Kalibracja kamer dla eksperymentu „Pi of the Sky".

Tomasz Krajewski

27 czerwca 2011

Spis treści

1 Kalibracja połówek. 3

2 Kalibracja wzmocnienia. 4

3 Wyniki analizy FLATów. 5
3.1 Zbieranie danych. 5
3.2 Uzyskiwanie uśrednionych FLATów. 7
3.2.1 Klasa zdjęć *Image*. 7
3.2.2 Funkcja *flatdiag*. 11
3.2.3 Funkcja *flatstat*. 14
3.2.4 Funkcja *flatana*. 18
3.3 Prezentacja wyników. 19

4 Porównanie kamer. 19
4.1 Zmienność związana z różnym oświetleniem. 19
4.2 Zróżnicowanie kamer. 20
4.3 Metodyka. 21

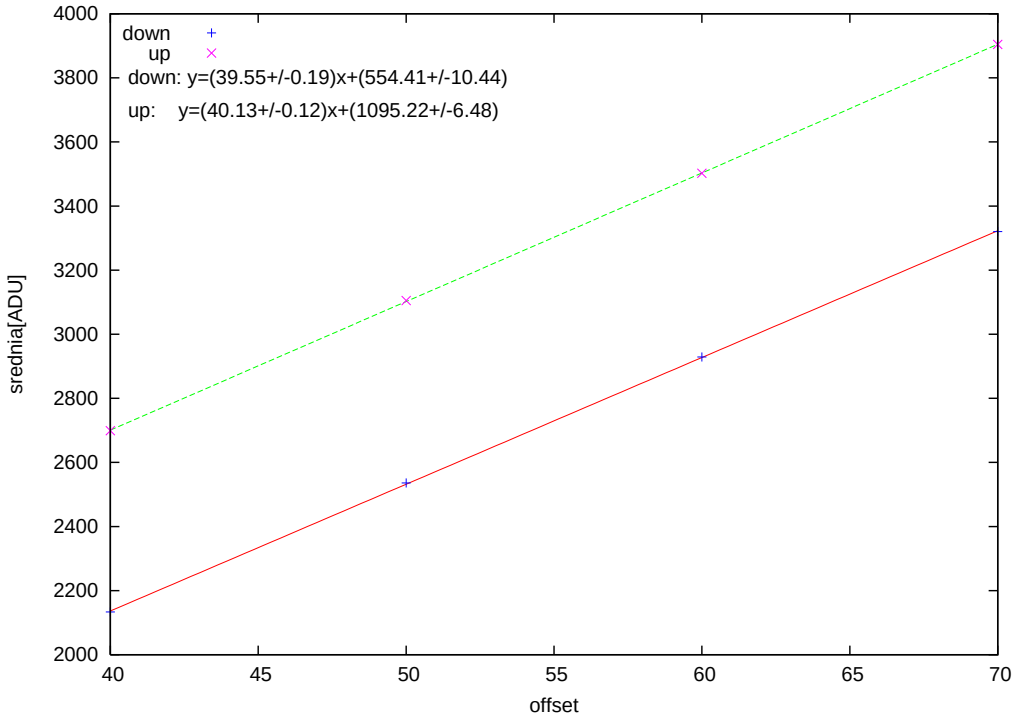
5 Podsumowanie. 25

Streszczenie

Artykuł zawiera opis metod wykorzystywanych przy kalibracji kamer dla eksperymentu „Pi of the Sky". Pokrótce opisano kalibracje połówek chipu CCD (offset oraz wzmocnienie). Główny nacisk położono na opisanie metod mających na celu poprawe uzyskiwanych obrazów przez uwzględnienie charakterystyki chipu oraz układu optycznego.

1 Kalibracja połówek.

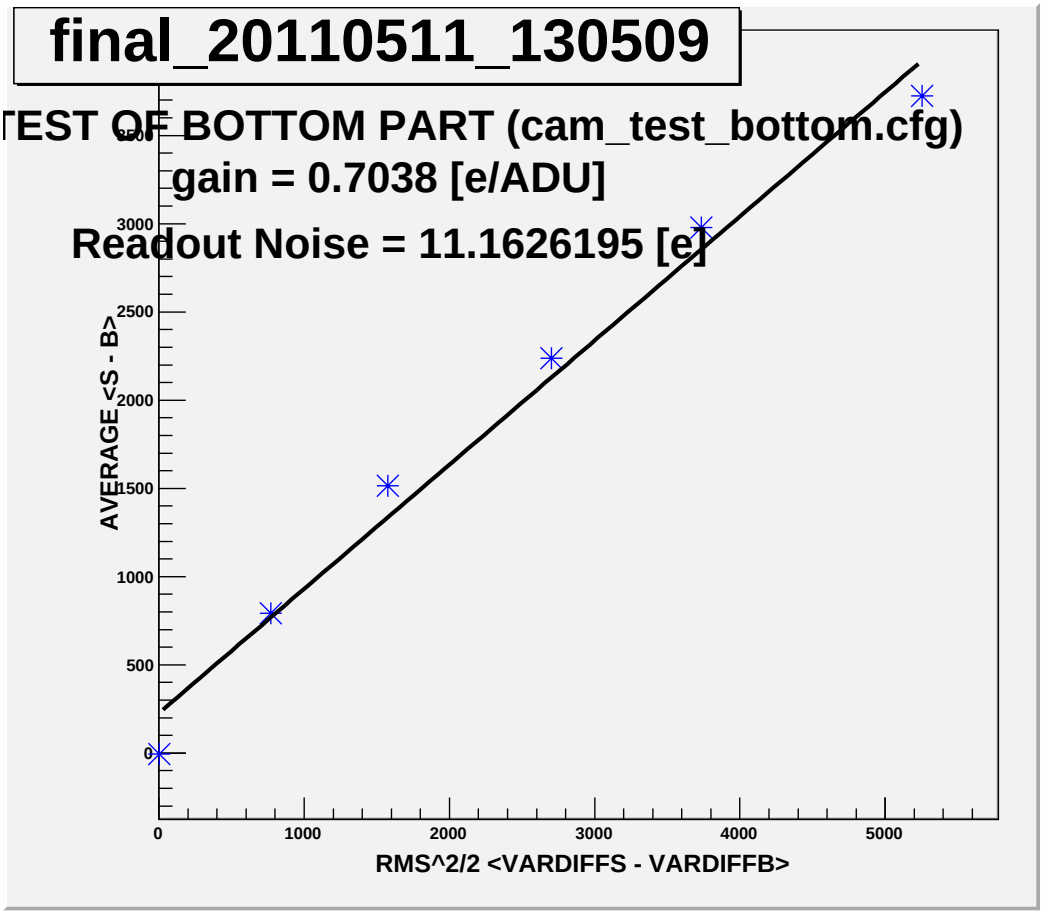
Kalibracja offsetu dla dwóch połówek chipu miała na celu zredukowanie efektu związanego z różnym poziomem odniesienia dla elementów dokonujących odczytu oraz przesyłu danych dla różnych fragmentów chipu. Ten element kalibracji został wykonany za pomocą skryptu, przygotowanego wcześniej. Wynikiem działania skryptu była wartość offsetu dla dolnej połówki chipu, dająca wartości najbardziej zbliżone do górnej połówki z ustawianiem wartości 50. Następnie uwzględnianio poprawki w pliku konfiguracyjnym kalibrowanej kamery. Skrypt analizował zdjęcia, robione przy zamkniętej migawce z czasem ekspozycji 10 sekund, na których rejestrowane były szumy termiczne kamery. Dodatkowo uzyskane zdjęcia dawały obraz szumów termicznych kamery. Wadą metody okazała się dość mała możliwośc regulacji – minimalny krok równy 1 w ustawieniach offestu, nie daje możliwości dokładnego dopasowania, czyli różnic na poziomie szumów termicznych. Wykres 1 prezentuje przykładowe wyniki skryptu kalibrującego offset połówek chipu kamery. Znaki + na wykresie obrazują punkty pomiarowe dla górnej połówki chipu, natomiast ciągła, czerwona linia obrazuje prostą dopasowaną do tych punktów. Znaki $\times$ na wykresie obrazują punkty pomiarowe dla dolnej połówki chipu, natomiast przerywana, zielona linia obrazuje prostą dopasowaną do tych punktów. Dodatkowo przedstawiono w lewym górnym rogu parametery dopasowania.



Rysunek 1: Znaki + obrazują punkty pomiarowe dla górnej połówki chipu, natomiast ciągła, czerwona linia obrazuje prostą dopasowaną do tych punktów. Znaki × obrazują punkty pomiarowe dla dolnej połówki chipu, natomiast przerywana, zielona linia obrazuje prostą dopasowaną do tych punktów.

2 Kalibracja wzmocnienia.

Kalibracja wzmocnienia, podobnie jak kalibracja offsetu, była wykonywana za przez skrypt pracujący na klatkach ciemnych. Skrypt analizował oddzielnie każdą z cześći chipu. Wyniki takiej analizy przedstawiają wykresy 2, 3, odpowiednio dla dolnej i górnej połówki chipu kamery. Punkty pomiarowe na wykresach zostały oznaczone niebieskimi znakami $*$, natomiast czarna linia obrazuje prostą dopasowaną do tych punktów. Dodatkowo przedstawiono parametery dopasowania.



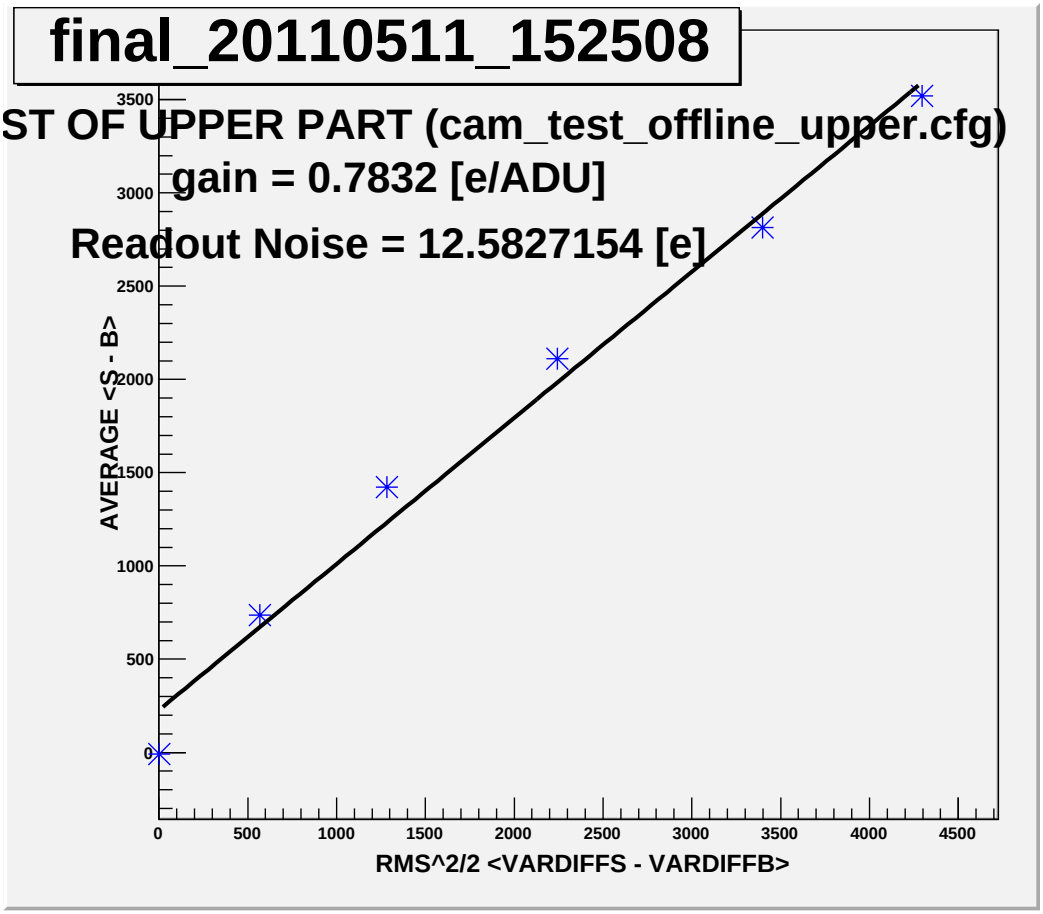
Rysunek 2: Wykres przedstawia wynik skryptu kalibrującego wzmocnienie dolnej połówki chipu. Punkty pomiarowe na wykresie zostały oznaczone niebieskimi znakami $*$, natomiast czarna linia obrazuje prostą dopasowaną do tych punktów.

Podczas kalibracji wzmocnienia, dokonano także analizy szumów termicznych kamer.

3 Wyniki analizy FLATów.

3.1 Zbieranie danych.

Głównym problemem artykułu jest poprawa jakości zdjęć wykonywanych przy pomocy kalibrowanych kamer. Zdjęcia robione kamerą CCD wyposażoną w obiektyw charakteryzują się nierównomierną jasnością w zależno-



Rysunek 3: Wykres przedstawia wynik skryptu kalibrującego wzmocnienie górnej połówki chipu. Punkty pomiarowe na wykresie zostały oznaczone niebieskimi znakami $*$, natomiast czarna linia obrazuje prostą dopasowaną do tych punktów.

ści od miejsca na chipie. Istnieją dwa główne powody powstawania takich niejednorodności. Po piewsze obrazowanie za pomocą układu optycznego daje największe natężenie światła na środku chipu. Po drugie wykorzystane chipy CCD nie są jednorodne, pomiędzy połówkami chipu przebiega linia bez elementów światłoczułych, a w nowszych chipach występują linie pikseli o zmniejszonej wydajności. W celu kompensacji wpływu układu optycznego jak i konstrukcji samego chipu, należało zbadać charakterystykę stosowanego układu. Główną ideą było otrzymanie zdjęcia przy równomiernym oświetleniu układu, które następnie mogłoby posłużyć do otworzenia obrazów obserwowanych z danych pochodzących z kamer. W tym celu układ który ma być wykorzystany do obserwacji był oświetlany jednorodym źródłem światła.

Zgodnie z zaleceniem producenta do tłumienia światła użyto kartek papieru, w ilości między 25 do 40 sztuk w zależności od kamery. Światło dochodzące bezpośrednio z lampy prowadziło do prześwietlenia zdjęć z czasem ekspozycji 10 sekund (przewidywanym jako docelowy w obserwacjach astronomicznych). Procedura zbierania danych rozpoczynała się od ustalenia odpowiedniej ilości kartek. W tym celu zbierano serie 5 zdjęć dal różnej ilości papieru, próbując doporowadzić do sygnału z kamery rzędu 17000 na środku chipu i 10000 bliżej brzegu. Takie postępowanie miało na celu badanie charakterystyki z zakresie liniowym układów elektronicznych odpowiedzialnych za zbieranie danych z chipu. Ponieważ użyte przetworniki miały rozdzielczość 64k bitów zakres 10000-17000 wydawał się bezpieczny. Następnie w obrębie odpowiedniego oświetlenia zebrano dwie seriie po 20 zdjęć dla dwóch różnych o 1 liczb kartek. Wszytkie notatki dotyczące zbierania danych były umieszczane w pliku *log*, który następnie służył do automatycznego odczytu danych o użytej liczbie kartek.

3.2 Uzyskiwanie uśrednionych FLATów.

Analizę otrzymanych zdjęć wykonano za pomocą dwóch funkcji w języku C++: *flatdiag* analizującej natężenie sygnału w zależności od liczby kartek, oraz *flatstat* prowadzącej do uzyskania uśrednionych zdjęć dla konkretnej liczby kartek, w środkowisku interpretatora *Astroroot.*, W celu przyspieszenia obliczeń w obu funkcjach, zamiast orginalnej **TH2D** środowiska *Root*, wykorzystano klasę *Image*. Obie funkcje zawarto w funkcji *flatana*, odczytującej dane z pliku *log* i wywołującej powyższe funkcje dla odpowiednich argumentów. Wyniki w posatci histogramów klasy **TH2D** są zapisywane do liku o rozszerzeniu *.root*.

3.2.1 Klasa zdjęć *Image*.

Poniżej przedstawiono kod źródłowy klasy *Image*. Trzeba zaznaczyć, że klasa ta ma tę przewage nad klasą **TH2D**, że jest mniej obciążająca obliczeniowo, co wynika z bardzo prostej struktury, pociągającej za sobą jednocześnie mniejszą funkcjonalność. Ważne jest aby przy analizie dużej liczby zdjęć korzystać z operatorów w formie skróconej ($+=$, $-=$, $*=$, $/=$), zamiast operatorów w formie długiej ($+$, $-$, $*$, $/$), ponieważ te drugie prowadzą do powstania nowego obiektu klasy.

```
using namespace std;
#include <iostream>

class Image {
public:
    Image (UInt_t x, UInt_t y);
    ~Image ();
```

```
    Double_t min ();
    Double_t max ();
    Double_t sizeof_x ();
    Double_t sizeof_y ();
    Double_t valueof (UInt_t x, UInt_t y);
    void scale(Double_t scale);
    void scale(Double_t scale, UInt_t min_x, UInt_t min_y,UInt_t max_x, UInt_t max_y);
    Double_t integrate();
    Double_t integrate(UInt_t min_x, UInt_t min_y,UInt_t max_x, UInt_t max_y);
    Double_t averageof();
    Double_t averageof(UInt_t min_x, UInt_t min_y,UInt_t max_x, UInt_t max_y);
    void sqrt();
    void fill (UInt_t x, UInt_t y, Double_t number);
    void fill (UInt_t x, UInt_t y, UShort_t number);
    TFDoubleImg* toDoubleImg();
private:
    UInt_t size_x,size_y;
    Double_t *value;
    friend Image& operator += (const Image& second);
    friend Image& operator -= (const Image& second);
    friend Image& operator *= (const Image& second);
    friend Image& operator /= (const Image& second);
    friend Image operator + (const Image& first, const Image& second);
    friend Image operator - (const Image& first, const Image& second);
    friend Image operator * (const Image& first, const Image& second);
    friend Image operator / (const Image& first, const Image& second);
};

Image::Image (UInt_t x, UInt_t y){
    size_x=x;
    size_y=y;
    value =  new Double_t[size_x*size_y+1];
    for (int i=0; i <= x*y+1 ; i++)
    {
        value[i]=0;
    }
}

Image::~Image(){
}

Double_t Image::sizeof_x (){
    return size_x;
}

Double_t Image::sizeof_y (){
    return size_y;
}

void Image::scale(Double_t scale){
    for (int i=0; i <= size_x*size_y ; i++)
    {
        value[i]*=scale;
    }
}

void Image::scale(Double_t scale, UInt_t min_x, UInt_t min_y,UInt_t max_x, UInt_t max_y){
    for (int ix=min_x; i <= max_x ; ix++)
    {
        for (int iy=min_y; i <= max_y ; iy++)
        {
        value[i]*=scale;
```

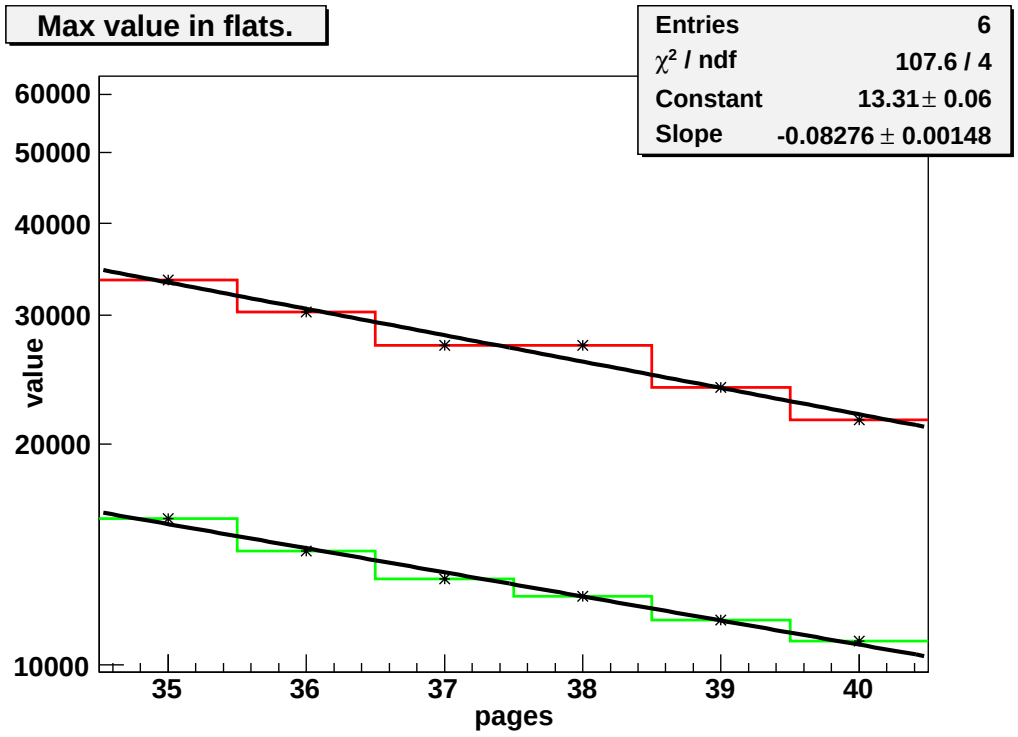
```
        }
    }
}

void Image::sqrt(){
    for (int i=0; i <= size_x*size_y ; i++)
    {
        value[i]=sqrt(value[i]);
    }
}

Double_t Image::integrate(UInt_t min_x, UInt_t min_y,UInt_t max_x, UInt_t max_y){
    Double_t integral=0;
    for (int ix=min_x; i <= max_x ; ix++)
    {
        for (int iy=min_y; i <= max_y ; iy++)
        {
        integral+=value[(iy-1)*size_x + ix];
        }
    }
    return integral;
}

Double_t Image::integrate(){
    Double_t integral=0;
    for (int i=0; i <= size_x*size_y ; i++)
    {
        integral+=value[i];
    }
    return integral;
}

Double_t Image::averageof(){
    return  ((this->integrate())/(size_x*size_y));
}

Double_t Image::averageof(UInt_t min_x, UInt_t min_y,UInt_t max_x, UInt_t max_y){
    return  ((this->integrate())/((max_x-min_x)*(max_y-min_y)));
}

void Image::fill(UInt_t x, UInt_t y, Double_t number){
    value[(y-1)*size_x + x]=number;
}

void Image::fill(UInt_t x, UInt_t y, UShort_t number){
    value[(y-1)*size_x + x]=((Double_t)number);
}

Double_t Image::valueof (UInt_t x, UInt_t y){
    return value[(y-1)*size_x + x];
}

Image& Image::operator += (const Image& second){
    for (int i=0; i <= size_x*size_y ; i++)
    {
        value[i]+=second.value[i];
    }
    return this;
}

Image& Image::operator -= (const Image& second){
    for (int i=0; i <= size_x*size_y ; i++)
```

```
    {
        value[i]-=second.value[i];
    }
    return this;
}

Image& Image::operator *= (const Image& second){
    for (int i=0; i <= size_x*size_y ; i++)
    {
        value[i]*=second.value[i];
    }
    return this;
}

Image& Image::operator /= (const Image& second){
    for (int i=0; i <= size_x*size_y ; i++)
    {
        value[i]/=second.value[i];
    }
    return this;
}

Image operator + (const Image& first, const Image& second){
    if (first.size_x!=second.size_x||first.size_y!=second.size_y){
        cout << Error << endl;
    }else{
        Image image(first.size_x,first.size_y);
        for (int i=0; i <= first.size_x*first.size_y ; i++)
        {
            image.value[i]=first.value[i] + second.value[i];
        }
        return image;
    }
}

Image operator - (const Image& first, const Image& second){
    if (first.size_x!=second.size_x||first.size_y!=second.size_y){
        cout << Error << endl;
    }else{
        Image image(first.size_x,first.size_y);
        for (int i=0; i <= first.size_x*first.size_y ; i++)
        {
            image.value[i]=first.value[i] - second.value[i];
        }
        return image;
    }
}

Image operator * (const Image& first, const Image& second){
    if (first.size_x!=second.size_x||first.size_y!=second.size_y){
        cout << Error << endl;
    }else{
        Image image(first.size_x,first.size_y);
        for (int i=0; i <= first.size_x*first.size_y ; i++)
        {
            image.value[i]=first.value[i] * second.value[i];
        }
        return image;
    }
}

Image operator / (const Image& first, const Image& second){
```

```
    if (first.size_x!=second.size_x||first.size_y!=second.size_y){
        cout << Error << endl;
    }else{
        Image image(first.size_x,first.size_y);
        for (int i=0; i <= first.size_x*first.size_y ; i++)
        {
            image.value[i]=first.value[i] / second.value[i];
        }
        return image;
    }
}

Double_t Image::min (){
    Double_t min=65000;
    for (int i=0; i <= size_x*size_y ; i++)
    {
        if (value[i]<min)
        {
            min = value[i];
        }
    }
    return min;
}

Double_t Image::max (){
    Double_t max=0;
    for (int i=0; i <= size_x*size_y; i++)
    {
        if (value[i]>max)
        {
            max = value[i];
        }
    }
    return max;
}

TFDoubleImg* Image::toDoubleImg(){
    TFDoubleImg * img = new TFDoubleImg ("name",size_x,2,size_y);
    for(iy=0;iy<size_y;iy++){
        for(ix=0;ix<size_x;ix++){
            img[0][iy][ix] = value[(iy-1)*(size_x) + ix];
        }
    }
    return img;
}
```

3.2.2 Funkcja *flatdiag*.

Funkcja *flatdiag* buduje histogram zawierający maksymalną, średnią oraz minimalną wartość dla uśrednionego zdjęcia w zależności od liczby kartek papieru użytych do tłumienia światła. Dodanych tych zostaje następnie dopasowany zanik wykładniczy. Wykres 4 prezentuje przykładowy wynik działania funkcji *flatdiag*. Na osi poziomej odłożono liczbe kartek, a na osi pionowej sygnał z kamery. Na wykresie przedstawiono zależność dla największej wartości na klatce uśrednionej (punkty przez które przechodzi czerwona linia) oraz wartości średniej (punkty przez które przechodzi zielona linia).



Rysunek 4: Na wykresie przesdtawiono zależność od liczby kartek użytych do tłumienia światła, dla największej wartości na klatce uśrednionej (czerwona linia), wartości średniej (czarna linia), oraz najmniejszej wartości na klatce (zielona linia).

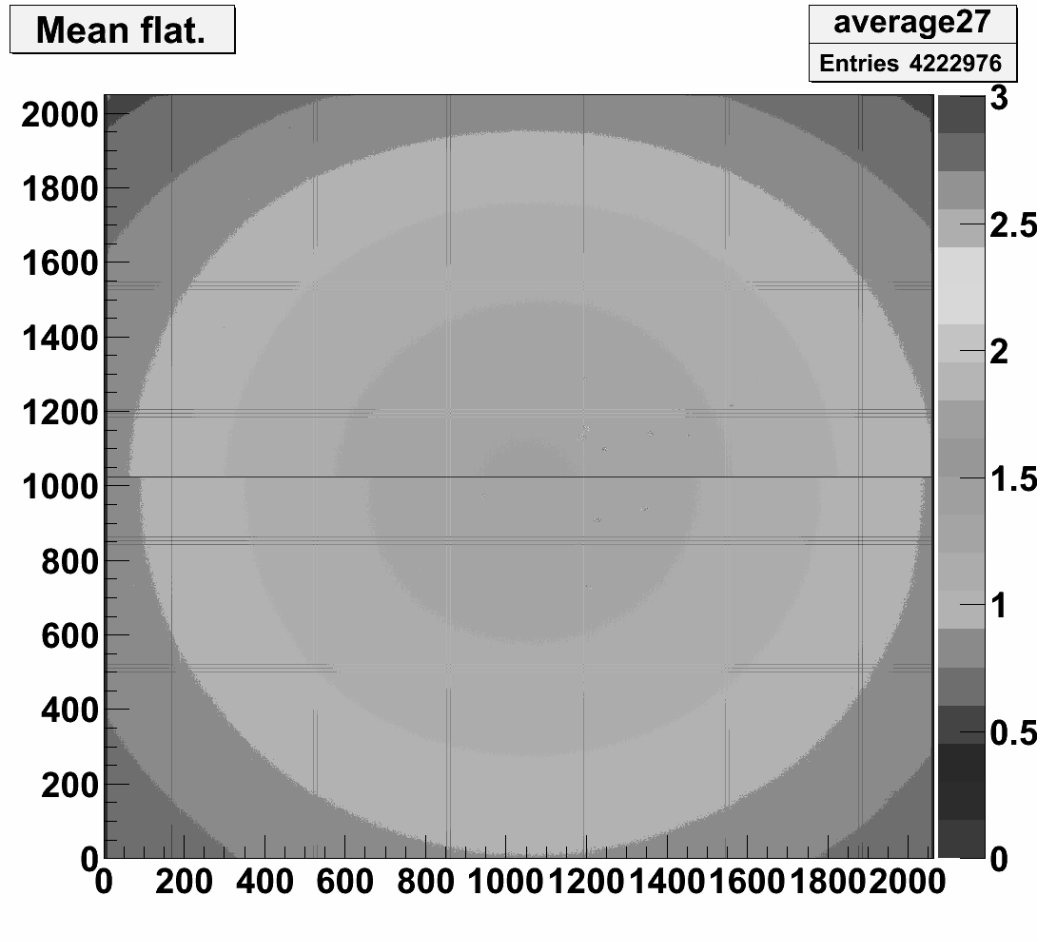
Poniżej zamieszczam także kod źródłowy funckji *flatdiag*.

```
#include <image.cpp>
using namespace std;

void flatdiag(string format, UInt_t first, UInt_t last, UInt_t cam, UInt_t pages, TH1D * haverage, TH1D * hmax, TH1D
{
    //
    // Load all flats in a local table
    //

    TFUShortImg * img;
    UInt_t isize[2];
    UShort_t val;
    UInt_t nx,ny,nz,ix,iy;

    nz=last-first+1;

    for(int id=first;id<=last;id++)
    {
        ostringstream ss;
        ss << id;
        string file=format;
```

```
        if(id<100 && id>=10){
            file += "0";
        }else if(id<10){
            file += "00";
        }
        file += ss.str() + ".fit";
        ss.clear();
        Printf("Reading file %s",file.c_str());
        img = dynamic_cast<TFUShortImg*>TFReadImage(file.c_str(),NULL,1);

        if (img == NULL)
        {
            TFError::PrintErrors();
            return NULL;
        }

        if(id==first)
        {
            img->GetSize(isize, kFALSE);

            Printf("Image %s read, %d x %d ",file.c_str(),isize[1],isize[0]);

            nx=isize[1];
            ny=isize[0];
            Image temp = Image(isize[1],isize[0]);
            Image average = Image(isize[1],isize[0]);

            Printf("Calculating average of %d images.", nz);
        }

        for(iy=0;iy<ny;iy++){
            for(ix=0;ix<nx;ix++){
                val = img[0][iy][ix];
                temp.fill(ix,iy,val);

            }
        }
        average += temp;
        delete img;
    }
    average.scale(1./nz);
    //
    // Writing data
    //
    Printf("Writing %d images.",nz);
    haverage->Fill(pages,average.averageof());
    hmax->Fill(pages,average.max());
    hmin->Fill(pages,average.min());

    /*
    Printf("Writing %d images to file.",nz);
    if( f == NULL ) return;
    haverage->Write();
    hmax->Write();
    hmin->Write();
    */
    return;
}
```

3.2.3 Funkcja *flatstat.*

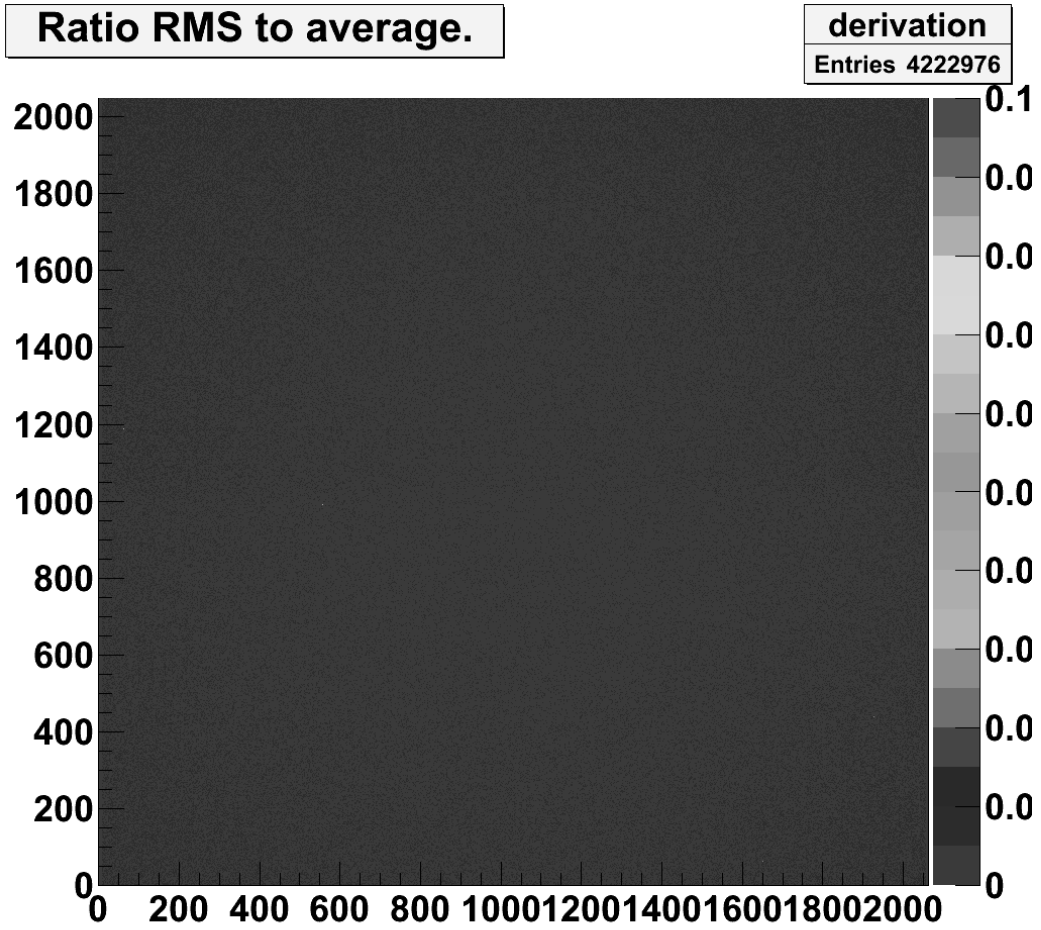
Funkcja *flatstat* oblicza średnie zdjecie, oraz średnie odchylenie kwadratowe z listy podanych zdjęć. Rysunek 5 przedstawia wynik działania funkcji, powstały z 20 zdjęć. Funkcja prezentuje także stosunek średniego odchylenia kwadratowego do uśrednionego zdjęcia, co prezentuje rysunek diagnostic. Funkcja dzieli zdjęcia przez ich średnią wartość. Poniżej kod źródłowy



Rysunek 5: Uśredniona klatka dla 27 kartek wykorzystanych do tłumienia światła dla kamery 37.

funkcji *flatstat.*

```
#include <image.cpp>
using namespace std;

void flatstat(string format, UInt_t first, UInt_t last, UInt_t cam, UInt_t pages, TFile *f)
{
    //
    // Load all flats in a local table
```



Rysunek 6: Średnie odchylenie standardowe dla 27 kartek wykorzystanych do tłumienia światła dla kamery 37.

```
//

TFUShortImg * img;
UInt_t isize[2];
UShort_t val;
Double_t scale=1;
UInt_t nx,ny,nz,nall,ix,iy,iz;


nz=last-first+1;

for(int id=first;id<=last;id++)
{
    ostringstream ss;
    ss << id;
    string file=format;
```

```
    if(id<100 && id>=10){
        file += "0";
    }else if(id<10){
        file += "00";
        cout << file;
    }
    file += ss.str() + ".fit";
    ss.clear();
    Printf("Reading file %s",file.c_str());
    img = dynamic_cast<TFUShortImg*>TFReadImage(file.c_str(),NULL,1);


    if (img == NULL)
    {
        TFError::PrintErrors();
        return NULL;
    }


    if(id==first)
    {
        img->GetSize(isize, kFALSE);

        Printf("Image %s read, %d x %d ",file.c_str(),isize[1],isize[0]);

        nx=isize[1];
        ny=isize[0];
        Image temp = Image(isize[1],isize[0]);
        Image average = Image(isize[1],isize[0]);

        Printf("Calculating average of %d images.", nz);

    }

    for(iy=0;iy<ny;iy++){
        for(ix=0;ix<nx;ix++){
            val = img[0][iy][ix];
            temp.fill(ix,iy,val);
        }
    }
    average += temp;
    delete img;
}
average.scale(1./nz);

for(int id=first;id<=last;id++)
{
    ostringstream ss;
    ss << id;
    string file=format;
    if(id<100 && id>=10){
        file += "0";
    }else if(id<10){
        file += "00";
        cout << file;
    }
    file += ss.str() + ".fit";
    ss.clear();
    Printf("Reading file %s",file.c_str());
    img = dynamic_cast<TFUShortImg*>TFReadImage(file.c_str(),NULL,1);
```

```
    if (img == NULL)
    {
        TFError::PrintErrors();
        return NULL;
    }


    if(id==first)
    {
        img->GetSize(isize, kFALSE);

        Printf("Image %s read, %d x %d ",file.c_str(),isize[1],isize[0]);

        nx=isize[1];
        ny=isize[0];
        Image average_error = Image(isize[1],isize[0]);

        Printf("Calculating standard deviation of average of %d images.",nz);
    }

    for(iy=0;iy<ny;iy++){
        for(ix=0;ix<nx;ix++){
            val = img[0][iy][ix];
            temp.fill(ix,iy,val);

        }
    }
    temp-=average;
    temp*=temp;
    average_error += temp;
    delete img;
}
average_error.scale(1./nz);
average_error.sqrt();

temp = average_error/average;
Printf("Error of average is highter than %f of average.",temp.max(), temp.min());
scale = average.averageof();
average_error.scale(1./scale);
average.scale(1./scale);

//
// Writing data
//

char haverage_name[100];
sprintf(haverage_name,"average%d",pages);
char haverage_error_name[100];
sprintf(haverage_error_name,"average_error%d",pages);
TH2D * haverage = new TH2D(haverage_name,"Mean flat.",nx,0.,1.0*nx,ny,0.,1.0*ny);
TH2D * haverage_error = new TH2D(haverage_error_name,"RMS flat.",nx,0.,1.0*nx,ny,0.,1.0*ny);
TH2D * hderivation = new TH2D("derivation","Ratio RMS to average.",nx,0.,1.0*nx,ny,0.,1.0*ny);

Printf("Writing %d images.",nz);

for(iy=0;iy<ny;iy++){
    for(ix=0;ix<nx;ix++)
    {
        haverage->Fill(ix,iy,average.valueof(ix,iy));
        haverage_error->Fill(ix,iy,average_error.valueof(ix,iy));
        hderivation->Fill(ix,iy,temp.valueof(ix,iy));
```

```
        }
    }

    Printf("Writing %d images to file.",nz);

    haverage->Write();
    haverage_error->Write();

    Printf("Printing graphs for camera %d.", cam);
    TCanvas *c2 = new TCanvas("c2","Average flat.",1094,1031);
    c2->SetGrayscale();
    gStyle->SetOptStat("ne");
    gStyle->SetPalette(1,0) ;
    gPad->SetFillColor(10);
    haverage->Draw("colz");

    Printf("Writing graphs for camera %d to files.", cam);
    char ffile[100];
    sprintf(ffile,"flatstat%d.png",pages);
    c2->Print(ffile, "png");

    hderivation->SetMaximum(0.1);
    hderivation->Draw("colz");

    char dfile[100];
    sprintf(dfile,"diagnostic%d.png",pages);
    c2->Print(dfile, "png");
    return;
}
```

3.2.4 Funkcja *flatana*.

Funkcja *flatana* wywołuje funkcje *flatdiag* oraz *flatstat* dla argumentów odczytanych z pliku *log*. Dane dla pierwszej funkcji są w nim odzielone od danych dla funkcji drugiej przez pusty wiersz. Poniżej zamieszaczam przykładowy plik *log* w celu przedstawienia używanej konwencji.

```
cam7_110502_00003-cam7_110502_00007 33 kartki
cam7_110502_00008-cam7_110502_00012 34 kartki
cam7_110502_00013-cam7_110502_00017 35 kartek
cam7_110502_00018-cam7_110502_00022 36 kartek
cam7_110502_00023-cam7_110502_00027 37 kartek
cam7_110502_00028-cam7_110502_00032 38 kartek
cam7_110502_00033-cam7_110502_00037 39 kartek
cam7_110502_00038-cam7_110502_00042 40 kartek

cam7_110502_00043-cam7_110502_00062 37 kartek
cam7_110502_00063-cam7_110502_00082 38 kartek
```

Funkcja *flatana* dopasowuje także zanik wykładniczy do wyników funkcji *flatdiag* i zapisuje współczynniki dopasowania w pliku *flatdiag.log*. Poniżej przedstawiono przykład tego pliku.

```
Maximum value fit results:
****************************************
Minimizer is Minuit / Migrad
Chi2                      =      28.2413
NDf                       =            4
Edm                       =  1.92421e-10
```

```
NCalls                    =           46
Constant                  =      13.3261   +/-   0.06231
Slope                     =   -0.0896786   +/-   0.0016715

Average value fit results:
****************************************
Minimizer is Minuit / Migrad
Chi2                      =      29.1017
NDf                       =            4
Edm                       =  1.17475e-09
NCalls                    =           52
Constant                  =      12.3886   +/-   0.0821692
Slope                     =    -0.079329   +/-   0.00220246
```

3.3 Prezentacja wyników.

Wyniki dla poszczególnych kamer są prezentowane w plikach *charakteristic.pdf*, będących wynikiem działania skryptu. Oprócz wyników kalibracji oraz opisanej analizy są tam załączone uwagi notowane podczas pracy z daną kamerą.

4 Porównanie kamer.

Wykorzystanie pełnych zdjęć nie stanowi problemu przy poprawianiu zdjęć „offline", jednak ta metoda raczej nie nadaje sie do wykorzystania w wydajnych automatycznych skryptach do analizy zdjęć astronomicznych. Dlatego ważne wydaje się odpowiedzenie na pytanie czy różne kamery maja podobną charakterystyke i czy można stworzyć i wykorzystać reprezentatywne zdjęcie charakterystyki? W tym celu porównywano charakterystyki 8 różnych kamer, 4 z chipem starego typu oraz 4 z chipem nowego typu. Najpierw należało odpowiedzieć na pytanie, czy uzyskane uśrednione zdjęcia dla różnych liczb kartek papieru są do siebie podobne.

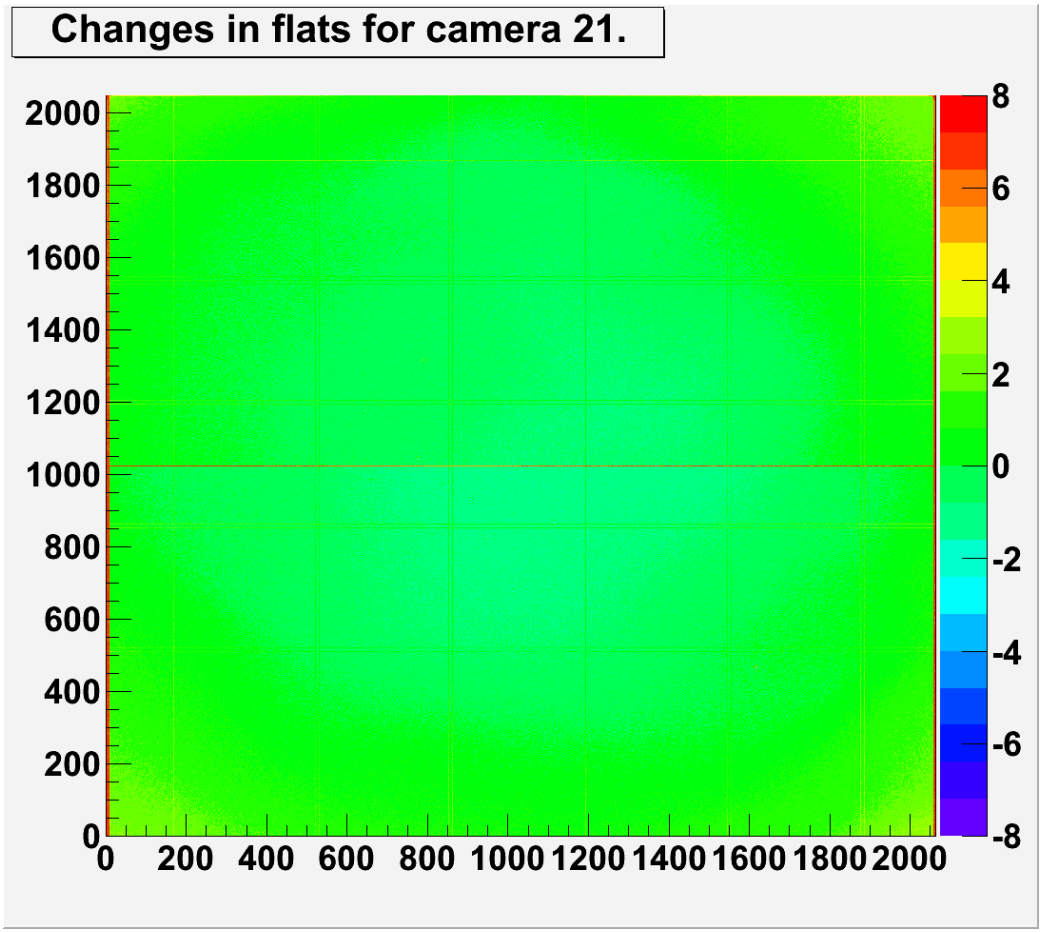
4.1 Zmienność związana z różnym oświetleniem.

W celu porównania dwóch zdjęć zastosowano następującą procedurę:

1. Uzyskanie zdjęcia, które jest średnią dwóch badanych.
2. Otrzymanie zdjęcia będącego różnicą dwóch porównywanych.
3. Podzielnie różnicy przez średnią wartość.

Wyniki otrzymane tą metodą zawierają się w zakresie [-1%,1%] (względna różnica miedzy zdjęciami jest rzędu 1%) dla charakterystyk dobrej jakości (powrócimy do zagadnienia jakości charaktersytki w sekcji 5), oraz [-8%,8%]

dla kamer oszronionych. Wydaje się że wartości te są niewielkie. Rysunek 7 przedstawia wyniki opisanej procedury, w procentach.

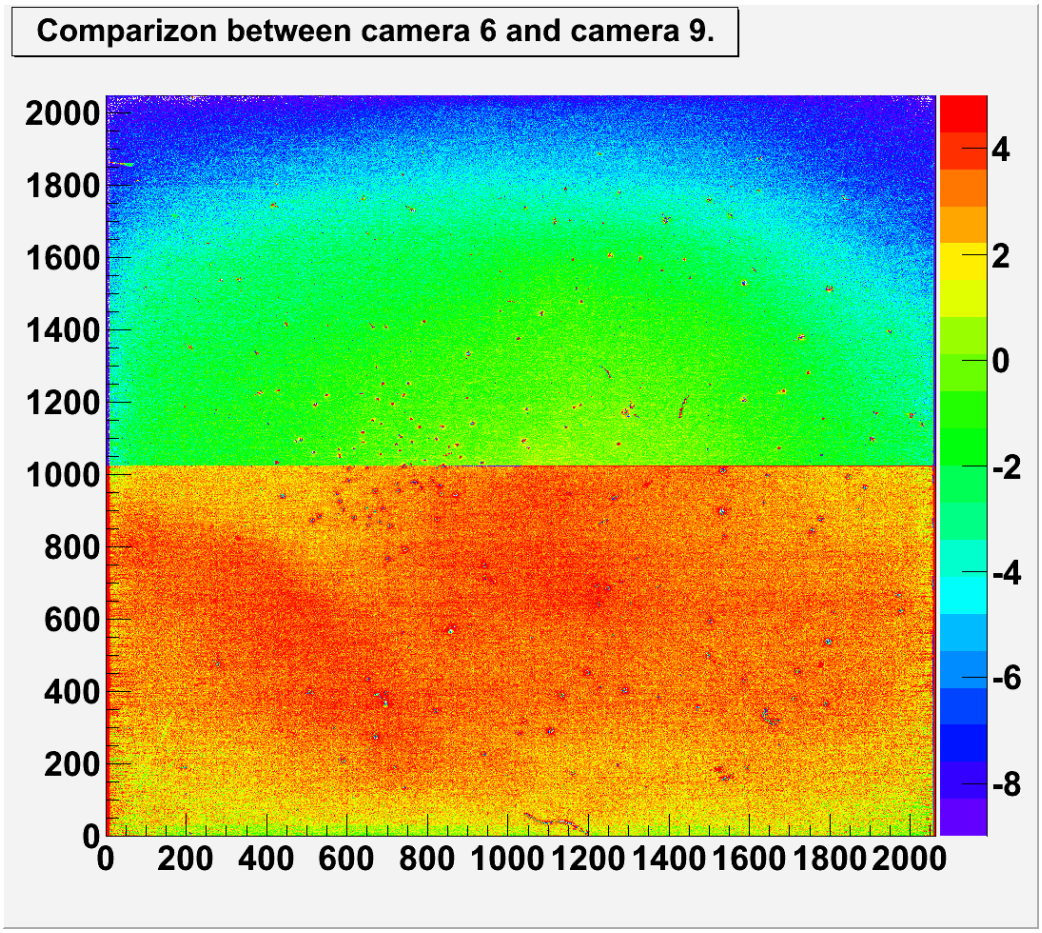


Rysunek 7: Porównanie uśrednionych zdjęć dla różnej ilości kartek dla kamery 21. Względna różnica wyrażona w procentach.

4.2 Zróżnicowanie kamer.

Następnym krokiem w odpowiedzi na pytanie jak kamery różnią się miedzy sobą było zastosowanie omówionej wyżej procedury do uśrednionych po liczbie kartek zdjęć z różnych kamer. Tym razem wyniki rzadko kiedy mieściły się w zakresie [-10%,10%] (częściej różnica wynosiła aż kilkanaście procent). Porównywano ze sobą jedynie kamery o tym samym typie chipu. Ponieważ zmienność wydaje się dość duża nie kontynuowano porównywania dla kamer

o różnych typach chipów. Rysunek 8 przedstawia przykładowe porównanie kamer, w tym przypadku kamery 6 z kamerą 9.



Rysunek 8: Porównanie kamery numer 6 z kamerą numer 9.

4.3 Metodyka.

W tej sekcji zamieszczone są funkcje wykorzystane do porównywania kamer, zawarte w pliku *starter.cpp*. Poniżej załączam jego zawartość.

```
void sqrtTH2D(TH2D* histogram)
{
    Double_t value;
    for(Int_t ix=0;ix<=histogram->GetNbinsX();ix++)
    {
        for(Int_t iy=0;iy<=histogram->GetNbinsY();iy++)
        {
            value = histogram->GetBinContent(ix,iy);
            value = sqrt(value);
```

```
            histogram->SetBinContent(ix,iy,value);
        }
    }
    return;
}

void comparizon(TH2D * first, TH2D * second, UInt_t first_name, UInt_t second_name)
{
    Double_t min=-10;
    Double_t max=10;
    string text="no";

    value->Add(first,second,0.5,0.5);
    value_error->Add(first,second,-1.,1.);
    temp->Divide(value_error,value);
    temp->Scale(100);
    temp->SetMinimum(min);
    temp->SetMaximum(max);

    char comp[100];
    sprintf(comp,"Comparizon between camera %d and camera %d.",first_name, second_name);
    temp->SetTitle(comp);

    gStyle->SetPalette(0);
    temp->Draw("colz");
    c1->Update();

    cout << "Zmienic zakres?" << endl;
    cin >> text;
    while (text=="yes"||text=="y")
    {
        cin >> max >> min;
        temp->SetMinimum(min);
        temp->SetMaximum(max);
        temp->Draw("colz");
        c1->Update();
        cout << "Zmienic zakres?" << endl;
        cin >> text;
    }

    gStyle->SetPalette(1,0);
    temp->Draw("colz");
    char name[100];
    sprintf(name,"%ddo%d.png",first_name, second_name);
    c1->Print(name,"png");
    sprintf(name,"%ddo%d.pdf",first_name, second_name);
    c1->Print(name,"pdf");
}

UInt_t analize(TH2D * first, TH2D * second, Int_t cam, TH2D * camera)
{
    //
    //Creating space
    //

    UInt_t nx = first->GetNbinsX();
    UInt_t ny=first->GetNbinsY();
    TH2D *result = new TH2D("result","Changes in flats.",nx,0.,1.0*nx,ny,0.,1.0*ny);
    TH2D *average = new TH2D("average","average",nx,0.,1.0*nx,ny,0.,1.0*ny);
    TH2D *average_error = new TH2D("average error","average error",nx,0.,1.0*nx,ny,0.,1.0*ny);

    gStyle->SetOptStat("");
```

```
    gStyle->SetPalette(1,0);

    //
    //Printing results for camera
    //

    average->Add(first,second,0.5,0.5);
    average_error->Add(first,second,-1.,1.);
    result->Divide(average_error,average);
    result->Scale(100);
    char title[100];
    sprintf(title,"Changes in flats for camera %d.",cam);
    result->SetTitle(title);
    result->SetMaximum(8);
    result->SetMinimum(-8);
    result->Draw("colz");
    char file[100];
    sprintf(file,"porzadnie%d.png",cam);
    c1->Print(file,"png");

    //
    // Creating output
    //

    char out[100];
    sprintf(out,"Average on camer %d.",cam);
    average->SetNameTitle(out,out);
    *camera=*average;
    return 0;
}

//void starter(){
//
// Starting work
//

TFile f5("flat_cam5.root");
TFile f6("flat_cam6.root");
TFile f7("flat_cam7.root");
TFile f9("flat_cam9.root");
TFile f10("flat_cam10.root");
TFile f21("flat_cam21.root");
TFile f35("flat_cam35.root");
TFile f37("flat_cam37.root");
TH2D *average27_37 = (TH2D*)f37.Get("average27");
TH2D *average28_37 = (TH2D*)f37.Get("average28");
TH2D *average_error27_37 = (TH2D*)f37.Get("average_error27");
TH2D *average_error28_37 = (TH2D*)f37.Get("average_error28");
TH2D *average32_35 = (TH2D*)f35.Get("average32");
TH2D *average33_35 = (TH2D*)f35.Get("average33");
TH2D *average_error32_35 = (TH2D*)f35.Get("average_error32");
TH2D *average_error33_35 = (TH2D*)f35.Get("average_error33");
TH2D *average37_21 = (TH2D*)f21.Get("average37");
TH2D *average38_21 = (TH2D*)f21.Get("average38");
TH2D *average_error37_21 = (TH2D*)f21.Get("average_error37");
TH2D *average_error38_21 = (TH2D*)f21.Get("average_error38");
TH2D *average40_10 = (TH2D*)f10.Get("average40");
TH2D *average41_10 = (TH2D*)f10.Get("average41");
TH2D *average_error40_10 = (TH2D*)f10.Get("average_error40");
TH2D *average_error41_10 = (TH2D*)f10.Get("average_error41");
TH2D *average37_9 = (TH2D*)f9.Get("average37");
TH2D *average38_9 = (TH2D*)f9.Get("average38");
```

```
TH2D *average_error37_9 = (TH2D*)f9.Get("average_error37");
TH2D *average_error38_9 = (TH2D*)f9.Get("average_error38");
TH2D *average37_7 = (TH2D*)f7.Get("average37");
TH2D *average38_7 = (TH2D*)f7.Get("average38");
TH2D *average_error37_7 = (TH2D*)f7.Get("average_error37");
TH2D *average_error38_7 = (TH2D*)f7.Get("average_error38");
TH2D *average34_6 = (TH2D*)f6.Get("average34");
TH2D *average35_6 = (TH2D*)f6.Get("average35");
TH2D *average_error34_6 = (TH2D*)f6.Get("average_error34");
TH2D *average_error35_6 = (TH2D*)f6.Get("average_error35");
TH2D *average37_5 = (TH2D*)f5.Get("average37");
TH2D *average38_5 = (TH2D*)f5.Get("average38");
TH2D *average_error37_5 = (TH2D*)f5.Get("average_error37");
TH2D *average_error38_5 = (TH2D*)f5.Get("average_error38");

TCanvas *c1 = new TCanvas("c1","Changes in flats.",1094,1031);

UInt_t nx = average27_37->GetNbinsX();
UInt_t ny = average27_37->GetNbinsY();

//
//Printing results for camera 5
//

TH2D *camera5 = new TH2D();
UInt_t cam5 = analize (average37_5,average38_5,5,camera5);

//
//Printing results for camera 6
//
TH2D *camera6 = new TH2D();
UInt_t cam6 = analize (average34_6,average35_6,6,camera6);

//
//Printing results for camera 7
//
TH2D *camera7 = new TH2D();
TH2D *cam7_error = new TH2D();
UInt_t cam7 = analize (average37_7,average38_7,7,camera7);

//
//Printing results for camera 9
//
TH2D *camera9 = new TH2D();
UInt_t cam9 = analize (average37_9,average38_9,9,camera9);

//
//Printing results for camera 10
//
TH2D *camera10 = new TH2D();
UInt_t cam10 = analize (average40_10,average41_10,10,camera10);

//
//Printing results for camera 21
//
TH2D *camera21 = new TH2D();
UInt_t cam21 = analize (average37_21,average38_21,21,camera21);

//
//Printing results for camera 35
//
TH2D *camera35 = new TH2D();
```

```
UInt_t cam35 = analize(average32_35,average33_35,35,camera35);

//
//Printing results for camera 37
//
TH2D *camera37 = new TH2D();
UInt_t cam37 = analize(average27_37,average28_37,37,camera37);

TH2D *temp = new TH2D("temp","Changes in flats.",nx,0.,1.0*nx,ny,0.,1.0*ny);
TH2D *value = new TH2D("average","average",nx,0.,1.0*nx,ny,0.,1.0*ny);
TH2D *value_error = new TH2D("average error","average error",nx,0.,1.0*nx,ny,0.,1.0*ny);

//}
```

Funkcja *sqrtTH2D* służy do pierwiastkowania piksel po pikselu histogramu typu klasy **TH2D**. Funkcja *analize* posłużyła do porównywania zdjęć dla tej samej kamery, natomiast funkcja *comparizon* służyła do porównywania różnych kamer. Funkcja *comparizon* zawiera możliwość zmiany zakresu histogramu przed zapisaniem w formie pliku graficznego. Aby uaktywnić tę opcję należy odpowiedzieć na pytanie „`Czy zmienic zakres?`" słowem „`yes`" lub literą „`y`", a następnie podać maksimum histogramu oraz minimum. Po podaniu innego znaku na wejście, funkcja stworzy pliki graficzne z histogramów. Domyślny zakres wynosi [-10,10]. Polecenia z pliku *starter.cpp* nie należące do wymienionych funkcji służą do przygotowania danych do obróbki oraz wykonania funkcji *analize* dla każdej z kamer.

5 Podsumowanie.

Podczas analizy danych okazało się, że wykonanie dobrych zdjęć, które mogłyby posłużyć do otrzymania charakterystyki kamery jest trudne. Kalibracja offsetu jak i wzmocnienia kamer, w odróżnieniu od wyznaczania charakterystyki, nie jest czuła na oszronienie chipu. Nawet niewielkie oszronienie niewidoczne ani gołym okiem na chipie kamery ani na zdjęciach, mocno wpływa na wyznaczenie charakterystyki. Jedynym sposobem na pozbycie się tego problemu się utrzymanie chłodzenia na kamerze wiele godzin (rzędu 8-12) przed robieniem zdjęć. Nawet dwugodzinne chłodzenie nie zapewniało całkowitego zaniku oszronienia chipu. Ciekawym faktem i wartym zbadania jest zanikanie oszronienia, pojawiającego się po właczeniu chłodzenia kamery. Poza cienkim oszronieniem na dużej powierzchni (przede wszystkim brzegach) chipu, które znika z czasem, na chipie pojawiają się także kryształki lodu. Nie znikają one z czasem i mają tendencje do powstawania w stałych punktach kamery. Ze względu na oszronienie analizę kamer 5 oraz 10 należy powtórzyć. Kamera 21 ma uszkodzony rząd pikseli na górnej połówce chipu z prawej strony. Pozostałe uwagi w plikach opisujących poszczególne kamery.