```
#include "fstream"
#include "TH1D.h"
#include "TVirtualfft.h"
#include "TF1.h"
#include "TCanvas.h"
#include "TMath.h"

// void Alignment( string filefolder = "170211/", string filebase = "print_011_", int fCu
t = 100, int fCut_scan = 100){
void Alignment( string filebase = "/Users/YASUDA/Data/Alignment/170830/print_023", int fC
ut = 100, int fCut_scan = 100){
  double RangeMin = -0.04000;
  double RangeMax =  0.04000;


  ////////////////////////////////
  // reading csv file
  ////////////////////////////////
  // const int Nwave = 3;
  const int Nwave = 2; // for calibration data not to include CH3 ( ONOSOKKI monitor for
measurement path )
  stringstream filename;


  std::vector<double> t[ Nwave];
  std::vector<double> v[ Nwave];

  for ( int iwave = 0 ; iwave < Nwave ; iwave++ ){
    filename.str("");
    // filename << "/Users/YASUDA/Data/Alignment/" << filefolder << filebase.c_str() << i
wave + 1 << ".csv";
    filename << filebase.c_str() << "_" << iwave + 1 << ".dat";

    cout << "opening ... " << filename.str().c_str();
    ifstream ifs(filename.str().c_str() , std::ios::in );
    if(!ifs){
      cout<<" ... error, file not found" << ":" << filename.str().c_str() <<  endl;
      return;
    }

    string str;
    int lineno = 0;
    int iword = 0;
    const int N = 10;
    string atoken[ N ];

    while(getline(ifs,str)){
      string token;
      istringstream stream(str);

      iword = 0;
      while(getline(stream,token,' ')){
          // cout << lineno << " " << iword << " " << token << endl;
          atoken[ iword ] = token;
          iword++;
      }
      //    cout << endl;

      double time;
      double volt;

      switch (lineno) {
          /* SAVE BY ASCII*/
      // case 0 :
      //   break;
      // case 1 :
      //   break;
      default :
```

```
      time = (double) atof( atoken[ 0 ].c_str() ) ;
      volt = (double) atof( atoken[ 1 ].c_str() ) ;

      if(RangeMin < time && time < RangeMax){
        t[ iwave ].push_back( time );
        v[ iwave ].push_back( volt );
      }
      break;
    }

    lineno++;
  }

  cout << ", read " << lineno << " lines." << endl;

  ifs.close();
} // iwave

cout << "number of point is " << t[ 0 ].size() << endl;

////////////////////////////////////
// define root file/canvas/histogram
////////////////////////////////////

stringstream rootfilename;
rootfilename.str("");
rootfilename << filebase.c_str() << ".root";

TFile * tfile = new TFile( rootfilename.str().c_str() ,"recreate");

TCanvas *c1 = new TCanvas("c1","c1");
TCanvas *c2 = new TCanvas("c2","c2");
TCanvas *c3 = new TCanvas("c3","c3");
TCanvas *c4 = new TCanvas("c4","c4");
TCanvas *c5 = new TCanvas("c5","c5");
TCanvas *c6 = new TCanvas("c6","c6");
TCanvas *c7 = new TCanvas("c7","c7");
TCanvas *c8 = new TCanvas("c8","c8");
TCanvas *c9 = new TCanvas("c9","c9");
TCanvas *c10 = new TCanvas("c10","c10");
TCanvas *c11 = new TCanvas("c11","c11");
TCanvas *c12 = new TCanvas("c12","c12");

const int n = t[ 0 ].size();
double Fs = 100.000;

TH1D *hraw_comb = new TH1D("hraw_comb" ,"Raw Waveform Comb",n, RangeMin,RangeMax);
hraw_comb->SetXTitle("Second [s]");
hraw_comb->SetYTitle("Voltage [V]");
hraw_comb->SetStats(0);

TH1D *hraw2_comb= new TH1D("hraw2_comb","RawSquared Comb",n,RangeMin,RangeMax);
hraw2_comb->SetXTitle("Second[s]");
hraw2_comb->SetLineColor(2);
hraw2_comb->SetStats(0);

TH1D *hraw_scan = new TH1D("hraw_scan" ,"Raw Waveform Scan",n, RangeMin,RangeMax);
hraw_scan->SetXTitle("Second [s]");
hraw_scan->SetYTitle("Voltage [V]");
hraw_scan->SetMarkerStyle(20);
hraw_scan->SetMarkerSize(0.1);
hraw_scan->SetStats(0);

TH1D *hraw_tgt  = new TH1D("hraw_tgt"  ,"Raw Waveform Target",n, RangeMin,RangeMax);
hraw_tgt->SetXTitle("Second [s]");
hraw_tgt->SetYTitle("Voltage [V]");

TH1 *hm    = new TH1D("hm","hm",n,0.,Fs);
```

```
  //  hm->SetName("MAGnoCut");
  hm->SetTitle("Magnitude of the 1st transform");
  hm->SetLineWidth(2);
  hm->SetXTitle("Frequency [kHz]");
  hm->SetYTitle("Magnitude");

  TH1 *hmcut = new TH1D("hmcut","hmcut",n,0.,Fs);
  //  hmcut->SetName("MAGCut");
  hmcut->SetLineColor(2);


  TH1 *hbcut = 0;

  TH1 *hbcut2= new TH1D("hbcut2","hbcut2",n,RangeMin,RangeMax);
  hbcut2->SetTitle("Fringe after Low Path Filter");
  hbcut2->SetXTitle("Time[sec]");
  hbcut2->SetYTitle("Rescaled Voltage [V]");
  hbcut2->GetYaxis()->SetRangeUser(0.,8.);
  hbcut2->SetStats(0);
  // hist->GetXaxis()->SetRangeUser(first,last);



  TH1D *h_diff   = new TH1D("h_diff",  "Diff",n,RangeMin,RangeMax);
  TH1D *h_diffsw = new TH1D("h_diffsw","Diffsw",n,RangeMin,RangeMax);
  h_diffsw->SetLineColor(2);

  TH1D *hnoise_tgt  = new TH1D("hnoise_tgt","noise target",150,-1.,11.);
  hnoise_tgt->SetTitle("Target Stage");
  hnoise_tgt->SetXTitle("Voltage [V]");
  hnoise_tgt->SetYTitle("Event");

  TH1D *hnoise_scan = new TH1D("hnoise_scan","noise scan",9,-0.09,0.09);
  hnoise_scan->SetTitle("Scanning Stage");
  hnoise_scan->SetXTitle("Voltage [V]");
  hnoise_scan->SetYTitle("Event");


  ////////////////////////////////////
  // filling histograms
  ////////////////////////////////////

  const int icomb = 0;
  const int iscan = 1;
  const int itgt  = 2;

  for ( int i = 0 ; i < t[ icomb].size() ; i++ ) {
    int binn = hraw_comb->FindBin( t[ icomb ][ i ] );
    if(fabs(hraw_comb->GetBinLowEdge(binn)- t[ icomb ][ i ])>0.00002
       && binn!=0
       && binn!=n+1
       ){
      binn += 1;
    }
    hraw_comb->SetBinContent(binn, v[ icomb ][ i ] );
    hraw_scan->SetBinContent(binn, v[ iscan ][ i ] );
    hraw_tgt->SetBinContent(binn, v[ itgt ][ i ] );

    // cout << binn << " " << hraw_comb->GetBinContent(binn) << " " << hraw_scan->GetBinC
ontent(binn) << " " << hraw_tgt->GetBinContent(binn) << endl;

    hraw2_comb->SetBinContent(binn, pow( v[ icomb ][ i ], 2 ) );

    hnoise_tgt->Fill( (double) v[ itgt ][ i ] );
    if(binn<1000){
      hnoise_scan->Fill( (double) v[ iscan ][ i ]);
    }
  }
```

```
////////////////////////////////////
// FFT
////////////////////////////////////

double re,im;
double *re_full = new Double_t[n];
double *im_full = new Double_t[n];
double *re_full_cut = new Double_t[n];
double *im_full_cut = new Double_t[n];

hm = hraw2_comb->FFT(hm,"MAG");

TVirtualFFT *fft = TVirtualFFT::GetCurrentTransform();


fft->GetPointComplex(0,re,im);
printf("1st transform: DC component: %f\n", re);
fft->GetPointComplex(n/2+1, re, im);
printf("1st transform: Nyquist harmonic: %f\n", re);

fft->GetPointsComplex(re_full, im_full);

TVirtualFFT *fftcut = TVirtualFFT::GetCurrentTransform();

for(int i=0;i < n ;i++){
  re_full_cut[i] = 0;
  im_full_cut[i] = 0;
}

for(int i=0;i < fCut ;i++){
  fftcut->GetPointComplex(  i,re_full_cut[  i], im_full_cut[  i]);
  fftcut->GetPointComplex(n-i,re_full_cut[n-i], im_full_cut[n-i]);
}

hmcut = hraw2_comb->FFT(hmcut,"MAG");

for(int i=0;i<n;i++){
  hmcut->SetBinContent(i+1,sqrt(pow(re_full_cut[i],2)+pow(im_full_cut[i],2)));
}

TVirtualFFT::SetTransform(0);

int m = n;
TVirtualFFT *fftBackCut = TVirtualFFT::FFT(1,&m,"C2R M K");
fftBackCut->SetPointsComplex(re_full_cut,im_full_cut);
fftBackCut->Transform();

hbcut = TH1::TransformHisto(fftBackCut,hbcut,"Re");
hbcut->SetName("hbcut");
hbcut->SetTitle("Frequency Cut");
hbcut->SetXTitle("# of Bin");

cout << "m="<< m << endl;
for(int i=0; i < m ; i++){
  hbcut2->SetBinContent(i,hbcut->GetBinContent(i)/m);
}

////////////////////////////////////
// FFT OF SCANNING STAGE
////////////////////////////////////

TH1 *hbcut_scan = new TH1D("hbcut_scan","hbcut_scan",n,RangeMin,RangeMax);
hbcut_scan->GetYaxis()->SetRangeUser(0.48,0.53);
hbcut_scan->GetXaxis()->SetRangeUser(-0.001,0.001);
// TH1 *hbcut_scan = 0;
hbcut_scan->SetTitle("Scanning after Low Path Filter");
hbcut_scan->SetXTitle("Time[sec]");
```

```
  hbcut_scan->SetYTitle("Rescaled Voltage [V]");
  hbcut_scan->SetStats(0);

  TH1 *hm_scan = new TH1D("hm_scan","hm_scan",n,0.,Fs);
  hm_scan->SetTitle("Magnitude of the 1st transform of scanning");
  hm->SetLineWidth(2);
  hm->SetXTitle("Frequency [kHz]");
  hm->SetYTitle("Magnitude");

  TH1 *hmcut_scan = new TH1D("hmcut_scan","hmcut_scan",n,0.,Fs);
  hmcut->SetLineColor(2);

  double re_scan,im_scan;
  double *re_full_scan = new Double_t[n];
  double *im_full_scan = new Double_t[n];
  double *re_full_cut_scan = new Double_t[n];
  double *im_full_cut_scan = new Double_t[n];


  hm_scan = hraw_scan->FFT(hm_scan,"MAG");

  TVirtualFFT *fft_scan = TVirtualFFT::GetCurrentTransform();


  fft_scan->GetPointComplex(0,re_scan,im_scan);
  printf("1st transform of scan: DC component: %f\n", re_scan);
  fft_scan->GetPointComplex(n/2+1, re_scan, im_scan);
  printf("1st transform of scan: Nyquist harmonic: %f\n", re_scan);

  fft_scan->GetPointsComplex(re_full, im_full);
  fft_scan->GetPointsComplex(re_full_scan, im_full_scan);

  TVirtualFFT *fftcut_scan = TVirtualFFT::GetCurrentTransform();

  for(int i=0;i < n ;i++){
    re_full_cut_scan[i] = 0;
    im_full_cut_scan[i] = 0;
  }

  for(int i=0;i < fCut_scan ;i++){
    fftcut_scan->GetPointComplex(  i,re_full_cut_scan[  i], im_full_cut_scan[  i]);
    fftcut_scan->GetPointComplex(n-i,re_full_cut_scan[n-i], im_full_cut_scan[n-i]);
  }

  hmcut_scan = hraw_scan->FFT(hmcut_scan,"MAG");

  for(int i=0;i<n;i++){
    hmcut_scan->SetBinContent(i+1,sqrt(pow(re_full_cut_scan[i],2)+pow(im_full_cut_scan[i]
,2)));
  }

  TVirtualFFT::SetTransform(0);

  int m_scan = n;
  TVirtualFFT *fftBackCut_scan = TVirtualFFT::FFT(1,&m,"C2R M K");
  fftBackCut_scan->SetPointsComplex(re_full_cut_scan,im_full_cut_scan);
  fftBackCut_scan->Transform();

  hbcut_scan = TH1::TransformHisto(fftBackCut_scan,hbcut_scan,"Re");
  hbcut_scan->SetName("hbcut_scan");
  hbcut_scan->SetTitle("Frequency Cut");
  hbcut_scan->SetXTitle("Time[sec]");

  cout << "m_scan="<< m_scan << endl;
  for(int i=0; i < m_scan ; i++){
    hbcut_scan->SetBinContent(i,hbcut_scan->GetBinContent(i)/m);
  }
```

```
  ////////////////////////////////////
  // Average of Scanning Stage
  ////////////////////////////////////

  int div = 50;
  TH1D *hs = new TH1D("hs","Average of Scanning stage",n/div,RangeMin,RangeMax);
  TH1D *hs_full = new TH1D("hs","Full bin _ Average of Scanning stage",n,RangeMin,RangeMa
x);
  hs->SetStats(0);
  double sum = 0;
  double ave = 0;
  for(int i=0; i < n/div ; i++){
      for(int isum=0; isum<div ; isum++){
          sum += hraw_scan->GetBinContent(i*div+isum);
          // cout << isum << " " << hraw_scan->GetBinContent(i*div+isum) << " " << sum <<
 endl;
      }
      ave = sum/div;
      hs->SetBinContent(i,ave);
      // cout << i*div << " " << sum << " " << ave << endl;
      sum = 0;
      ave = 0;
  }

  for(int i = 0; i < n/div ; i++ ) {
      for(int j = 0; j<div ; j++){
          hs_full->SetBinContent(i*div + j,(hs->GetBinContent(i+1)-hs->GetBinContent(i))/
div*j + hs->GetBinContent(i));
          // cout << i*div + j << " " << hs_full->GetBinContent(i*div+j) << endl;
      }
  }

  ////////////////////////////////////
  // peak finding
  ////////////////////////////////////

  double diff = 0.;
  int    sw   = 0;

  const int npeak = 100;
  double t_peak[ npeak ];
  double v_peak[ npeak ];
  double v_peak_scan[ npeak ];
  double v_peak_ave[ npeak ];
  double ono_scan = 2000.0 ; /* um/voltage */
  for ( int i = 0 ; i < npeak ; i++ ) {
    t_peak[ i ] = -1.;
    v_peak[ i ] = -1.;
    v_peak_scan[ i ] = -1.;
    v_peak_ave[ i ] = -1.;
  }
  int    ipeak = 0;
  double diff_sw = 300.0;
  for(int i=0;i<n;i++){
    diff= (hbcut->GetBinContent(i+2)) - ((hbcut->GetBinContent(i+1)));
    h_diff->SetBinContent(i+1,diff);

    if( sw == 1 || diff > diff_sw ) {
      sw = 1;
    }

    if ( sw == 1 && diff < 0. ) {
      sw = 0;

      if ( ipeak < npeak ) {
        t_peak[ ipeak ] = (double) hbcut2->GetXaxis()->GetBinCenter(i+1);
        v_peak[ ipeak ] = (double) hraw_scan->GetBinContent(i+1);
```

```
        v_peak_scan[ ipeak ] = (double) hbcut_scan->GetBinContent(i+1);
        v_peak_ave[ ipeak ] = (double) hs_full->GetBinContent(i+1);
        ipeak ++;
      }
    }

    h_diffsw->SetBinContent(i+1, (double) 100 * sw);

  }

  if ( ipeak < 0 ) {
    cout << "error bin_peak < 0 " << endl;
  }

  cout << "ipeak, time[sec], voltage[V], ono[um],LP voltage[V], ono_scan[um], ave voltage
[V], ave ono_scan[um]  " << endl;
  for ( int i = 0 ; i < ipeak ; i++ ) {
    cout << i
         << " " << t_peak[ i ]
         << " " << v_peak[ i ]
         << " " << v_peak[ i ] * ono_scan
         << " " << v_peak_scan[ i ]
         << " " << v_peak_scan[ i ] * ono_scan
         << " " << v_peak_ave[ i ]
         << " " << v_peak_ave[ i ] * ono_scan
         << endl;
  }

  if ( ipeak == 2 ) {
    double tdiff = t_peak[ 1 ] - t_peak[ 0 ];
    double vdiff = v_peak[ 1 ] - v_peak[ 0 ];
    double vdiff_scan = v_peak_scan[ 1 ] - v_peak_scan[ 0 ];
    double vdiff_ave = v_peak_ave[ 1 ] - v_peak_ave[ 0 ];
    double disp = v_peak[ 1 ] * ono_scan - v_peak[ 0 ] * ono_scan;
    double disp_scan = v_peak_scan[ 1 ] * ono_scan - v_peak_scan[ 0 ] * ono_scan;
    double disp_ave = v_peak_ave[ 1 ] * ono_scan - v_peak_ave[ 0 ] * ono_scan;
    cout << "time difference [s] = " << tdiff << endl;
    cout << "voltage difference [V] = " << vdiff << endl;
    cout << "voltage difference after low path of scanning [V] = " << vdiff_scan << endl;
    cout << "Ave : voltage difference [V] = " << vdiff_ave << endl;
    cout << "displacement [um] = " << disp << endl;
    cout << "displacement after low path of scanning [um] = " << disp_scan << endl;
    cout << "Ave : displacement [um] = " << disp_ave << endl;
  }

  hnoise_tgt->Fit("gaus");
  //  hnoise_scan->Fit("gaus");

  cout << "Mean of tgt gaussian = " << hnoise_tgt->GetFunction("gaus")->GetParameter(1) <
< endl;
  cout << "Position of tgt [um] = " << hnoise_tgt->GetFunction("gaus")->GetParameter(1) *
 5.00 << endl;




  // std::vector<double> sum;
  // std::vector<double> ave;
  // for(int i=0; i<n/div ; i++){
  //     for(int j=0; j<div; j++){
  //         sum.push_back(v[ iscan ][i*div + j]);
  //     }
  //     ave.push_back(sum[i]/div);
  //     cout << i*10 << " " << sum[i] << " " << ave[i] << endl;

  // }

  ////////////////////////////////
```

```
  // Draw results
  ////////////////////////////////////


  // for(int binn = 0; binn < t[icomb].size() ; binn++){
  //     cout << binn << " " << hraw_comb->GetBinContent(binn) << " " << hraw_scan->GetBi
nContent(binn) << " " << hraw_tgt->GetBinContent(binn) << endl;
  // }


  gStyle->SetOptFit();
  c1->Draw();
  c1->Divide(2,2);
  c1->cd(1);
  hraw_comb->Draw();
  c1->cd(2);
  hraw2_comb->Draw();
  c1->cd(3);
  hraw_scan->Draw();
  c1->cd(4);
  hraw_tgt->Draw();

  // c1->Update();
  // c1->Modified();

  c2->cd();
  hm->Draw();
  hmcut->Draw("SAME");

  c3->cd();
  // hbcut->Draw();
  hbcut2->Draw();
  h_diffsw->Draw("SAME");
  hbcut->Draw("SAME");


  c4->Divide(1,2);
  c4->cd(1);
  hnoise_tgt->Draw();
  c4->cd(2);
  hnoise_scan->Draw();

  c5->cd();
  h_diff->Draw();
  h_diffsw->Draw("SAME");


  c6->cd();
  hbcut->Draw();
  h_diffsw->Draw("SAME");

  c7->cd();
  hbcut2->Draw("SAME");
  // h_diffsw->Draw("SAME");
  hraw_scan->Draw("SAME");

  c8->cd();
  hbcut_scan->Draw();

  c9->cd();
  hm_scan->Draw();
  hmcut_scan->Draw("SAME");

  c10->cd();
  hbcut2->Draw();
  h_diffsw->Draw("SAME");
  hbcut_scan->Draw("SAME");
```

```
  // canvas->DrawFrame(xmin,ymin,xmax,ymax);
  // hist->Draw("same");
  // canvas->RedrawAxis();

  /* Wrong function */
  // TF1 *hb_scan_fit = hbcut_scan->GetFunction("pol1");
  // TH1D *hres = new TH1D("hres","Fit Residuals", n, RangeMin, RangeMax);

  // for(int ibin = 0; ibin < n ; ibin++){
  //     double res = (hbcut_scan->GetBinContent(ibin) - hb_scan_fit->Eval(hbcut_scan->Ge
tBinCenter(ibin)));
  //     hres->SetBinContent(ibin, res);
  // }
  // c11->cd();
  // hres->Draw();

  c11->cd();
  hs->Draw();

  c12->cd();
  hs_full->Draw();

  tfile->Write();
  // tfile->Close();


}
```