BELLE2-NOTE-PL-2018-036
DRAFT Version 1.0
December 7, 2018

Validation VXD alignment in phase 2 using collision data

J. Kandra*

Charles University, Prague, Czech Republic

Abstract

At beginning of phase 2 VXD alignment was determined and the first VXD alignment was published in BELLE2-NOTE-TE-2018-006. After the Belle 2 detector collect large number collision data the alignment procedure was repeated. The first alignment was checked and improvements was applied to Belle 2 analysis and software for further analysis. The full of data collected during phase 2 was analysed for VXD alignment purpose. We were looked for stability of VXD alignment constants and running monitoring parameters in time. Validation VXD alignment were done using track-to-hit residuals. The figures show residuals in both directions using runs for alignment determination and running of means of residuals as function of other runs (time).

*Electronic address: jakub.kandra@karlov.mff.cuni.cz

Contents

1. SUMMARY

- Data set: Experiment 3 and Runs:
  - For determination VXD alignment: 577, 578, 579, 580, 674, 677, 686, 782, 783, 785, 786
  - For validation VXD alignment: 3418, 3420, 3421, 3422, 3423, 3443, 3446, 3448, 3449, 3460, 3461, 3462, 3463, 3521, 3534, 3547, 3625, 3645, 3706, 3707, 3770, 3772, 3773, 3785, 3787, 3788, 3789, 3790, 3981, 3983, 3992, 3994, 4035, 4074, 4076, 4084, 4085, 4087, 4088, 4089, 4173, 4174, 4187, 4334, 4339, 4340, 4343, 4389, 4393, 4395, 4397, 4398, 4399, 4437, 4439, 4440, 4441, 4442, 4559, 4564, 4577, 4579, 4679, 4687, 4688, 4691, 4692, 4694, 4791, 4796, 4798, 4799, 4814, 4941, 4944, 4945, 4947, 4948, 4953, 4956, 4957, 4958, 5007, 5009, 5010, 5013, 5014, 5015, 5016, 5017, 5019, 5100, 5187, 5194, 5239, 5240, 5241, 5324, 5325, 5466, 5468, 5469, 5607, 5613
- Event selection:
  - For determination VXD alignment: at least one track with 3 VXD hits.
  - For validation VXD alignment:
    1. at least one track with 3 VXD hits
    2. SVD and PXD hits associated with track
- Release: Master at 3b34985a7be1e78695bf357ccf01c68a42d0a283
- Database: Global tag "Calibration_Offline_Development"
- Geometry: phase 2 geometry and magnetic field "MagneticFieldPhase2"
- No Monte Carlo simulation
- Minimal SVD cluster time is set to -999.
- Hot pixel masking algorithm was applied, but the SVD is without masking on offline software level.

- Used raw data can be found in phase 2 directory at KEKCC
- Used scripts can be found in "/home/belle2/jkandra/basf2/beam/alignment/phase2/data"
- Integrated luminosity of used runs:

Determination VXD alignment

Run	577	578	579	580	674	677	686	782	783	785	786
Luminosity [nb^{-1}]	111	379	201	141	∅	∅	∅	115	322	∅	100

Validation VXD alignment

Run	3418	3420	3421	3423	3443	3448	3449	3460	3461	3462	3463
Luminosity [nb^{-1}]	1189	1276	1410	2265	1464	1704	1196	1657	1490	1083	878
Runs	3521	3534	3547	3625	3645	3706	3707	3770	3772	3773	3785
Luminosity [nb^{-1}]	∅	1240	1490	228	450	1011	1134	1601	1290	1429	498
Runs	3787	3788	3789	3790	3981	3983	3992	3994	4035	4074	4076
Luminosity [nb^{-1}]	1051	1654	677	820	1601	1662	1358	1869	1075	1864	2386
Runs	4084	4085	4087	4088	4089	4173	4174	4187	4334	4339	4340
Luminosity [nb^{-1}]	1998	2775	2243	2620	2444	∅	1533	793	551	815	1024
Runs	4343	4389	4393	4395	4397	4398	4399	4437	4439	4440	4441
Luminosity [nb^{-1}]	706	1906	∅	2437	1523	2549	1905	2315	1799	729	1687
Runs	4442	4559	4564	4577	4579	4679	4687	4688	4691	4692	4694
Luminosity [nb^{-1}]	1240	∅	∅	∅	∅	625	420	466	522	388	352
Runs	4791	4796	4798	4799	4814	4941	4944	4945	4947	4948	4956
Luminosity [nb^{-1}]	882	3244	1110	1339	1700	1075	1103	1980	809	906	2917
Runs	4957	4958	5007	5009	5010	5013	5014	5016	5017	5019	5100
Luminosity [nb^{-1}]	1406	1773	404	1431	1087	1460	1976	2520	1185	1187	1604
Runs	5187	5194	5239	5240	5241	5324	5325	5466	5468	5469	5607
Luminosity [nb^{-1}]	3064	2959	2888	3220	4395	2431	1789	2147	1689	2369	2825
Runs	5613	**Total luminosity**									
Luminosity [nb^{-1}]	2501	**147211**									

2. FIGURES

The validation figures (from 1 to 6) are done for each VXD sensor. Rows represent two measurements in local coordinates of VXD sensor plane. In left column it can be seen track-to-hit residual filled data used in alignment procedure and right column shows validation results as running or fluctuating means of track-to-hit residuals as function of run number (time).

The validation plots shows some unexpected behaviour e.g. in figure 2 run 3625 or figure 3 for runs from 4334 to 4393. These effects can be explain by low statistics or changing of beam (collisions) conditions.

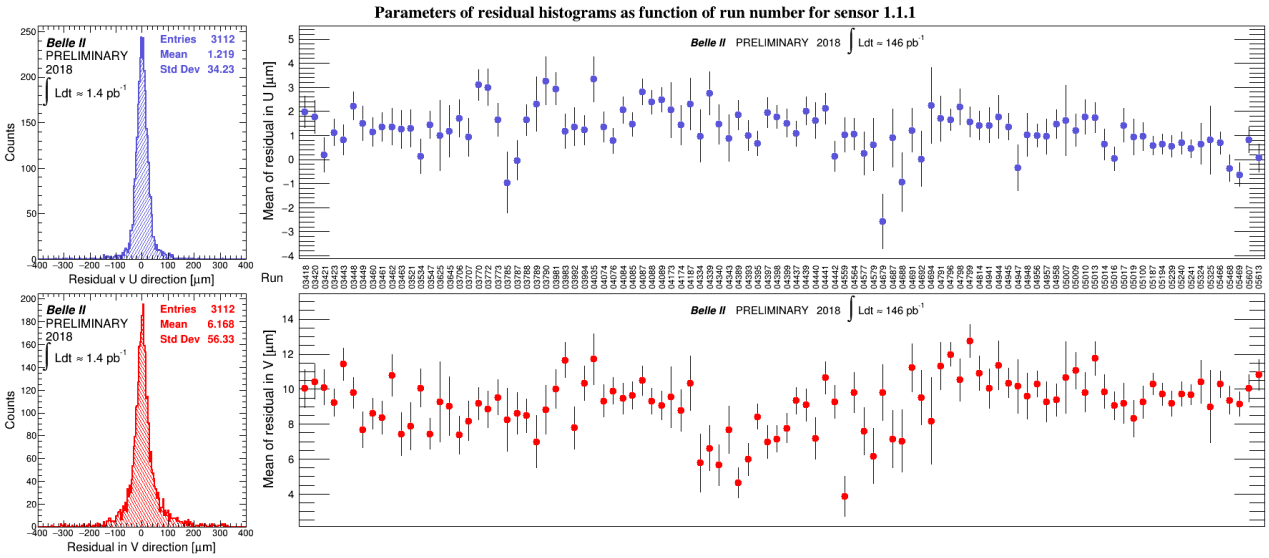

FIG. 1: Validation plot for 1.1.1 sensor

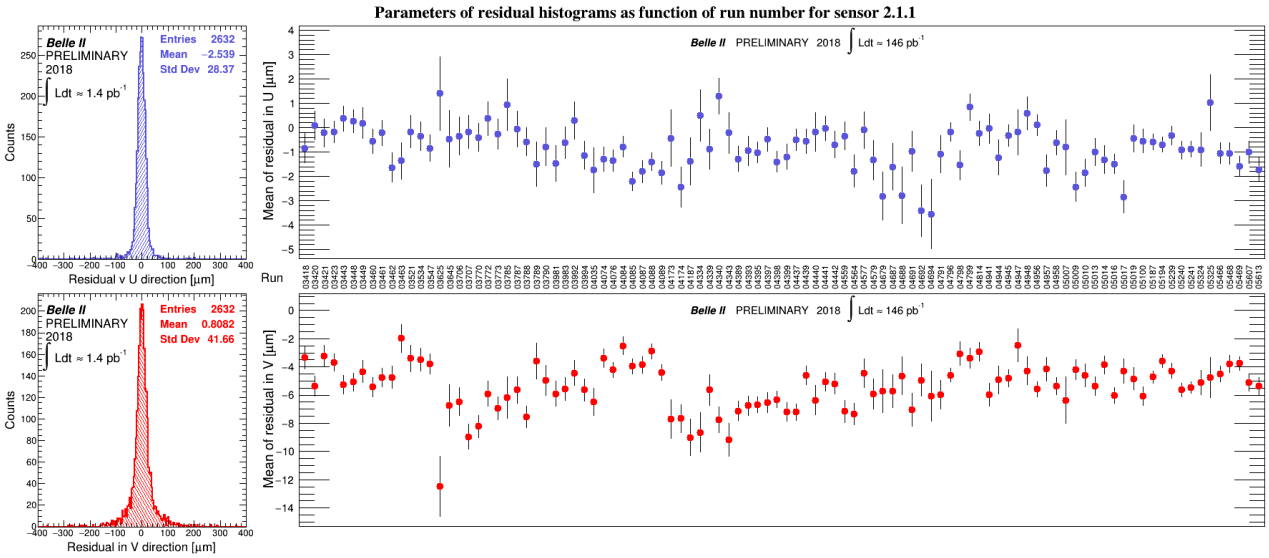

FIG. 2: Validation plot for 2.1.1 sensor

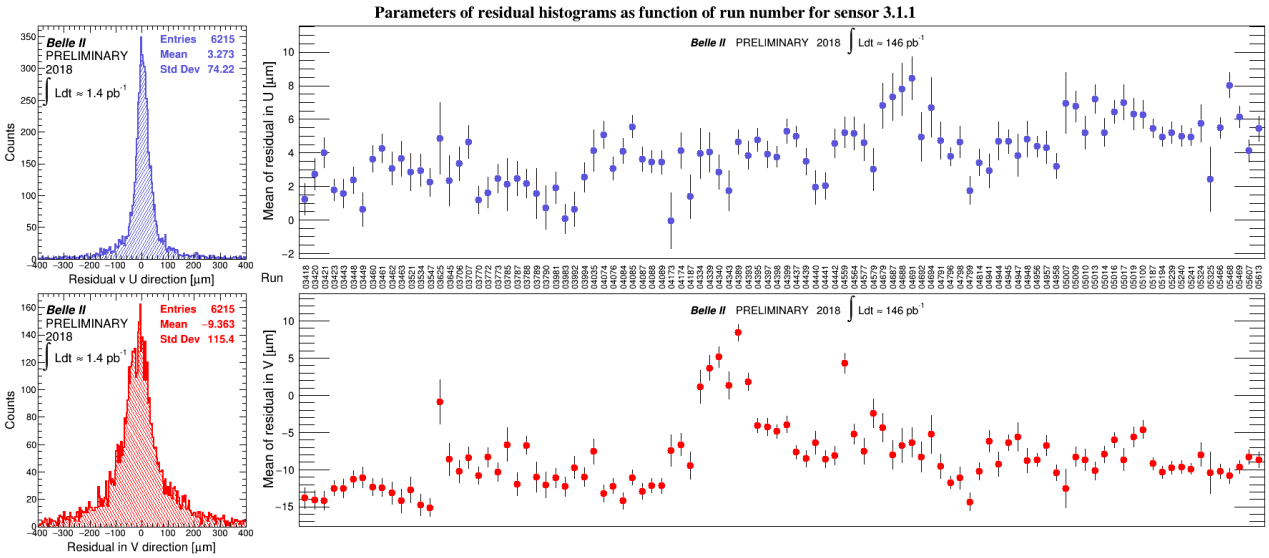

FIG. 3: Validation plot for 3.1.1 sensor

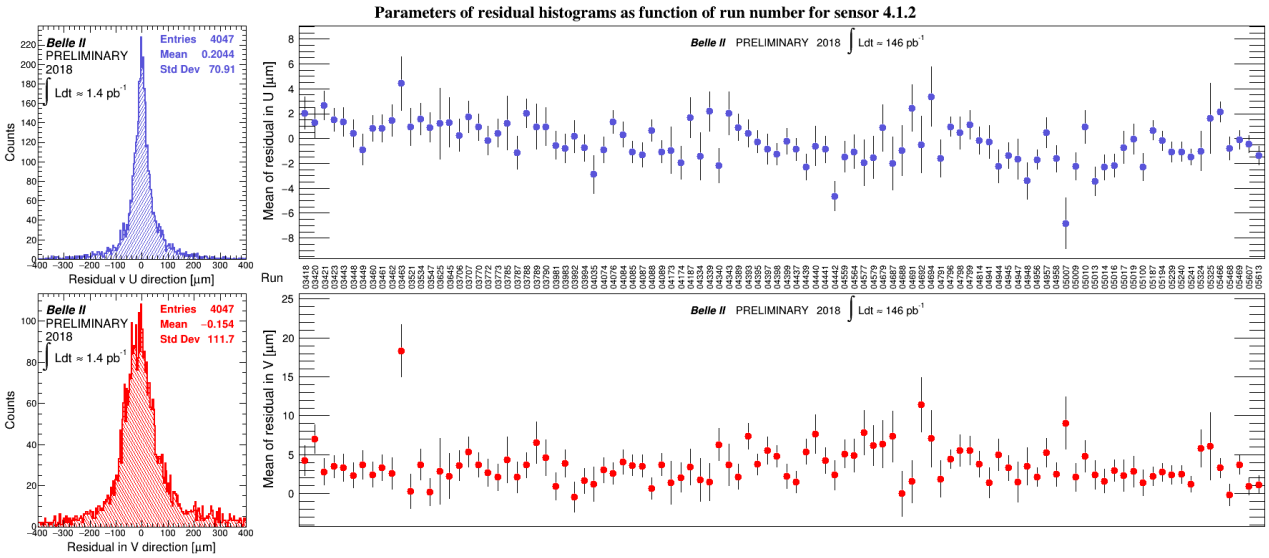

FIG. 4: Validation plot for 4.1.2 sensor

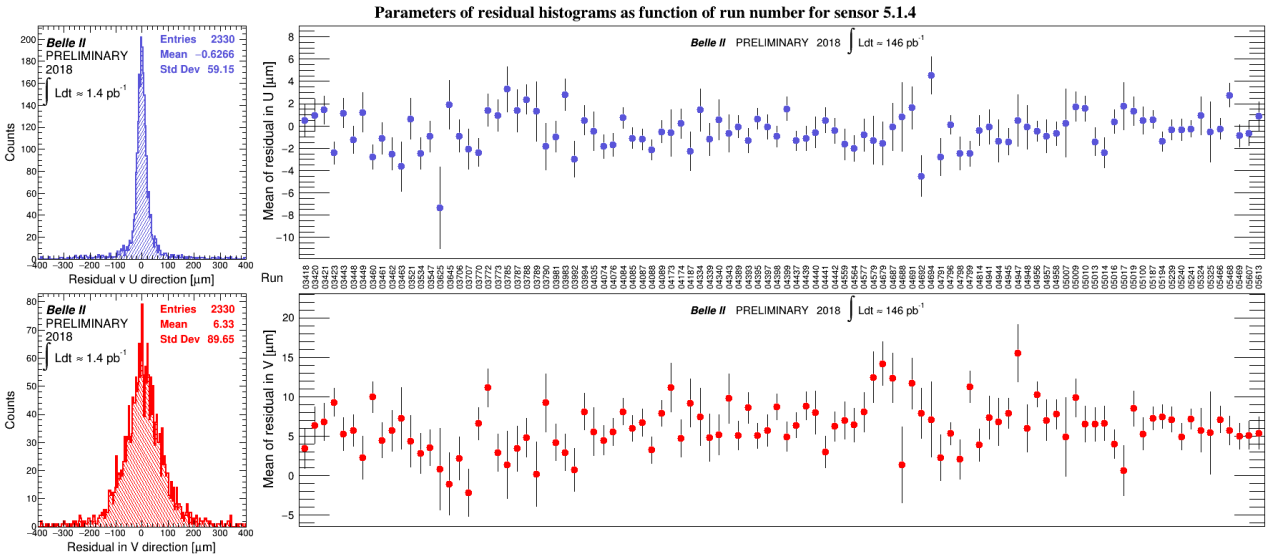

FIG. 5: Validation plot for 5.1.4 sensor

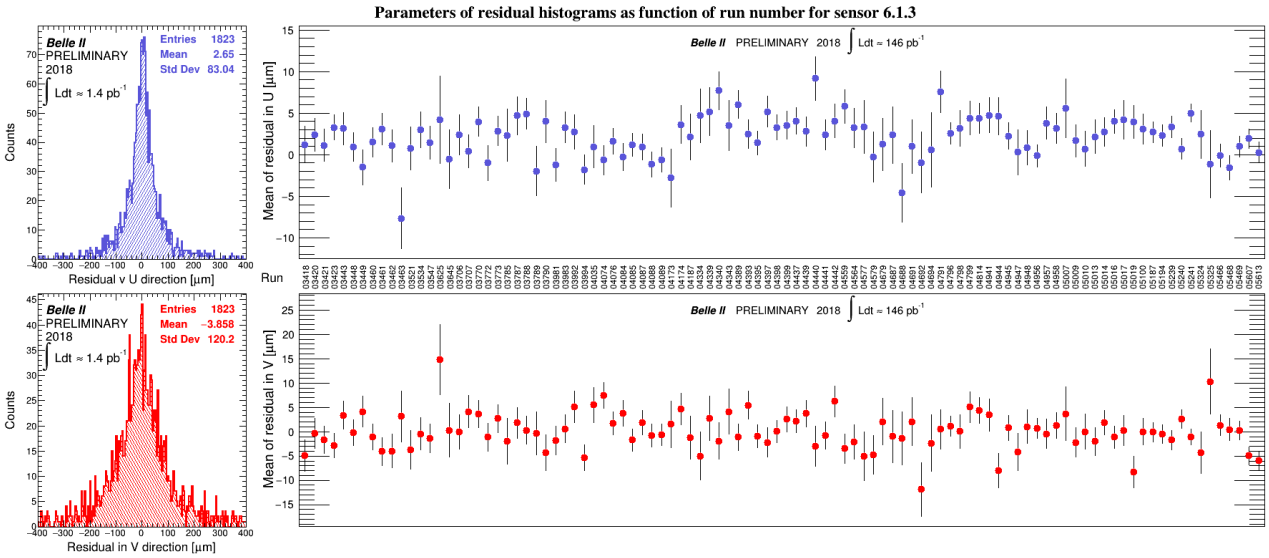

FIG. 6: Validation plot for 6.1.3 sensor

3. PROPOSED FIGURE LABEL

Validation plot shows stability (running) means of track-to-hit residuals as function of runs (time) for each VXD sensor. The validation plot is divided to four histograms. Left column of histograms present track-to-hit residuals of tracks used for determination of alignment parameters. Right column describe changing means of residuals depends on run number. Top row of histograms shows results for u local coordinate of measurement. Bottom row demonstrates results for v local coordinate.

4. ADDITIONAL INFORMATION

We were used basf2 at master branch with commit 3b34985a7be1e78695bf357ccf01c68a42d0a283. We were read database object from global tag (GT) "Calibration_Offline_Development". We were used PXD data after offline masking. For SVD it is not applied masking procedure.

Working procedure from raw data at KEKCC to validation plots are divided to several parts:

- Script for selection good candidates for VXD validation
- Main reconstruction steering script
- Submission and merging script
- Script for plotting validation figures

4.1. The selection script ("EventSelectorBeamData.py")

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import sys
import math
import os
from modularAnalysis import *
from basf2 import *
import ROOT
from ROOT import Belle2

class EventSelectorBeamData(Module):

    """
    Python module to select collision data
    (which pass VXD volume).
    """

    def __init__(self):
        """ init """
        super(EventSelectorBeamData, self).__init__()

    def event(self):
        """ Return True if event is fine, False otherwise """

        someOK = False
        someOK2 = False

        RecoTracksArray = Belle2.PyStoreArray('RecoTracks')
        TracksArray = Belle2.PyStoreArray('Tracks')

        TrackFitResultsArray = Belle2.PyStoreArray('TrackFitResults')
        if RecoTracksArray.getEntries() > 1:
```

```
            for recoTrack in RecoTracksArray:
                if recoTrack.hasCDCHits():
                    if recoTrack.hasPXDHits() and recoTrack.hasSVDHits(): # For validation
                    #if recoTrack.hasPXDHits() or recoTrack.hasSVDHits(): # For alignment validation
                        for TrackFitResult in TrackFitResultsArray:
                            nVXDLayers = TrackFitResult.getHitPatternVXD().getNVXDLayers()
                            if nVXDLayers >= 3:
                                someOK = True
                                break

        if someOK:
            if not someOK:
                EventMetaData = Belle2.PyStoreObj('EventMetaData')
                event = EventMetaData.getEvent()
                print('Event', event, 'has two RecoTracks and PXD or SVD hits. It will be stored.')

        super(EventSelectorBeamData, self).return_value(someOK)
```

4.2. The main reconstruction steering script was ("1_AnalyzeKEKCCPhysics.py")

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from basf2 import *
from ROOT import Belle2
import rawdata
from svd import add_svd_reconstruction
from pxd import add_pxd_reconstruction
import tracking
import reconstruction
from EventSelectorBeamData import EventSelectorBeamData
import datetime

reset_database()
use_database_chain()
use_central_database("Calibration_Offline_Development")
use_local_database("/home/belle2/jkandra/basf2/beam/alignment/phase2/data/MagneticFieldPhase2/dbcache.txt")

main = create_path()
main.add_module('RootInput', inputFileName =
'/ghi/fs01/belle2/bdata/Data/Raw/e000'+exp+'/r'+run+'/sub00/physics.000'+exp+'.'+run+'.HLT'+hlt+'.f'+part+'.root',
excludeBranchNames=['Tracks', 'TrackFitResults'], entrySequences=['0:8000'])
main.add_module('Gearbox', fileName='/geometry/Beast2_phase2.xml')
components = [
    'BeamPipe',
    'MagneticField',
    'PXD',
    'SVD',
    'CDC',
        'ARICH',
    'ECL'
    ]

main.add_module('Geometry', components=components)
rawdata.add_unpackers(main, components=components)
main.add_module("ActivatePXDPixelMasker")
add_svd_reconstruction(main, applyMasking=True)
add_pxd_reconstruction(main)

tracking.add_geometry_modules(main, components=components)

tracking.add_tracking_reconstruction(main,
                                     components=components,
                                     pruneTracks=False,
                                     skipGeometryAdding=False)

main.add_module('Ext')

reconstruction.add_ecl_modules(main, components)
main.add_module('ECLTrackShowerMatch')
main.add_module('ECLElectronId')

store = create_path()
eventSelectorBeamData=EventSelectorBeamData()
eventSelectorBeamData.if_true(store, AfterConditionPath.CONTINUE)
main.add_module(eventSelectorBeamData)

store.add_module('HistoManager',histoFileName =
'/ghi/fs01/belle2/bdata/group/detector/VXD/phase2/alignment/data/DQM/beam.'+exp+'.'+run+'.'+hlt+'.'+part+'.root')
store.add_module('AlignDQM', TracksStoreArrayName="Tracks", RecoTracksStoreArrayName="RecoTracks")

main.add_module('Progress')
process(main)
print(statistics)
```

4.3. The script used for submission jobs at KEKCC and merging partial root files

```
#!/bin/bash
```

```
# Run numbers with number of files useful in validation
runs=("3418" "3419" "3420" "3421" "3422" "3423" "3443" "3446" "3448" "3449" "3460" "3461" "3462" "3463" "3521" "3534")
parts=("10" "11" "11" "13" "11" "20" "13" "11" "15" "10" "15" "13" "9" "9" "9" "8")
#runs=("3547" "3625" "3645" "3706" "3707" "3770" "3772" "3773" "3785" "3787" "3788" "3789" "3790" "3981" "3983" "3992")
#parts=("10" "10" "13" "24" "26" "32" "24" "27" "13" "16" "27" "11" "13" "21" "21" "16")
#runs=("3994" "4035" "4074" "4076" "4084" "4085" "4087" "4088" "4089" "4173" "4174" "4187" "4334" "4339" "4340" "4343")
#parts=("24" "12" "14" "15" "15" "19" "15" "17" "16" "9" "22" "11" "9" "15" "16" "8" "16")
#runs=("4389" "4393" "4395" "4397" "4398" "4399" "4437" "4439" "4440" "4441" "4442" "4559" "4564" "4577" "4579" "4679")
#parts=("13" "22" "14" "23" "17" "23" "15" "9" "14" "12" "39" "33" "20" "17" "19")
#runs=("4687" "4688" "4691" "4692" "4694" "4791" "4796" "4798" "4799" "4814" "4941" "4944" "4945" "4947" "4948" "4953")
#parts=("15" "18" "14" "13" "12" "12" "44" "14" "17" "26" "11" "16" "21" "10" "9" "21" "37" "24" "18")
#runs=("4956" "4957" "4958""5007" "5009" "5010" "5013" "5014" "5015" "5016" "5017" "5019" "5100" "5187" "5194" "5239")
#parts=("37" "24" "18" "8" "17" "12" "17" "26" "13" "31" "15" "13" "15" "9" "14" "21")
#runs=("5240" "5241" "5324" "5325" "5466" "5468" "5469" "5607" "5613")
#parts=("26" "37" "11" "8" "10" "9" "11" "15" "19")

# Runs used for determination alignment parameters of VXD sensors
#runs=("577" "578" "579" "580" "674" "677" "686" "782" "783" "785" "786")
#parts=("3" "12" "6" "4" "2" "2" "8" "3" "9" "8" "6")

for i in {0..11}; do # loop over members in fields <runs> and <parts>
    for j in {2..4}; do # loop over HLT files
        printf -v run5 "%05d" ${runs[$i]} # from XXX to 00XXX
        for part5 in $(seq -f "%05g" 0 ${parts[$i]}); do # loop over all used files
            bsub -o ./log/log_${run5}_${part5}_${j}.txt basf2 1_AnalyzeKEKCCPhysics.py 3 $run5 $part5 $j
        done
    done
done

cd /ghi/fs01/belle2/bdata/group/detector/VXD/phase2/alignment/data/DQM/

for i in {0..6}; do
    printf -v run5 "%05d" ${runs[$i]}

    array=()
    array+="hadd -f Runs/beam.3.${run5}.root "

    for j in {2..4}; do
        for part5 in $(seq -f "%05g" 0 ${parts[$i]}); do
            array+="beam.3.${run5}.${j}.${part5}.root "
        done
    done

    echo "${array[*]}"
    command="$(${array[*]})"
    echo "${command}"
done
```

4.4. The script for creating validation plots from produced root files per each run

```
#include <sstream>
#include <string>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <array>
#include <limits>

#include "Belle2Style.C"
#include "Belle2Labels.C"
#include "Belle2Utils.C"

void MyAnalysis(){

  const int number = 112; // 170
  const int alignedNumber = 11;
  const int variable = 4; // 6
  bool graph = false;
  bool graphSensor = true;

  const int color[15] = {9, 2, 8, 94, 6, 46, 1, 28, 42, 38, 51, 100, 14};

  string alignedRuns[alignedNumber] = {"00577", "00578", "00579", "00580", "00674",
                                       "00677", "00686", "00782", "00783", "00785", "00786"};
  string runs[200] = {"03290", "03291", "03298", "03299", "03341", // 5
                      "03348", "03349", "03391", "03395", "03397", "03398",  // 6
                      "03417", "03418", "03420", "03421", "03423", "03443",
                      "03448", "03449", "03460", "03461", "03462", "03463", // 12
                      "03521", "03534", "03547", // 3
                      "03625", "03645", // 2
                      "03706", "03707", "03770", "03772", "03773", "03785", "03787", "03788", "03789", "03790", // 10
                      "03981", "03983", "03992", "03994", // 4
                      "04035", "04074", "04076", "04084", "04085", "04087", "04088", "04089", // 9
                      "04173", "04174", "04187", // 3
                      "04334", "04339", "04340", "04343", "04389", "04393", "04395", "04397", "04398", "04399", // 10
                      "04437", "04439", "04440", "04441", "04442", // 5
                      "04559", "04564", "04577", "04579", // 4
                      "04679", "04687", "04688", "04691", "04692", "04694", // 6
                      "04791", "04796", "04798", "04799", // 2
                      "04814", // 1
                      "04941", "04944", "04945", "04947", "04948", "04956", "04957", "04958", // 8
                      "05007", "05009", "05010", "05013", "05014", "05016", "05017", "05019", // 8
```

```
                    "05100", "05187", "05194", // 3
                    "05239", "05240", "05241", // 3
                    "05324", "05325",
                    "05466", "05468", "05469",
                    "05607", "05613"
                    };

TFile * alignedFile[alignedNumber];
TFile * file[number];

for (int i = 0; i < alignedNumber; i++){
  string fileName = "beam.3." + alignedRuns[i] + ".root";
  alignedFile[i] = new TFile(fileName.c_str());
  cout << fileName << "\n";
}

for (int i = 0; i < number; i++){
  string fileName = "beam.3." + runs[i] + ".root";
  file[i] = new TFile(fileName.c_str());
  cout << fileName << "\n";
}

TFile *result = new TFile("residuals.root", "recreate");

string label_direction[2] = {"U", "V"};
string label_sensor[18] = {"1_1_1", "1_1_2", "2_1_1", "2_1_2", "3_1_1", "3_1_2", "4_1_1", "4_1_2",
                           "4_1_3", "5_1_1", "5_1_2", "5_1_3", "5_1_4", "6_1_1", "6_1_2", "6_1_3", "6_1_4", "6_1_5"};
string label_plot[18] =   {"1.1.1", "1.1.2", "2.1.1", "2.1.2", "3.1.1", "3.1.2", "4.1.1", "4.1.2",
                           "4.1.3", "5.1.1", "5.1.2", "5.1.3", "5.1.4", "6.1.1", "6.1.2", "6.1.3", "6.1.4", "6.1.5"};

static TStyle* belle2Style =  Belle2Style();
gROOT->SetStyle("BELLE2");
gROOT->ForceStyle();

TH1F * alignedHistogram[18][2][alignedNumber];
TH1F * alignedResult[18][2];
TPaveStats * statistics[2];
float alignedSigma[18][2][11], alignedMean[18][2][11], alignedSigmaError[18][2][11], alignedMeanError[18][2][11];
for (int l = 0; l < 18; l++){
  for (int j = 0; j < 2; j++){
    TList * list = new TList;
    for (int i = 0; i < alignedNumber; i++){
        string histogram_label = "AlignmentDQMSensors/Residuals1D/Alig_UBResiduals"
                                 + label_direction[j] + "_" + label_sensor[l];
        alignedHistogram[l][j][i] = (TH1F*)alignedFile[i]->Get(histogram_label.c_str());
        alignedHistogram[l][j][i]->SetDirectory(0);
        list->Add(alignedHistogram[l][j][i]);
    stringstream word_j;
    stringstream word_l;
    string number_j;
    string number_l;
    word_j << j;
    word_j >> number_j;
    word_l << l;
    word_l >> number_l;
    string result_label = "alignedResult[" + number_l + "][" + number_j + "]";
    alignedResult[l][j] = (TH1F*)alignedHistogram[l][j][0]->Clone(result_label.c_str());
    alignedResult[l][j]->Reset();
    alignedResult[l][j]->Merge(list);
  }
}

TH1F * histogram[18][2][number];
//TPaveStats * statistics[2];
float sigma[18][2][number], mean[18][2][number], sigmaError[18][2][number], meanError[18][2][number];
for (int i = 0; i < number; i++){
  for (int j = 0; j < 2; j++){
    for (int l = 0; l < 18; l++){
      string histogram_label = "AlignmentDQMSensors/Residuals1D/Alig_UBResiduals"
                               + label_direction[j] + "_" + label_sensor[l];
      histogram[l][j][i] = (TH1F*)file[i]->Get(histogram_label.c_str());
      //sigma[l][j][i] = histogram[l][j][i]->GetStdDev();
      mean[l][j][i] = histogram[l][j][i]->GetMean();
      //sigmaError[l][j][i] = histogram[l][j][i]->GetStdDevError();
      meanError[l][j][i] = histogram[l][j][i]->GetMeanError();
      //mean[l][j][i] = histogram[l][j][i]->GetEntries();
    }
  }
}

if (graphSensor){
  auto uRMS = new TMultiGraph();
  auto vRMS = new TMultiGraph();
  auto uMean = new TMultiGraph();
  auto vMean = new TMultiGraph();

  TCanvas * sensors[18];

  for (int j = 0; j < 18; j++){

    auto uRMS = new TMultiGraph();
    auto vRMS = new TMultiGraph();
    auto uMean = new TMultiGraph();
    auto vMean = new TMultiGraph();
```

```
TGraphErrors* uMeanSensor = new TGraphErrors();
TGraphErrors* vMeanSensor = new TGraphErrors();
TGraphErrors* uRMSSensor = new TGraphErrors();
TGraphErrors* vRMSSensor = new TGraphErrors();

for (int l = 0; l < 100; l++){
  uMeanSensor->SetPoint(l,l,mean[j][0][l+number-100]);
  uMeanSensor->SetPointError(l,0,meanError[j][0][l+number-100]);
  vMeanSensor->SetPoint(l,l,mean[j][1][l+number-100]);
  vMeanSensor->SetPointError(l,0,meanError[j][1][l+number-100]);
  uRMSSensor->SetPoint(l,l,sigma[j][0][l+number-100]);
  uRMSSensor->SetPointError(l,0,sigmaError[j][0][l+number-100]);
  vRMSSensor->SetPoint(l,l,sigma[j][1][l+number-100]);
  vRMSSensor->SetPointError(l,0,sigmaError[j][1][l+number-100]);
}

uMeanSensor->SetMarkerColor(9);
uMeanSensor->SetMarkerStyle(20);
vMeanSensor->SetMarkerColor(2);
vMeanSensor->SetMarkerStyle(20);
uRMSSensor->SetMarkerColor(9);
uRMSSensor->SetMarkerStyle(20);
vRMSSensor->SetMarkerColor(2);
vRMSSensor->SetMarkerStyle(20);

uRMS->Add(uRMSSensor, "p");
vRMS->Add(vRMSSensor, "p");
uMean->Add(uMeanSensor, "p");
vMean->Add(vMeanSensor, "p");

stringstream canvas_word;
string canvas_number;
canvas_word << j;
canvas_word >> canvas_number;
string canvas_title = "sensors[" + canvas_number + "]";

sensors[j]= new TCanvas(canvas_title.c_str(),"",1800, 800);
TPad * first = new TPad("first", "first", 0.2, 0.53, 0.99, 0.96);
first->SetTopMargin(0.01);
first->SetBottomMargin(0.01);
first->SetRightMargin(0.01);
first->SetLeftMargin(0.04);
first->Draw();
TPad * second = new TPad("second", "second", 0.00, 0.48, 0.2, 0.96);
second->SetTopMargin(0.01);
second->SetBottomMargin(0.11);
second->SetRightMargin(0.05);
second->SetLeftMargin(0.14);
second->Draw();
TPad * third = new TPad("third", "third", 0.2, 0.045, 0.99, 0.475);
third->SetTopMargin(0.01);
third->SetBottomMargin(0.01);
third->SetRightMargin(0.01);
third->SetLeftMargin(0.04);
third->Draw();
TPad * fourth = new TPad("fourth", "fourth", 0.00, 0.0, 0.2, 0.48);
fourth->SetTopMargin(0.02);
fourth->SetBottomMargin(0.10);
fourth->SetRightMargin(0.05);
fourth->SetLeftMargin(0.14);
fourth->Draw();

string titleText = "Parameters of residual histograms as function of run number for sensor " + label_plot[j];
TText *title = new TText(0.5,0.98,titleText.c_str());
title->SetTextAlign(22);
title->SetTextColor(kBlack);
title->SetTextFont(23);
title->SetTextSize(25);
title->SetTextAngle(0);
title->Draw();

TText *xtitle = new TText(0.21,0.5,"Run");
xtitle->SetTextAlign(22);
xtitle->SetTextColor(kBlack);
xtitle->SetTextFont(43);
xtitle->SetTextSize(18);
xtitle->SetTextAngle(0);
xtitle->Draw();

TText *xtext[100];
for (int i = 0; i < 100; i++){
  xtext[i] = new TText(0.2359+0.007485*i, 0.5, runs[i+number-100].c_str());
  xtext[i]->SetTextAlign(22);
  xtext[i]->SetTextColor(kBlack);
  xtext[i]->SetTextFont(43);
  xtext[i]->SetTextSize(13);
  xtext[i]->SetTextAngle(90);
  xtext[i]->Draw();
}

first->cd();
uMean->Draw("A");
uMean->GetXaxis()->SetLabelSize(0);
uMean->GetXaxis()->SetTickLength(0);
```

```
uMean->GetYaxis()->SetLabelSize(0.05);
uMean->GetYaxis()->SetTitle("Mean of residual in U [#mum]");
uMean->GetYaxis()->CenterTitle();
uMean->GetYaxis()->SetTitleSize(0.06);
uMean->GetYaxis()->SetTitleOffset(0.30);
uMean->GetXaxis()->SetRangeUser(0.0,(float)(100-1));
uMean->SetMaximum(uMean->GetHistogram()->GetMinimum()+
          (uMean->GetHistogram()->GetMaximum()-uMean->GetHistogram()->GetMinimum())*1.1);
BELLE2Label(0.425, 0.90, "", 1);
myText(0.47, 0.90, 1, "PRELIMINARY");
myText(0.55, 0.90, 1, "2018");
myText(0.58, 0.90, 1, "#int Ldt #approx 146 pb^{-1}");

second->cd();
alignedResult[j][0]->Draw();
alignedResult[j][0]->SetTitle("");
alignedResult[j][0]->GetYaxis()->SetLabelSize(0.04);
alignedResult[j][0]->GetXaxis()->SetLabelSize(0.04);
alignedResult[j][0]->GetYaxis()->SetTitleSize(0.05);
alignedResult[j][0]->GetXaxis()->SetTitleSize(0.05);
alignedResult[j][0]->GetYaxis()->SetTitleOffset(1.4);
alignedResult[j][0]->GetYaxis()->SetTitle("Counts");
alignedResult[j][0]->GetYaxis()->CenterTitle();
alignedResult[j][0]->GetXaxis()->SetTitle("Residual v U direction [#mum]");
alignedResult[j][0]->GetXaxis()->CenterTitle();
alignedResult[j][0]->SetFillStyle(3356);
alignedResult[j][0]->SetFillColor(9);
alignedResult[j][0]->SetLineColor(9);
gPad->Update();
TPaveStats * statisticsU = (TPaveStats*)alignedResult[j][0]->FindObject("stats");
statisticsU->SetOptStat(1110);
statisticsU->SetTextColor(9);
statisticsU->SetTextSize(0.045);
statisticsU->SetBorderSize(0);
statisticsU->SetX1NDC(0.60);
statisticsU->SetX2NDC(0.92);
statisticsU->SetY1NDC(0.80);
statisticsU->SetY2NDC(0.97);
gPad->Modified();
statisticsU->Draw();
BELLE2Label(0.17, 0.91, "", 1);
myText(0.17, 0.86, 1, "PRELIMINARY");
myText(0.17, 0.81, 1, "2018");
myText(0.16, 0.72, 1, "#int Ldt #approx 1.4 pb^{-1}");

third->cd();
vMean->Draw("A");
vMean->GetXaxis()->SetLabelSize(0);
vMean->GetXaxis()->SetTickLength(0);
vMean->GetYaxis()->SetLabelSize(0.05);
vMean->GetYaxis()->SetTitle("Mean of residual in V [#mum]");
vMean->GetYaxis()->CenterTitle();
vMean->GetYaxis()->SetTitleSize(0.06);
vMean->GetYaxis()->SetTitleOffset(0.30);
vMean->GetXaxis()->SetRangeUser(0.0,(float)(100-1));
vMean->SetMaximum(vMean->GetHistogram()->GetMinimum()+
          (vMean->GetHistogram()->GetMaximum()-vMean->GetHistogram()->GetMinimum())*1.1);
BELLE2Label(0.425, 0.90, "", 1);
myText(0.47, 0.90, 1, "PRELIMINARY");
myText(0.55, 0.90, 1, "2018");
myText(0.58, 0.90, 1, "#int Ldt #approx 146 pb^{-1}");

fourth->cd();
alignedResult[j][1]->Draw();
alignedResult[j][1]->SetTitle("");
alignedResult[j][1]->GetYaxis()->SetLabelSize(0.04);
alignedResult[j][1]->GetXaxis()->SetLabelSize(0.04);
alignedResult[j][1]->GetYaxis()->SetTitleSize(0.05);
alignedResult[j][1]->GetXaxis()->SetTitleSize(0.05);
alignedResult[j][1]->GetYaxis()->SetTitleOffset(1.4);
alignedResult[j][1]->GetYaxis()->SetTitle("Counts");
alignedResult[j][1]->GetYaxis()->CenterTitle();
alignedResult[j][1]->GetXaxis()->SetTitle("Residual in V direction [#mum]");
alignedResult[j][1]->GetXaxis()->CenterTitle();
alignedResult[j][1]->SetFillStyle(3365);
alignedResult[j][1]->SetFillColor(2);
alignedResult[j][1]->SetLineColor(2);
gPad->Update();
TPaveStats * statisticsV = (TPaveStats*)alignedResult[j][1]->FindObject("stats");
statisticsV->SetOptStat(1110);
statisticsV->SetTextColor(2);
statisticsV->SetTextSize(0.045);
statisticsV->SetBorderSize(0);
statisticsV->SetX1NDC(0.60);
statisticsV->SetX2NDC(0.92);
statisticsV->SetY1NDC(0.78);
statisticsV->SetY2NDC(0.95);
gPad->Modified();
statisticsV->Draw();
BELLE2Label(0.17, 0.90, "", 1);
myText(0.17, 0.85, 1, "PRELIMINARY");
myText(0.17, 0.80, 1, "2018");
myText(0.16, 0.72, 1, "#int Ldt #approx 1.4 pb^{-1}");
```

```
        string histogram_title = "sensor" + label_output[j];
        histogram_title = "pics/" + histogram_title + ".png";
        sensors[j]->SaveAs(histogram_title.c_str());
      }
    }
}
```