that is applied to the CMS measurements in the datafile is based on the comparison of simulated CMS measurements with Monte Carlo simulations of the tt_{bar} system.
Important notes:
-) In the application of the STD to the CMS measurements it has been assumed that the shape of the deviation of CMS to MC is symmetric around its mean value. However, in reality this is NOT true. It has has a long tail to larger deviations. For future improvements, this could be taken into account.
-) The calculation of the errors on the mass and rapidity from the Roman Pots is not yet completely correct. If two tracks are measured, the final error in the mass is not calculated as a weighted error. Both error values count for 50% here. See the function XiCalculator(). *Besides it could be there is an error in the function that calculates these Xi-error values ("proton_reco.computeXiSpline", see "ExclusiveTop.cc" script). This since they should not exceed 5.5%, while they do in the current situation.*

4) Plotting and saving plots:
At this point all the calculations have been performed. The last actions to do are drawing the histograms, plotting the TGraphErrors plot that include the errors on measurements and save both of them. Note that the plots of the TGraphErrors, based on calculations involving ξ measured by the Roman Pots are saved in a different directory than the two histograms originating from simulation data.

8.2 Assistant functions

In this section the assistant functions will be discussed.
For all the functions the library "boost.tuple.tuple.hpp" is used, which allows for making functions with more than one output. To define a function that returns multiple outputs, use the following structure:
function definition: "boost::tuple<$\text{Type}_{\text{OutVar1}}$, $\text{Type}_{\text{OutVar2}}$, ... > *functionname*($\text{Type}_{\text{InVar1}}$ $\text{Name}_{\text{InVar1}}$, $\text{Type}_{\text{InVar2}}$ $\text{Name}_{\text{InVar2}}$,...)"
Declare which variables the function needs to output:
"return boost::make_tuple($\text{Name}_{\text{output variable 1}}$, $\text{Name}_{\text{output variable 2}}$)".
Note that after this statement the function stops excecuting, so make sure everything is coded above this statement. Also, in case of using "if" statements, make sure this return statement appears in every possible way of going into the if statement. Otherwise the script will not work.
Setting variables of Main script using the boost function:
boost::tie($\text{Name}_{\text{MainVar1}}$, ...) = BoostFunctionName($\text{Name}_{\text{InVar1}}$, ...)

MeanSTDcalculator():
This function has a lower and a higher boundary on the mass as an input as well as a TTree. From this TTree it takes all the events that have been measured, then it discards all the events that are either below or above or do not have a total amount of 4 jets whereof two are b-jets. From the remaining events it constructs a vector of the relative mismatch between the CMS measured mass and the MC input mass. Thereafter it calculates the mean and the STD of this vector and returns it. By doing this one can calculate the bias and the STD in the bias that CMS has compared to MC.

MeanSTDreturn():
This function takes the mass measured by CMS and a matrix as an input. This matrix is constructed by having a row vector and in each row there is a column vector. The point where this matrix is constructed is in part 1) of the main script. Each row contains the lower mass boundary, the higher mass boundary, the relative mean deviation of CMS compared to MC in this mass region and the STD of this mean. What this function does is that it accepts a measured mass, seeks which relative mean and STD belong to it, and give these values back as a return output. Then in part 2) of the script these outputs are used to compensate for the bias of CMS.

XiCalculator():
This is the function that calculates the mass and rapidity of the ttbar system as measured by the forward protons measured in the Roman Pots. Therefore it needs the number of forward track entries per event and the potnumber, armnumber, ξ-value and the error in the ξ-value corresponding to the forward tracks now it still contains Mttbar_Meas as an input variable, but this one is not used and can be removed. Besides fwrdtrkEntries all the input variables are vectors. Initially the forward tracks are not organized by arm and pot number, all of them are said to be in the same "bucket". So the first thing the function does is organizing all the events over 4 buckets (defined by the arm and the potnumbers). Note that the arm numbers used are 0 and 1 and the pot numbers are 2 and 3, which is due to how the input file is organized. Now that every track is stored in the right bucket, the mass and rapidity could be calculated.
In this script, for "simplicity", we only look at one possible tt_{bar} event at the time. This is not a severe simplification since the probability of having more than one event is rare. The filter applied to just select events that have one (possible) tt_{bar} system is the following. The number of events in all the pots may at maximum be one. This is to not compare a track of proton from collision A

with a proton from collision B. Besides the number of tracks in each arm should be at least 1. This basically requires at least one track in one of the pots, so that we can compare the ξ-value of one arm with the other arm. It is allowed to have in total two tracks in one arm. This means that the proton went through both Roman Pots in the arm. If the event complies to all these requirements, then the ξ-value is calculated. In case there are two tracks in one arm, the average ξ-value is calculated as well as the error in the ξ-value. *not that up till now, this is not yet the weighted error.* Thereafter the mass and rapidity from the Roman Pot measurement are calculated with their errors.
The "ControlVariable" used in script *MeanSTDreturn()* and *XiCalculator()* check whether the event fulfils the set requirements. If yes ControlVariable equals 1 and the event will be taken into account in the main script, if not, then ControlVariable equals 0 and the event is disregarded in the main script.

```
1  #include <iostream>
2  #include <string>
3  #include <math.h>
4  #include <boost/tuple/tuple.hpp>
5
6  using namespace std;
7
8  boost::tuple<Int_t, Float_t, Float_t, Float_t, Float_t> ...
       XiCalculator(Int_t fwdtrkEntries, Int_t fwdtrk_pot[], Int_t ...
       fwdtrk_arm[], Float_t xi[], Float_t xi_error[], Float_t Mttbar_Meas);
9  boost::tuple<Float_t, Float_t> MeanSTDCalculator(Float_t Mlow, Float_t ...
       Mhigh, TTree *Boom);
10 boost::tuple<Int_t, Float_t, Float_t> MeanSTDreturn(Float_t Mttmeas, ...
       vector<vector<float> > &MErr);
11
12
13
14 // ------------------------ BEGIN OF MAIN FUNCTION ...
       ------------------------ //
15 void MC_and_XiMassAndRap_ErrCorrection(){
16 cout << "Hello user, I have started excecuting your program" << endl;
17 /*
18 ////////////////////////////////////////////////////////////////////////////////
19 Making vector containing the means and std from CMS compared to MC
20 ////////////////////////////////////////////////////////////////////////////////
21 */
22
23 TFile *f0 = new TFile("MC13TeV_TTJets.root"); //!!!!!!!!!!!!!!!! NOTE ...
       THAT YOU USE THE RIGHT (simulation) DATAFILE FOR THE CALCULATIONS ...
       !!!!!!!!!!!!!!!!
24 TTree *data0 = (TTree*)f0->Get("data");
25
26 vector<vector<float> > MErrInfo(4);
27 int cSize = 0;
28 float MhighInit = 250; // Fill in the appropriate value (or ask user to ...
       fill this in)
29 float MhighMax = 2500; // Fill in the appropriate value (or ask user to ...
       fill this in)
30 float Mlow, Spacing;
31 Spacing  = 50;
32 Float_t Mean1, STD1;
33 for (float Mhigh=MhighInit; Mhigh<MhighMax; Mhigh=Mhigh+Spacing){
34 Mlow = MhighInit-Spacing;
35 boost::tie(Mean1, STD1) = MeanSTDCalculator(Mlow, Mhigh, data0);
36 for (Int_t i = 0; i < 4; i++){
37 MErrInfo[i].resize(cSize+1);
38 }
```

```
39 MErrInfo[0][cSize] = Mlow;
40 MErrInfo[1][cSize] = Mhigh;
41 MErrInfo[2][cSize] = Mean1;
42 MErrInfo[3][cSize] = STD1;
43 cSize++;
44 }
45
46 /*
47 ////////////////////////////////////////////////////////////////////////////////////////////////
48 FILLING THE HISTOGRAM SHOWING THE MONTE CARLO MASS VS THE CMS MASS ... WITHOUT AND WITH ERROR CORRECTION
49 ////////////////////////////////////////////////////////////////////////////////////////////////
50 */
51 Int_t nentries00 = (Int_t)data0->GetEntries();
52 Float_t Mttbar_gen00, Mttbar_rec00;
53 Int_t njets00, nbjets00;
54 data0->SetBranchAddress("mttbarGen",&Mttbar_gen00);
55 data0->SetBranchAddress("mttbarMeas",&Mttbar_rec00);
56 data0->SetBranchAddress("nj",&njets00);
57 data0->SetBranchAddress("nb",&nbjets00);
58
59
60 TH2F *MC_vs_CMS = new TH2F("mGenRec","mtt_{gen} vs mtt_{rec}: No correction;mtt_{gen};mtt_{rec}",300,0,3000,300,0,3000);
61 for(Int_t ii = 0; ii<nentries00; ii++){
62 data0->GetEntry(ii);
63 if(njets00 == 4 && nbjets00 == 2){
64 MC_vs_CMS->Fill(Mttbar_gen00,Mttbar_rec00);
65 }
66 }
67
68 TH2F *MC_vs_CMS_ErrCor = new TH2F("mGenRec","mtt_{gen} vs mtt_{rec}: Error correction;mtt_{gen};mtt_{rec}",300,0,3000,300,0,3000);
69 Int_t ControlVariable00;
70 Float_t Mean00, STD00;
71 Float_t Mttbar_rec_cor00;
72
73 for(Int_t ii = 0; ii<nentries00; ii++){
74 data0->GetEntry(ii);
75 if(njets00 == 4 && nbjets00 == 2){
76 boost::tie(ControlVariable00, Mean00, STD00) = MeanSTDreturn(Mttbar_rec00, MErrInfo);
77 if(ControlVariable00 == 1){
78 Mttbar_rec_cor00 = Mttbar_rec00*(1-0.01*Mean00);
79 MC_vs_CMS_ErrCor->Fill(Mttbar_gen00,Mttbar_rec_cor00);
80 }
81 }
82 }
83
84 cout << "The program just finished making the array containing the Mean and STD values" << endl;
85
86
87
88 /*
89 ////////////////////////////////////////////////////////////////////////////////////////////////////////////
90 FILLING THE VECTORS NEEDED TO PLOT THE ERROR CORRECTED MASS AND RAPIDITY OF CMS COMPARED TO ROMAN POTS
91 ////////////////////////////////////////////////////////////////////////////////////////////////
```

```
92 */
93
94 TFile *f1 = new ...
       TFile("Data13TeV_SingleMuonSingleElectron_2016CG.root"); ...
       //!!!!!!!!!!!!!!!! NOTE THAT YOU USE THE RIGHT DATAFILE FOR THE ...
       CALCULATIONS !!!!!!!!!!!!!!!!
95 TTree *data = (TTree*)f1->Get("data");
96 Int_t nentries = (Int_t)data->GetEntries();
97
98 Int_t njets, nbjets;
99 // Float_t Mttbar_gen, Mttbar_rec;
100 Float_t Mttbar_Meas, Rapidity, Eta;
101 Int_t fwdtrkEntries,fwdtrk_pot[50], fwdtrk_arm[50];
102 Float_t fwdtrk_x[50], fwdtrk_y[50], xi[50], xi_error[50];
103
104 data->SetBranchAddress("mttbar",&Mttbar_Meas);
105 data->SetBranchAddress("Rapidity",&Rapidity);
106 data->SetBranchAddress("eta",&Eta);
107 data->SetBranchAddress("nj",&njets);
108 data->SetBranchAddress("nb",&nbjets);
109 data->SetBranchAddress("fwdtrkEntries",&fwdtrkEntries);
110 data->SetBranchAddress("fwdtrk_pot",fwdtrk_pot);
111 data->SetBranchAddress("fwdtrk_arm",fwdtrk_arm);
112 data->SetBranchAddress("fwdtrk_x",fwdtrk_x);
113 data->SetBranchAddress("fwdtrk_y",fwdtrk_y);
114 data->SetBranchAddress("xi",xi);
115 data->SetBranchAddress("xi_error",xi_error);
116
117 Int_t ConVar, ControlVariable, Counter=0;
118 Float_t Mttbar_XiFill, Mttbar_Xi_Err, RapFill, RapErrFill;
119 TVectorF Xm, Ym, eXm, eYm;
120 TVectorF Xr, Yr, eXr, eYr;
121 Float_t Mean2, STD2;
122
123 // If chosen by user, only use events with in total 4 jets.
124 for (Int_t i=0; i<nentries; i++){
125 data->GetEntry(i);
126 if (njets == 4 && nbjets == 2)
127 {
128 boost::tie(ConVar, Mttbar_XiFill, Mttbar_Xi_Err, RapFill, RapErrFill) = ...
       XiCalculator(fwdtrkEntries,fwdtrk_pot,fwdtrk_arm,xi,xi_error,Mttbar_Meas);
129 boost::tie(ControlVariable, Mean2, STD2) = MeanSTDreturn(Mttbar_Meas, ...
       MErrInfo);
130
131 if(ConVar == 1 && ControlVariable == 1){  // check whether there are at ...
       least tracks in both pots and check whether this mass has a defined ...
       Mean and STD.
132 Counter++;
133 Xm.ResizeTo(Counter); eXm.ResizeTo(Counter);
134 Ym.ResizeTo(Counter); eYm.ResizeTo(Counter);
135 Xm[Counter-1] = Mttbar_Meas*(1-0.01*Mean2); eXm[Counter-1] = ...
       0.01*STD2*Mttbar_Meas;  //Please control whether you did this ...
       correctly Sjoerd.
136 Ym[Counter-1] = Mttbar_XiFill; eYm[Counter-1] = Mttbar_Xi_Err;
137
138 Xr.ResizeTo(Counter); eXr.ResizeTo(Counter);
139 Yr.ResizeTo(Counter); eYr.ResizeTo(Counter);
140 Xr[Counter-1] = Rapidity; eXr[Counter-1] = 0;
141 Yr[Counter-1] = RapFill; eYr[Counter-1] = RapErrFill;
142 }
```

```
143 }
144 }
145
146 /*
147 ////////////////////////////////////////////////////////////////////////////////////////////
148 PLOTTING THE MC vs CMS MASS HISTOGRAM AND THE ERROR CORRECTED MASS AND ...
        RAPIDITY PLOT OF CMS AND ROMAN POTS
149 ////////////////////////////////////////////////////////////////////////////////////////////
150 */
151 TCanvas *c000 = new TCanvas;
152 MC_vs_CMS->Draw("colz");
153 c000->Update();
154 TImage *MC_vs_CMS_Plot = TImage::Create();
155 MC_vs_CMS_Plot->FromPad(c000);
156 MC_vs_CMS_Plot->WriteImage("/afs/cern.ch/user/s/sloenen/workspace/CMSSW_8_0_28/src/TopLJet
157
158 TCanvas *c00 = new TCanvas;
159 MC_vs_CMS_ErrCor->Draw("colz");
160 c00->Update();
161 TImage *MC_vs_CMS_ErrCor_Plot = TImage::Create();
162 MC_vs_CMS_ErrCor_Plot->FromPad(c00);
163 MC_vs_CMS_ErrCor_Plot->WriteImage("/afs/cern.ch/user/s/sloenen/workspace/CMSSW_8_0_28/src/
164
165 TCanvas *c0 = new TCanvas;
166 TGraphErrors *CTPPS_Mass = new TGraphErrors(Xm,Ym,eXm,eYm);
167 CTPPS_Mass->SetTitle("Mass CMS vs. TOTEM; Mtt_{CMS}; Mtt_{RMP}");
168 CTPPS_Mass->SetMarkerStyle(kOpenCircle);
169 CTPPS_Mass->SetMarkerColor(kBlue);
170 CTPPS_Mass->SetLineColor(kBlue);
171 CTPPS_Mass->Draw("APE");
172 c0->Update();
173 TImage *CTPPS_Mass_figure = TImage::Create();
174 CTPPS_Mass_figure->FromPad(c0);
175 CTPPS_Mass_figure->WriteImage("/afs/cern.ch/user/s/sloenen/workspace/CMSSW_8_0_28/src/TopL
176
177 TCanvas *c = new TCanvas;
178 TGraphErrors *CTPPS_Rap = new TGraphErrors(Xr,Yr,eXr,eYr);
179 CTPPS_Rap->SetTitle("Rapidity CMS vs. TOTEM; Ytt_{CMS}; Ytt_{RMP}");
180 CTPPS_Rap->SetMarkerStyle(kOpenCircle);
181 CTPPS_Rap->SetMarkerColor(kBlue);
182 CTPPS_Rap->SetLineColor(kBlue);
183 CTPPS_Rap->Draw("APE");
184 c->Update();
185 TImage *CTPPS_Rap_figure = TImage::Create();
186 CTPPS_Rap_figure->FromPad(c);
187 CTPPS_Rap_figure->WriteImage("/afs/cern.ch/user/s/sloenen/workspace/CMSSW_8_0_28/src/TopLJ
188
189 }
190
191
192 ////////////////////////////////////////////// UNDERNEATH ARE THE ...
        FUNCTIONS THAT ARE USED IN THE MAIN FUNCTION ...
        ////////////////////////////////////////////////////////
193
194
195
196
197 /*
198 ////////////////////////////////////////////////////////////////////////////////////////////
199 Making function to calculate the mean and the STD of the overshoot in a ...
```

```
      range of CMS measured energy values of ttbar system
200 ////////////////////////////////////////////////////////////////////////////////////////////////////
201 */
202 boost::tuple<Float_t, Float_t> MeanSTDCalculator(Float_t Mlow, Float_t ...
      Mhigh, TTree *Boom )
203 {
204 cout << "Just entered the MeanSTDCalculator function, which contains ...
      the TTree Boom pointer" << endl;
205 Float_t Mttbar_err, Mttbar_gen, Mttbar_meas;
206 Int_t njets, nbjets;
207 Boom->SetBranchAddress("mttbarErr",&Mttbar_err);
208 Boom->SetBranchAddress("mttbarGen",&Mttbar_gen);
209 Boom->SetBranchAddress("mttbarMeas",&Mttbar_meas);
210 Boom->SetBranchAddress("nj",&njets);
211 Boom->SetBranchAddress("nb",&nbjets);
212
213 std::vector<float> Evec;
214 Int_t nentries0 = (Int_t)Boom->GetEntries();
215 for(Int_t k=0; k<nentries0; k++){
216 Boom->GetEntry(k);
217 if(Mttbar_meas > Mlow && Mttbar_meas < Mhigh && njets == 4 && nbjets == 2){
218 Evec.push_back(100*Mttbar_err/Mttbar_meas);
219 }
220 }
221
222 Float_t Mean, STD;
223 Mean = TMath::Mean(Evec.begin(), Evec.end());
224 STD = TMath::RMS(Evec.begin(), Evec.end());
225
226 return boost::make_tuple(Mean,STD);
227 }
228
229
230 /*
231 ////////////////////////////////////////////////////////////////////////////////////////////////////
232 Making function to extract the right mean and STD value for the ...
      measured CMS ttbar system, so we can include errorbars on mttbar_CMS
233 ////////////////////////////////////////////////////////////////////////////////////////////////////
234 */
235 boost::tuple<Int_t, Float_t, Float_t> MeanSTDreturn(Float_t Mttmeas, ...
      vector<vector<float> > &MErr)
236 {
237 Float_t Mean, STD;
238 Int_t LoopVar;
239 Int_t ControlVariable = 0;
240 Int_t Msize = MErr[1].size();
241
242 if(Mttmeas ≤ MErr[0][0] || Mttmeas > MErr[1][Msize-1]){
243 ControlVariable=0;
244 }
245 for(Int_t l = 0; l<Msize;l++){
246 if(Mttmeas > MErr[0][l] && Mttmeas ≤ MErr[1][l]){
247 ControlVariable = 1;
248 Mean = MErr[2][l];
249 STD = MErr[3][l];
250 }
251 }
252
253 return boost::make_tuple(ControlVariable, Mean, STD);
254 }
```

```
255
256
257
258
259 /*
260 ////////////////////////////////////////////////////////////////////////////////////////////////
261 Making function to calculate Mttbar and Rapidity from Roman Pots and ...
       its error
262 ////////////////////////////////////////////////////////////////////////////////////////////////
263 */
264 boost::tuple<Int_t, Float_t, Float_t, Float_t, Float_t> ...
       XiCalculator(Int_t fwdtrkEntries, Int_t fwdtrk_pot[], Int_t ...
       fwdtrk_arm[], Float_t xi[], Float_t xi_error[], Float_t Mttbar_Meas)
265 {
266 Int_t ControlVariable;
267 Float_t Xi_0_Avg, Xi_1_Avg, Xi_0_Avg_Error, Xi_1_Avg_Error, Mttbar_Xi, ...
       Mttbar_Xi_Error;
268 Float_t Rapidity, Rapidity_Error;
269
270 if(fwdtrkEntries ≥ 2 && fwdtrkEntries !=0){
271 Int_t NPosttSys = 1;
272 Int_t NArm0=0, NArm1=0, NPot02=0, NPot03=0, NPot12=0, NPot13=0;
273 Float_t Xi_02[50], Xi_03[50], Xi_12[50], Xi_13[50], Xi_02_error[50], ...
       Xi_03_error[50], Xi_12_error[50], Xi_13_error[50];
274
275 // See how many tracks there are in each arm
276 Int_t Counter02 = 0, Counter03 = 0, Counter12 = 0, Counter13 = 0;
277 for (Int_t iCount = 0; iCount<fwdtrkEntries; iCount++){
278 if(fwdtrk_arm[iCount] == 0){
279 NArm0++;
280 if(fwdtrk_pot[iCount] == 2){
281 NPot02++;
282 Xi_02[Counter02] = xi[iCount];
283 Xi_02_error[Counter02] = xi_error[iCount];
284 Counter02++;
285 }
286 if(fwdtrk_pot[iCount] == 3){
287 NPot03++;
288 Xi_03[Counter03] = xi[iCount];
289 Xi_03_error[Counter03] = xi_error[iCount];
290 Counter03++;
291 }
292 }
293 if(fwdtrk_arm[iCount] == 1){
294 NArm1++;
295 if(fwdtrk_pot[iCount] == 2){
296 NPot12++;
297 Xi_12[Counter12] = xi[iCount];
298 Xi_12_error[Counter12] = xi_error[iCount];
299 Counter12++;
300 }
301 if(fwdtrk_pot[iCount] == 3){
302 NPot13++;
303 Xi_13[Counter13] = xi[iCount];
304 Xi_13_error[Counter13] = xi_error[iCount];
305 Counter13++;
306 }
307 }
308 }
309
```

```
310 /*
311 We only select events in which ONE ttbar system is created. Not less, ...
        not more.
312 This is done by requiring at maximum 1 track in each pot (maximum one ...
        proton on each side). Besides, it also requires at least one track ...
        in each arm.
313 */
314
315 /*
316 In the following every Xi value has an equal weight in determining the ...
        final Xi value.
317 The correct way to do it, is giving each Xi value a weight, which is ...
        determined by the size of its uncertainty. With this weight one ...
        calculates the final Xi-value.
318 See:
319 Weight0_2 = 1/pow(Xi_02_error[jCount],2);   --> What if Xi_02_error is ...
        not measured, then you devide by 0. So seek a solution for that.
320 Weight0_3 = 1/pow(Xi_03_error[jCount],2);   --> What if Xi_03_error is ...
        not measured, then you devide by 0. So seek a solution for that.
321 TotWeight0 = Weight0_2 + Weight0_3;
322 Xi_0_Avg = 1/TotWeight0*(Weight0_2 * Xi_02[jCount] + Wieght0_3 * ...
        Xi_03[jCount]);
323 Xi_0_Avg_Error = sqrt( pow(1/TotWeight0,2) * ( pow(Weight0_2 * ...
        Xi_02_error[jCount],2) + pow(Weigth0_3 * Xi_03_error[jCount],2) )  );
324 */
325 Float_t Weight0_2, Weight0_3, TotWeight0, Weight1_2, Weight1_3, TotWeight1;
326 if(NPot02 ≤ 1 && NPot03 ≤ 1 && NPot12 ≤ 1 && NPot13 ≤ 1 && NArm0 ≥ 1 && ...
        NArm1 ≥ 1){
327 for(Int_t jCount = 0; jCount<NPosttSys; jCount++){
328
329 Xi_0_Avg = ( NPot02*Xi_02[jCount] + NPot03*Xi_03[jCount] )/(NPot02+NPot03);
330 Xi_0_Avg_Error = 0.055*Xi_0_Avg;  // ( NPot02*Xi_02_error[jCount] + ...
        NPot03*Xi_03_error[jCount] )/(NPot02+NPot03);
331
332 Xi_1_Avg = ( NPot12*Xi_12[jCount] + NPot13*Xi_13[jCount] )/(NPot12+NPot13);
333 Xi_1_Avg_Error = 0.055*Xi_1_Avg;  // ( NPot12*Xi_12_error[jCount] + ...
        NPot13*Xi_13_error[jCount] )/(NPot12+NPot13);
334
335 Mttbar_Xi = sqrt( pow(13000.0,2) * Xi_0_Avg * Xi_1_Avg); // the 13000 ...
        stands for 13TeV. This since the Mttbar_Meas is measured in units of ...
        GeV.
336 Mttbar_Xi_Error = sqrt(pow(sqrt( pow(13000.0,2) / Xi_0_Avg * ...
        Xi_1_Avg)*Xi_0_Avg_Error,2) + pow(sqrt( pow(13000.0,2) * Xi_0_Avg / ...
        Xi_1_Avg)*Xi_1_Avg_Error,2));
337
338 Rapidity = 0.5*log(Xi_0_Avg/Xi_1_Avg);
339 Rapidity_Error = sqrt( pow( 1/(2*Xi_0_Avg)*Xi_0_Avg_Error, 2) + pow( ...
        -1/(2*Xi_1_Avg)*Xi_1_Avg_Error, 2) );
340 }
341
342 ControlVariable = 1;
343
344 return ...
        boost::make_tuple(ControlVariable,Mttbar_Xi,Mttbar_Xi_Error,Rapidity,Rapidity_Error);
345 }
346
347 else {
348 ControlVariable = 0;
349 return ...
        boost::make_tuple(ControlVariable,Mttbar_Xi,Mttbar_Xi_Error,Rapidity,Rapidity_Error);
```

```
350 }
351 }
352
353 else {
354 ControlVariable = 0;
355 return ...
        boost::make_tuple(ControlVariable,Mttbar_Xi,Mttbar_Xi_Error,Rapidity,Rapidity_Error);
356 }
357
358 }
```