JOINT INSTITUTE FOR NUCLEAR RESEARCH
Veksler and Baldin laboratory of High Energy Physics

FINAL REPORT ON THE START PROGRAMME

BM@N Central Tracker Efficiency for Negative Pion Tracks

Supervisor:
Mr Vasilii Plotnikov

Student:
Reem Mohamed, Egypt
Cairo University

Participation period:
July 17 – September 3, 2022

Dubna, 2022

Abstract

BM@N experiment in the NICA complex at JINR is a fixed target experiment aimed to study the production of strange particles in relativistic nucleus-nucleus collisions. In the experimental run of March 2018, argon and krypton ion beams were used with various fixed targets. This report focuses on obtaining the efficiency of both the three planes of the two-coordinate forward silicon detectors (FwdSi) and the six planes of the two-coordinate Gaseous Electron Multiplier (GEM) detectors in the central tracker system for the reconstructed negative tracks in the Ar/Kr run. These results are also compared with the results obtained from the simulated Monte Carlo (MC) tracks.

1. Introduction

BM@N, which stands for Baryonic Matter at Nuclotron, is an experiment conducted in the NICA complex at JINR to study the properties of relativistic heavy-ion collisions[1]. This experiment aims to provide a better undertanding of Quantum Chromodynamics (QCD) matter at densities similar to those predicted to exist in compact stellar objects[1]. Moreover, the studies of this experiment may shed light on the role of hyperons in neutron stars[1]. Since the first experimental run of the BM@N experiment, multiple ion beams were used such as deuteron, carbon, argon and krypton beams[2]. As for the experimental run of March 2018, argon and krypton beams were used with five fixed targets[2]. To identify the charged particles and nucleus fragments in the inelastic reactions of the argon and krypton beams with the fixed targets, a variety of detector systems were used.

In this report, the main goal is to calculate the efficiency of the central tracker system for negative pion tracks. The average efficiencies of the Si and GEM stations for negative tracks range from 80 to 95%. These results showed a good agreement with Monte Carlo simulated tracks.

2. Experimental Set-Up

The scheme of the BM@N set-up that was used in the experimental run of March 2018 is shown in **Figure 1**. In this experimental run, Ar/Kr beam was used, and the set-up consisted of a central tracker mounted inside the SP41 analyzing magnet, an outer tracking system, a time-of-flight (ToF) system, calorimeters, a trigger system, and read-out electronics and data acquisition (DAQ) system[3,4]. The advantage of such a set-up is that it can provide track measurements with high precision, and it can be used for particle identification with low and high momenta and can be used for the collision centrality analysis[1,2]. Moreover, it is feasible to obtain the BM@N optimal geometrical acceptance and momentum resolution for various beam energies and different processes, and this is due to the 1 m vertical gap between the poles of the SP41 analyzing magnet which in turn allows the magnetic field to reach a maximum value of 1.2 T [4].

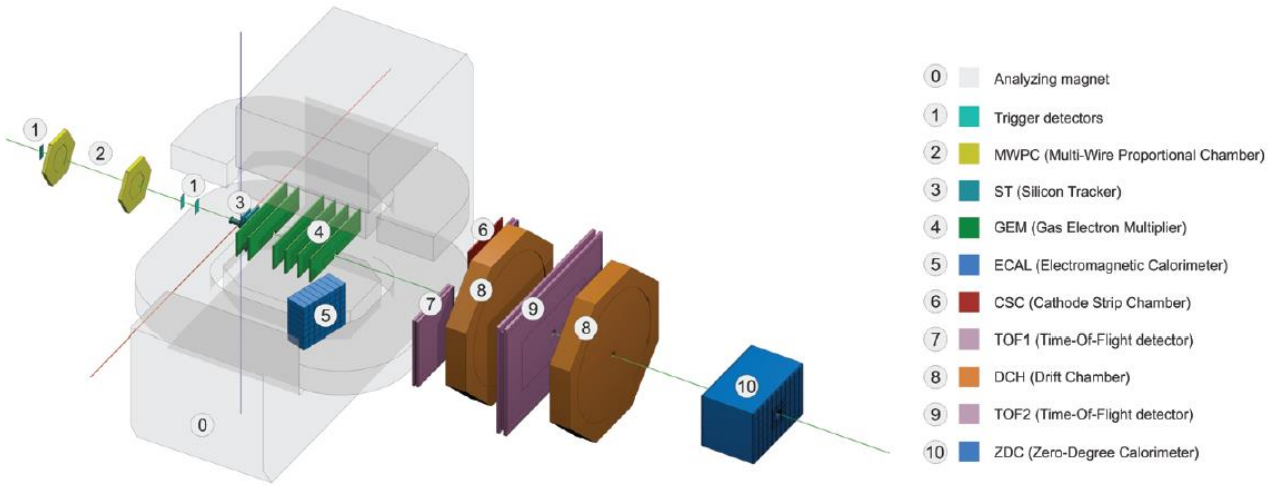


Figure 1: BM@N experimental set-up.[5]

2.1. Central Tracker

The central tracker is responsible for measuring the momenta and multiplicities of the resulting charged particles tracks. According to the experimental run of March 2018, the central tracking system shown in **Figure 2** consisted of 6 planes of two-coordinate Gaseous Electron Multiplier (GEM) detectors and 3 planes of two-coordinate forward silicon detectors (FwdSi)[1,2]. The GEM detectors are an excellent choice for the tracking system of the BM@N experiment as they are capable of stable operation in high radiation loadings up to 10^5 Hz/cm^2 and can operate in a strong magnetic field reaching to 1.5 T. Also, the GEM detectors possess the basic requirements for a tracking system such as having high momentum and spatial resolution and having efficiencies better than 95% and possessing the maximum possible geometrical acceptance in terms of the BM@N experiment dimension[1]. As for the FwdSi detectors, they were added to the central tracker to enhance the primary vertex reconstruction and to improve the tracking efficiency[4]. The central tracking system was tuned to measure the strange V0 particles soft decaying products and Monte-Carlo simulation was used to optimize the placement of the GEM and FwdSi detectors[2].

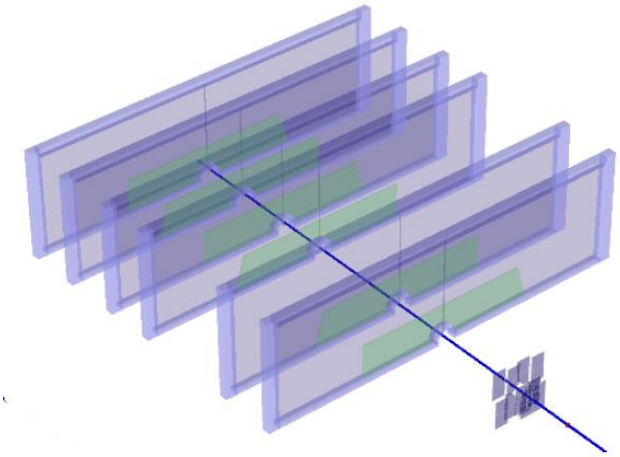


Figure 2: Central tracker configuration. It consists of 6 planes of GEM detector and 3 planes of FwdSi detector.

2.2. ToF System

The time-of-flight system is used for the identification of the particles, and it consists of a start detector T0 situated near to the target, ToF-400 detector, and ToF-700 detector. The latter two detectors are based on the technologies of multi-gap Resistive Plate Chamber (mRPC) and are installed at distances of about 4 m and 7 m from the target, respectively[3]. The ToF system has a time resolution of 80-100 ps which allows the discrimination between hadrons (π, K, p) and is also sufficient to separate between light nuclei with momenta reaching to few GeV/c[1].

2.3. Outer Tracker

The outer tracker serves as a link between the hits detected by the ToF-400 and ToF-700 detectors and the measured tracks in the central tracking system, and it consists of two drift chambers (DCH) and a cathode strip chamber (CSC). The DCH and the CSC are installed outside the magnetic field, and both filters the bad tracks. The DCH has another function that is to measure the momentum and the angular distribution of the beam. The CSC was used for the first time in the run of March 2018, and this is to compensate the low efficiency of the DCH. The tracks reconstructed from the GEM and CSC hits and extrapolated to the ToF-400 have an improved momentum resolution and can be used in the separation of the secondary particles (π, p, K, light nuclei) in the 0.5-3.5GeV/c momentum range. In the upcoming runs, two CSCs will be installed to cover the ToF-700 system and replace the DCH.[1,3]

2.4. Calorimeters

There are two types of calorimeters installed in the BM@N experiment, the zero-degree calorimeter (ZDC) and the electro-magnetic calorimeter (ECAL). The ZDC measures the energy of forward

going particles to be used in the analysis of the collision centrality, while ECAL studies the processes with electro-magnetic probes (γ, $e^{\pm}$) in the final state.[4]

2.5. Trigger System

The trigger system consists of a trigger, T0, and beam detectors. The system is included in the BM@N configuration to trigger the nucleus-nucleus collisions in the target effectively, and to provide the start signal for the ToF detectors with time resolution of picoseconds. Moreover, the trigger system monitors the beam characteristics and background.[4]

2.6. Read-Out Electronics and DAQ System

The DAQ system is responsible for the realization of data transfer from the read-out electronics in the detector to the storage system. The DAQ system consists of electronic modules, network infrastructure, and a software. The experimental data stored in the DAQ storage is in a binary format and then it is digitised and converted into a ROOT format to be integrated into the BmnRoot framework.[1]

3. BmnRoot Framework

The BmnRoot was developed to support the BM@N experiment. It is based on the ROOT environment and the FairRoot framework and uses the C++ programming language for execution. The BmnRoot is used to study the detector performance and define the experimental setup. It also provides the simulation of events and is used for the reconstruction of tracks. In addition, it is used for the physics analysis of the experimental and simulated data. The comparison between the real data and the simulation results can be done easily using the BmnRoot framework.[6]

4. Fundamental Concepts

In order to use and understand the BmnRoot framework effectively, one should be familiar with ROOT and be familiar with some concepts that are discussed in this section.

4.1. ROOT Macros

A ROOT macro is C++ program which can contain ROOT classes and ROOT objects and can be compiled either by ROOT CINT command interpreter or by ACLiC compiler[6,7]. In this section, multiple examples of ROOT macros are presented.

The following macro describes an example of creating a one-dimensional histogram with 1000 random numbers having gaussian distribution with a mean of 50 and a standard deviation of 2 using the TRandom2 class. The output of this macro is shown in the following figure.

```
void t1()
{
   TRandom2 *rand = new TRandom2();
   TH1F *hist = new TH1F("hist", "Histogram",
100, 0, 100);
   for(int i=0; i<1000; i++)
   {
      double r = rand->Gaus(50,2);
      cout<<r<<endl;
      hist->Fill(r);
   }
   TCanvas *c1 = new TCanvas();
   hist->Draw();
}
```

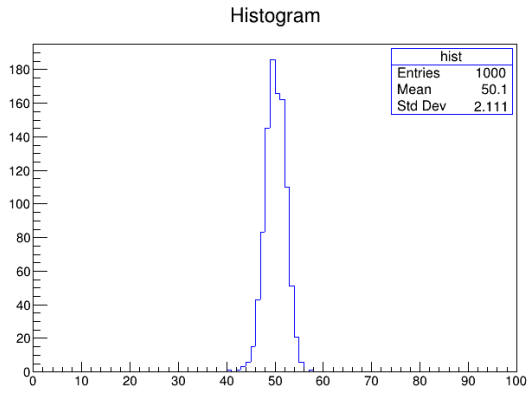


Figure 3: **Left**: A simple macro that creates a 1D gaussian histogram. **Right**: The macro's output.

The following macro is an example of storing data in a root file using TFile class. In this example, two one-dimensional histograms are created, one with a gaussian distribution and one with a uniform distribution. Both of the histograms are stored in a root file called entries.root. The output of such a macro can be viewed using TBrowser as shown in **Figure 4**.

```
void t2()
{
   TFile *f = new TFile("entries.root", "recreate");
   TRandom2 *rand = new TRandom2();
   TH1F *hist1 = new TH1F("hist1", "Histogram", 100, 0, 100);
   TH1F *hist2 = new TH1F("hist2", "Histogram", 100, 0, 100);

   for(int i=0; i<1000; i++)
   {
      double r = rand->Gaus(50,2);
      cout<<r<<endl;
      hist1->Fill(r);
      double x = rand->Uniform(100);
      cout<<x<<endl;
      hist2->Fill(x);
   }
   f->Write();
   f->Close();
}
```

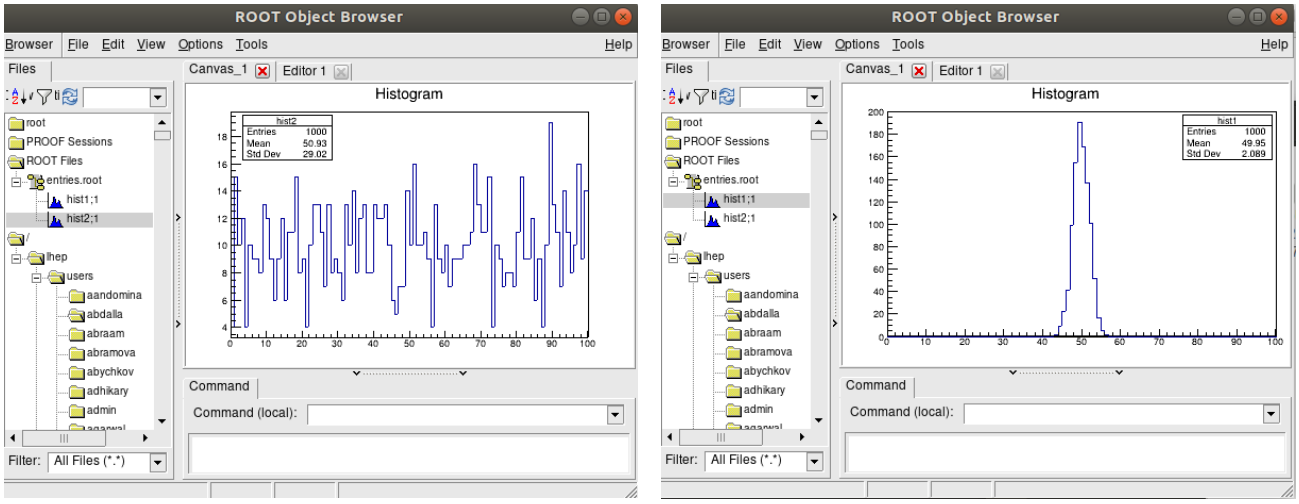


Figure 4: **Top**: A macro that stores data in a root file using TFile class. **Bottom**: An example of viewing the contents of root file in ROOT Object Browser.

Another way of storing data in ROOT is by using the TTree class. An example of such a way is shown in **Figure 5**. In this example, a new root file (tree.root) and a tree with two branches (Gauss and Uniform) are created. The two branches are filled with 1000 random numbers. The tree can be viewed using TBrowser and the visual representation of the two branches are shown in the right panel of **Figure 5**.

```
void t3()
{
   TFile *f = new TFile("tree.root",
"recreate");
   TTree *tree = new TTree("Tree",
"Distributions");
   double gaus, flat;
   tree->Branch ("Gauss", &gaus,
"Gauss/D");
   tree->Branch ("Uniform", &flat,
"Uniform/D");

   for(int i=0; i<1000; i++)
   {
      flat = gRandom->Uniform(0,100);
      gaus = gRandom->Gaus(50,2);
      tree->Fill();
   }
   f->Write();
   f->Close();
}
```

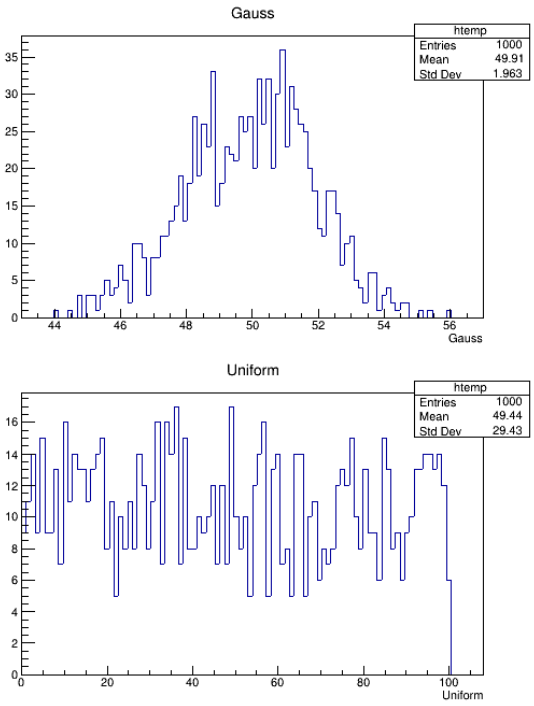


Figure 5: **Left**: A macro that stores data using TTree class. **Right**: The branches of the tree "tree" inside the tree.root file.

The following macro reads the root file that was created in the previous macro (tree.root) and get the entries of the Gauss branch and save these values inside a new tree which in turn is saved in a new root file (read.root)

```
void t4()
{
   TFile *f2 = new TFile("tree.root", "recreate");
   TTree *t1 = new TTree(("t1", "t1");

   double g;
   t1->Branch("Gauss", &g, "Gauss/D");
   TTree *Tree = (TTree*)f2->Get("Tree");
   Tree->SetBranchAddress("Gauss", &g);

   double entries = Tree->GetEntries();

   for(int i=0; i<entries; i++)
   {
      Tree->GetEntry(i);
      cout<<g<<endl;
      t1->Fill();
   }
   TFile *f1 = new TFile("read.root", "recreate");
   t1->Write();
   f1->Write();
   f1->Close();
   f2->Close();
}
```

Figure 6: An example of a macro that reads an existing root file and save one of its branches in a new tree in a new root file.

4.2. Insights into C++

To take advantage of the advanced features of ROOT, one should know some useful concepts in C++ such as classes and preprocessor directives.

4.2.1. Classes

Classes are used for defining new data types that meets the user requirements. A class consists of data members and member functions[8]. Data members must be defined inside its class while member functions can be defined inside or outside the class definition[9]. And, to use these members, objects must be created, and they can access the class members using the *dot* (.) operator. Pointers to objects can also access the class members but using the *arrow* (→) operator. Class members are by default private unless they are specified in the class definition to be public. Objects and pointers can directly access the public class members; however, they are not allowed to access private class members. Only the class members can access the private section of the class[8] .

Two of the main characteristics of Object-Oriented Programming (OOP) in C++ that are implemented in classes are inheritance and data encapsulation[10]. Inheritance allows new defined classes to use members of existing classes as if they were part of these classes[9]. The new defined classes are called derived classes or child classes while the existing classes are called base classes or parent classes[9]. The inheritance of classes can be seen in the classes of BmnRoot. As for data encapsulation, it is evident in private class members as they can only be accessed only from within the class itself and not from outside. This is important because it ensures that the data members and functions of a class are not modified or called by other classes in an unpredictable manner[10].

4.2.2. Preprocessor Directives

One of the distinctive characteristics of C++ is preprocessor directives[11]. The preprocessor directives indicate that some particular lines need to be preprocessed before the compilation of the source code by the compiler. These preprocessor directives are recognised by the preprocessor by the preceding hash sign (#) such as #define, #include, #ifdef, and #pragma etc[12]. For example, during the preprocessing, #include directive merges the header files with the source file[13]. Classes of BmnRoot are implemented in macros using preprocessor directives.

4.3. Bash Shell Script

Repeating a series of commands manually for more than 500 times as will be seen in Section 5 is not practical. Thus, to automate the process of executing a series of commands for a number of times is done by inserting these lines of commands into a bash shell script file and running it for the required number of times. The bash script file is read and executed line by line by the bash program. The bash scripts are identified by the file extension of .sh.

4.4. Sun Grid Engine

The NICA cluster uses a batch-queueing system called the Sun Grid Engine (SGE) to run multiple tasks on multiple machines. The SGE has a scheduler for assigning machines to tasks and a queueing mechanisms. Each queue has a number of slots, and each slot can only be assigned to one task. To use the SGE, a job must be defined by specifying which program to run, the input and output files, and the arguments. After that, the defined job is submitted to the SGE by *qsub* command. And to know the status of the submitted job, *qstat* command is used, while to delete a particlar job, one should use *qdel <job-ID>*.[14]

5. Si/GEM Efficiency for π^- Tracks

The main task is to obtain the efficiency of Si/GEM stations for negative pion (π^-) tracks for each target separately. The targets used in the experimental run of March 2018 were Carbon (C), Aluminium (Al), Copper (Cu), Tin (Sn), and Lead (Pb). However, the tracks that are used in calculating the efficiency must meet the quality criteria of having at least 4 GEM hits. The condition of having at least 2 Si hits was removed, as it was found that the number of Si hits are much larger in the experiment than in the Monte Carlo simulation (MC). And this is due to the large charge of the Ar ions which causes the large relaxation time of the electronics in the Si stations.

The method of obtaining the efficiency relies on the fact that each Si/GEM station is covered with a 1×1 cm grid of cells. The algorithm of calculating the efficiency uses two counters, the numerator and the denominator and loops over the nine Si/GEM stations. Then, for each cell in each station, the counters will be increased depending on whether the reconstructed track has a hit in the given station or not. So, if the track does not have a hit in that station, the algorithm extrapolates the track and tries to find the cell of the hit from the coordinates of the point of the intersection of the track with the given station. And for that cell, only the denominator counter will be increased. Whereas, if the track has a hit in the given station, the algorithm will exclude this hit and checks if the track still meets the quality criteria. Thus, if the track meets the quality criteria without the excluded hit, both the numerator counter and the denominator counter will be increased for the cell of the hit. After that, for each cell in each station, the counters are summed for all tracks, and the efficiency is calculated by dividing the numerator by the denominator as shown in **Figure 7**.

```
for (size_t stf = 0; stf < STATIONS_NUMBER - 1; stf++)
{
    Int_t rect = 0, rect1 = 1, rect2 = NrectY;
    for (size_t rect0 = 0; rect0 <= NrectXY - 1; rect0++)
    {
       Float_t eff =
(Float_t)effUprect[stf][rect0]/(Float_t)effDorect[stf][rect0];
       if (std::isnan(eff))
          eff = 1.;
      myfile << stf << " " << rect0 << " " << eff << " " <<
effUprect[stf][rect0] << " " << effDorect[stf][rect0] << "\n";

       if (!std::isnan(eff) && eff > -0.5 && eff < 1.05 && effUprect[stf][rect0]
<= effDorect[stf][rect0])
          hist[stf]->SetBinContent(rect1, rect2, eff);
       cout << "SetBinContent " << rect1 << " " << rect2 << " " << eff << endl;

       if ((rect0 + 1) % NrectY == 0)
       {
          rect1++;
          rect2 = NrectY + 1;
        }
        rect2--;
        rect++;
     }
}
```

Figure 7: The algorithm for calculating the efficiency of each cell in each station.

The algorithm for obtaining the efficiency of Si/GEM stations for the π^- tracks in each run is located inside the /nica/mpd6/reem/NegEff/GEMeffRun7neg.C macro. This macro is a modified version of

the /nica/mpd6/plotnikov/run7_Ar_automation/efficiencysigem/GEMeffRun7.C macro. The reconstructed tracks that are used as an input for the GEMeffRun7neg.C macro are located inside the directory of /nica/mpd6/plotnikov/run7_Ar_automation/identification/ and take the form of {RUN_ID}.root where the {RUN_ID} is the run number which ranges from 3756 to 4704. The process of obtaining the efficiency for each run is automated using a bash shell script file (EfficiencyGEM.sh) in the /nica/mpd6/reem/NegEff/ directory to save time and effort. After the execution of the EfficiencyGEM.sh script, two files are created for each run, GEMeffRun7neg_{RUN_ID}.root and effneg{RUN_ID}.txt, and they are located in /nica/mpd6/reem/NegEff/EffGEMNeg directory. The GEMeffRun7neg_{RUN_ID}.root files contain 2-D histograms of the efficiencies for each station which can be seen in **Figure 8**, and the effneg{RUN_ID}.txt files contain the same output but in a text format. Then, in each station for each target, the numerator and denominator of each cell are summed for all runs, and the average efficiency is obtained by dividing the summed numerator by the summed denominator. The average efficiency is obtained using the SumEfficienciesGEM.C macro that is found in the /nica/mpd6/reem/NegEff/ directory, and the process is automated for the five targets using the EfficiencyAggregationGEM.sh script that is found in the latter directory.

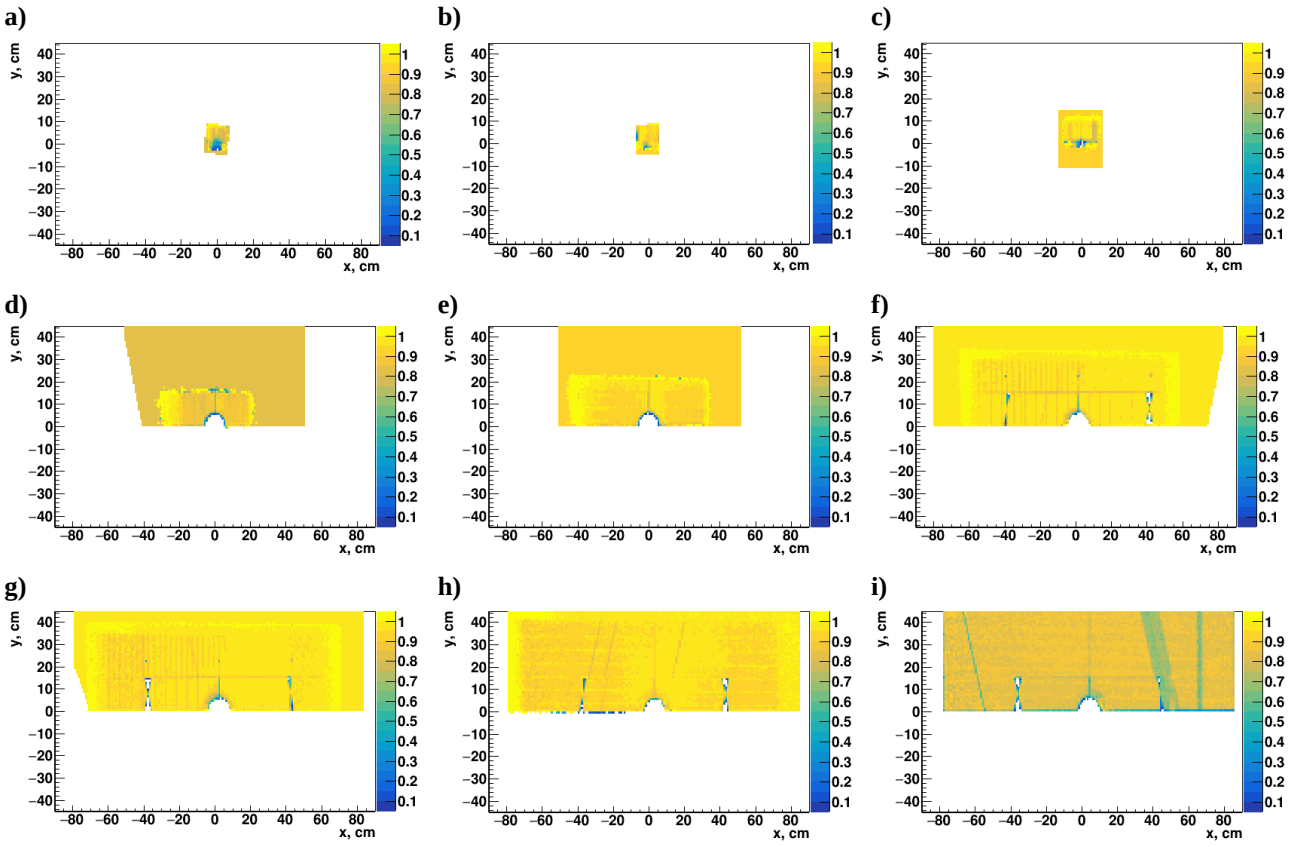


Figure 8: Two-dimensional efficiency histograms for negative tracks for: **a)** the first FwdSi station, **b)** the second FwdSi station, **c)** the third FwdSi station, **d)** the first GEM station, **e)** the second GEM station, **f)** the third GEM station, **g)** the fourth GEM station, **h)** the fifth GEM station, **i)** the sixth GEM station.

A comparison between the efficiencies of the Si/GEM stations for positive and negative tracks for the Copper target are given in **Figure 9**. Each panel in **Figure 9** was obtained by plotting the division the x-projections of the numerator by the x-projections of the denominator from the previously obtained 2-d numerator and denominator histograms to get the x-projections of the efficiencies for the negative and positive tracks on the same plot. It can be seen that for the first four

GEM stations, the efficiency for the negative tracks has a wider range in the x-axis than for the positive tracks.

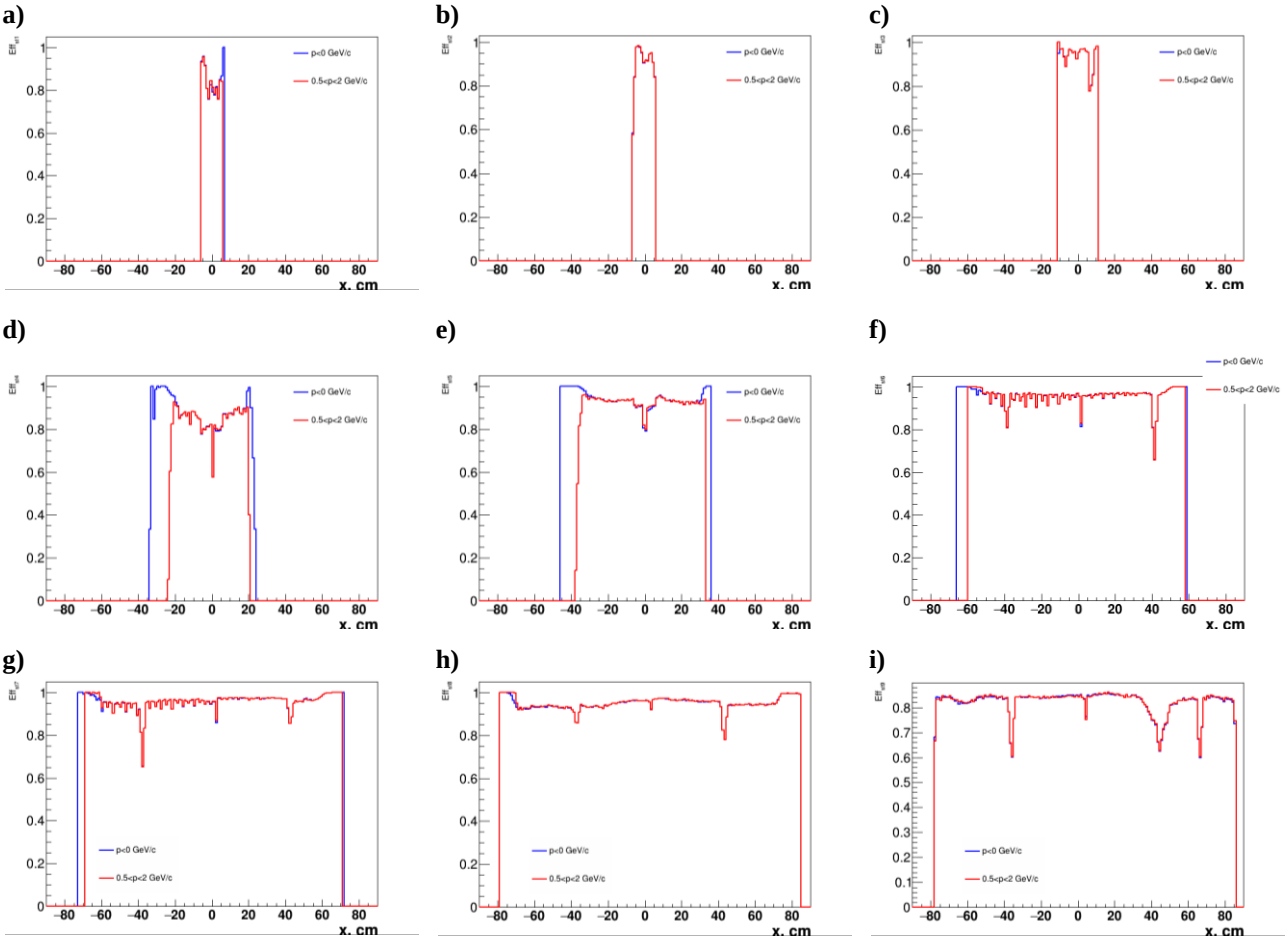


Figure 9: A comparison between the x-projection of the efficiencies for the negative tracks (blue line) and the positive tracks (red line) for: **a)** the first FwdSi station, **b)** the second FwdSi station, **c)** the third FwdSi station, **d)** the first GEM station, **e)** the second GEM station, **f)** the third GEM station, **g)** the fourth GEM station, **h)** the fifth GEM station, **i)** the sixth GEM station.

The next step is to compare the efficiencies of the Si/GEM for the experimental tracks with the MC reconstructed tracks. But first, we need to go through the process of reconstructing MC tracks. The MC model used to reconstruct tracks is DCM-QGSM which stands for Dubna Cascade Model and Quark Gluon String Model[15]. The reconstruction of MC tracks is done using the mainMcQsub.sh script that is found in /nica/mpd6/plotnikov/vp_r7_v2 directory. The reconstruction process starts with obtaining dcmqgsm_ArCu_<run>.sim.root files, and they contain the generated tracks. And then, these latter files are used as an input file to obtain dcmqgsm_ArCu_<run>_tmp1.tra.root files which contain Si, GEM, CSC digits in an intermediate format, which in turn are used to obtain dcmqgsm_ArCu_<run>_tmp2.tra.root files that contain Si, GEM, CSC digits, clusters, and hits in CBM format, as well as short tracks. The latter files play an important role in calculating the efficiencies of Si/GEM stations, as the suppression of hits is implemented during obtaining these files. This suppression of hits is done inside the CbmStsFindHits class using the IsDetected method. However, we will need reconstructed MC tracks without any hit suppression to get the normalized efficiencies of the Si/GEM stations. Thus, the Si/GEM hit suppression must be disabled in the CbmStsFindHits class and then obtain the dcmqgsm_ArCu_<run>_tmp2.tra.root files. After that, using the previous files, dcmqgsm_ArCu_<run>_tmp3.tra.root files are obtained which contain long tracks. The last step in the reconstruction of the MC tracks that will used in calculating the

efficiency is to obtain dcmqgsm_ArCu_<run>.tra.root files from the latter files. The final version of the track files contains two branches, one for all the long tracks and one for the long tracks that are confirmed in CSC.

The process for obtaining the Si /GEM efficiencies for MC tracks is similar to that of data experimental tracks. The same macros of GEMeffRun7neg.C and SumEfficienciesGEM.C are used to calculate the efficiencies for each run and to obtain the average efficiency for each station respectively.

Now, to obtain the normalized efficiencies of Si/GEM stations, EfficiencyNormalizeDataToMC.C macro is executed using EfficiencyNormalizeDataToMC.sh script for all targets. Then, the obtained normalized efficiencies are implemented inside the CbmStsFindHits class, and we need to repeat the process of getting the MC reconstructed tracks with enabling the Si/GEM hit suppression and calculate the new Si/GEM efficiencies for the newly obtained MC tracks.

To obtain a good matching between the Si/GEM efficiencies of experimental data and MC tracks, we need to calculate the difference between the data and MC efficiencies and then apply the difference to the MC efficiencies inside the CbmStsFindHits class. A visual representation of calculating the difference between the efficiencies is given in **Figure 10**.

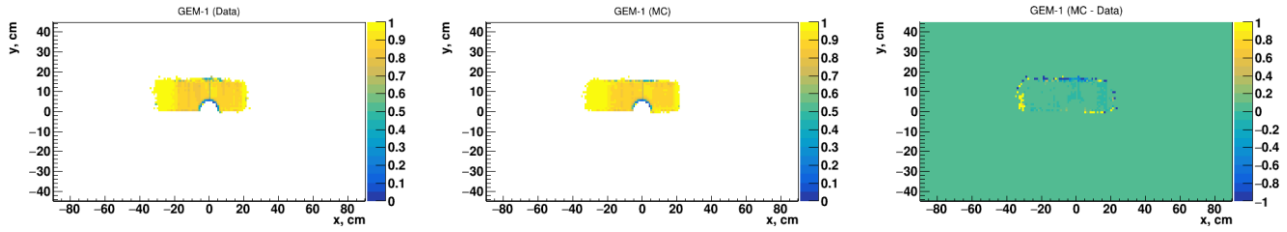


Figure 10: **Left**: 2-D histogram of the efficiency of the first GEM station for the real experimental reconstructed tracks. **Center**: 2-D histogram of the efficiency of the first GEM station for the MC simulated tracks. **Right**: 2-D histogram of the difference of the two efficiencies of the first GEM station.

The CbmStsFindHits class contained previous adjustments to the efficiencies that was applied manually to the x and y ranges for positive tracks. These values were also used as a base adjustment for the Si/GEM efficiencies for the negative tracks. However, in addition to the base adjustments, the further difference between the efficiencies was applied to the hit suppression using the following lines.

```
TFile *fdiff = new TFile("/nica/mpd6/reem/NegEff/diff/output.root","read");

TH2F *diff = (TH2F *)fdiff->Get(Form("difference%d", stationNr-1));

int binx = diff->GetXaxis()->FindBin(posX);
int biny = diff->GetYaxis()->FindBin(posY);
double d = diff->GetBinContent(binx,biny);

eff -= d;
```

Figure 11: The algorithm for reading the difference file and applying this difference to the Si/GEM efficiency for MC simulated tracks.

The matching between the Si/GEM efficiencies for the experimental data and MC simulated data before the applying the additional adjustments using the difference values is shown in **Figure 12**. For more precise matching of the efficiencies, multiple iterations may be needed. For example, after the first iteration, the difference between the newly obtained Si/GEM efficiencies for MC tracks and the those of the experimental data tracks are calculated and applied to the CbmStsFindHits class

along with the previous differences. Then, new MC tracks and efficiencies are obtained. If the new matching is not sufficient, more iterations can be done in a similar way. In **Figure 13**, the matching between the Si/GEM efficiencies of experimental data and MC tracks after 3 iterations can be seen.

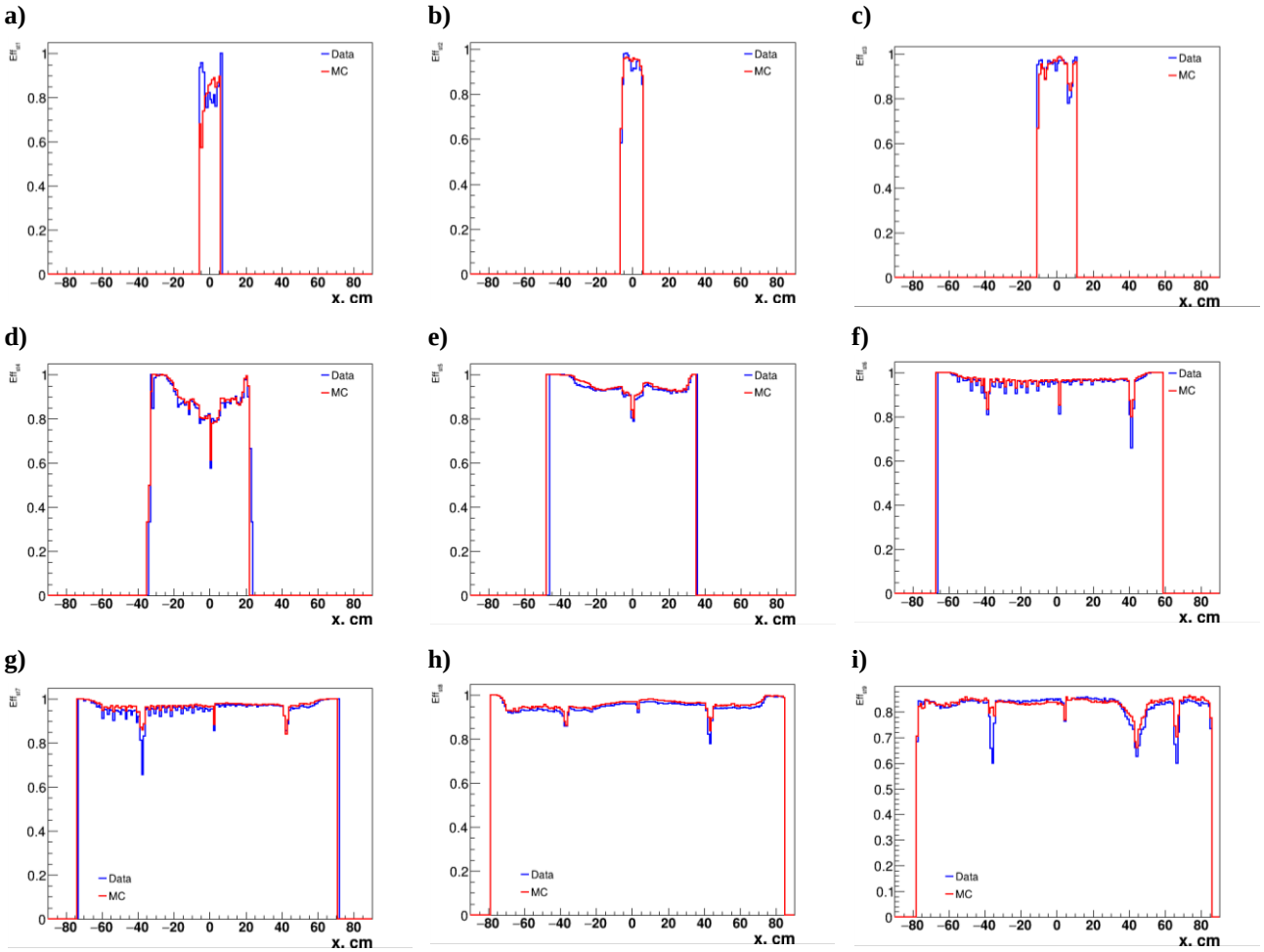
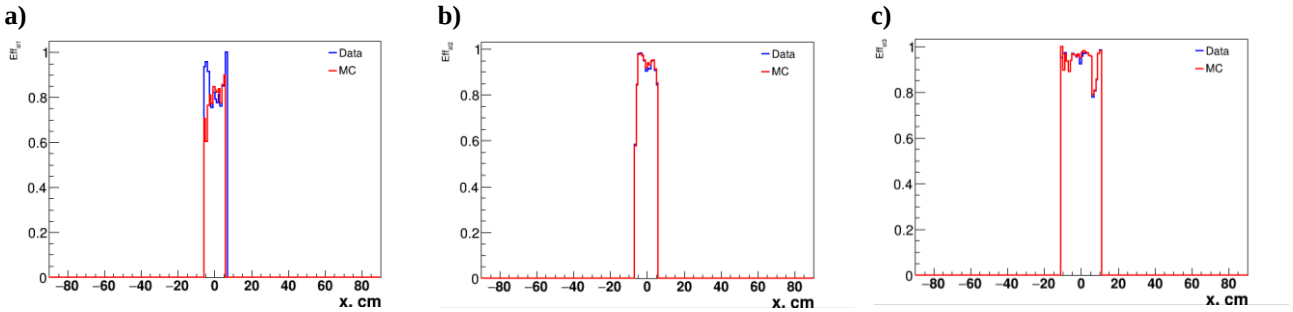


Figure 12: A comparison between the x-projection of the efficiencies for real experimental data (blue line) and the MC simulated data (red line) for negative tracks before additional adjustments for: **a)** the first FwdSi station, **b)** the second FwdSi station, **c)** the third FwdSi station, **d)** the first GEM station, **e)** the second GEM station, **f)** the third GEM station, **g)** the fourth GEM station, **h)** the fifth GEM station, **i)** the sixth GEM station.



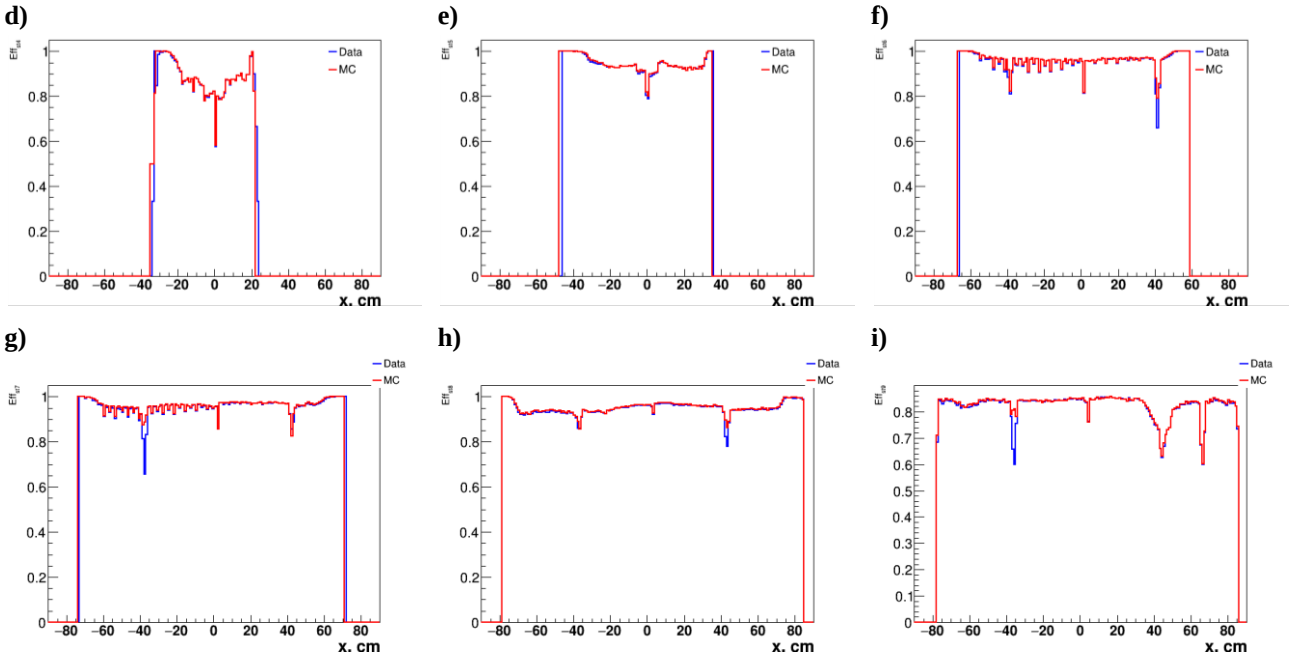


Figure 13: A comparison between the x-projection of the efficiencies for real experimental data (blue line) and the MC simulated data (red line) for negative tracks after three iterations for: **a)** the first FwdSi station, **b)** the second FwdSi station, **c)** the third FwdSi station, **d)** the first GEM station, **e)** the second GEM station, **f)** the third GEM station, **g)** the fourth GEM station, **h)** the fifth GEM station, **i)** the sixth GEM station.

A good agreement between MC and real experimental data is observed for the Si/GEM efficiencies calculated with negative tracks. It can be seen from the latter figures that the average efficiency for Si and GEM stations for π^- tracks ranges from 80 to 95%.

6. Conclusion

BM@N is a fixed target experiment aimed to study the relativistic nucleus-nucleus collisions. The experimental set-up of the BM@N consisted of several systems of detectors. However, the focus of this report is on the central tracking system, the FwdSi and GEM stations. The process of obtaining the efficiency for each station in the central tracking system was preceded by developing a good understanding of many concepts of C++ and getting familiar with ROOT. The average efficiency of Si/GEM stations obtained for negative tracks was found to range from 80 to 95%. This result agrees with the previously obtained efficiencies for positive tracks. It can be noted that for negative tracks, the efficiencies have a wider range in the x-axis for the first four GEM stations than for the positive tracks. Moreover, the real data efficiencies showed a good agreement with the efficiencies for the MC simulated data.

7. References

1. *The Report on Project "Studies of Baryonic Matter at the Nuclotron (BM@N)."*; 2021.
2. Gornaya J, Kapishin M, Plotnikov V, et al. Hyperons at the BM@N experiment: first results. *EPJ Web Conf*. 2019;204:1-9. doi:10.1051/epjconf/201920401006
3. Alishina K, Plotnikov V, Kovachev L, Petukhov Y, Rumyantsev M. Charged Particle Identification by the Time-of-Flight Method in the BM@N Experiment. *Phys Part Nucl*. 2022;53(2):470-475. doi:10.1134/S106377962202006X
4. *Studies of Baryonic Matter at the Nuclotron.*; 2019. https://bmn.jinr.ru/wp-content/uploads/2019/07/BMN_Project.pdf

5. Batyuk P, Baranov D, Merts S, Rogachevsky O. Event reconstruction in the BM@N experiment. *EPJ Web Conf*. 2019;204:1-7. doi:10.1051/epjconf/201920407012

6. BM@N collaboration. *BmnRoot Simulation and Analysis Framework for the BM@N Experiment Start Guide*.; 2016. https://bmn.jinr.ru/wp-content/uploads/2019/07/BmnRoot_Start_Guide.pdf

7. ROOT team. ROOT macros and shared libraries. https://root.cern/manual/root_macros_and_shared_libraries/

8. Stroustrup B. *The C++ Programming Language*. Fourth.; 2013. doi:10.1145/954127.954144

9. Stanley B. Lippman, Josee LaJoie BEM. *C++ Primer*. Fifth.; 2012.

10. Chapter: A Little C++. ROOT. https://root.cern.ch/root/htmldoc/guides/users-guide/ALittleC++.html

11. C++ Preprocessor. W3schools. https://www.w3schools.in/cplusplus/preprocessor

12. Preprocessor directives. Microsoft. Published 2021. https://docs.microsoft.com/en-us/cpp/preprocessor/preprocessor-directives?view=msvc-170

13. Chandra A. Preprocessor Directives in C++. Published 2022. https://www.scaler.com/topics/cpp/cpp-preprocessor-directives/

14. An Intro to SGE. http://talby.rcs.manchester.ac.uk/~ri/_linux_and_hpc_lib/sge_intro.html

15. Baznat M, Botvina A, Musulmanbekov G, Toneev V, Zhezher V. Monte-Carlo Generator of Heavy Ion Collisions DCM-SMM. *Phys Part Nucl Lett*. 2020;17(3):303-324. doi:10.1134/S1547477120030024

8. **Acknowledgments**

First and foremost, I would like to express my deepest appreciation to Mr. Vasilii Plotnikov for his invaluable supervision, support, and patience. He has been a tremendous source of support for me throughout the course of this project. He has always been available to answer any questions that I had and provided me with assistance whenever I needed it. I would like to extend my sincere thanks to the START program coordinators, Ms. Elena Karpova and Ms. Julia Rybachuk, for their warm welcome and support. Finally, I would like to thank the Joint Institute for Nuclear Research (JINR) for providing me with the opportunity to participate in this program. My experience at JINR has been truly rewarding and I am very grateful for everything that I have learned during my stay here.