

Toolkit for testing scientific CCD cameras

Janusz Użycki^a, Lech Mankiewicz^b, Marcin Molak^a, Grzegorz Wrochna^c

^a Faculty of Physics, Warsaw University of Technology

^b Center for Theoretical Physics PAS, Warsaw;

^c Soltan Institute for Nuclear Studies, Warsaw.

ABSTRACT

The CCD Toolkit [1] is a software tool for testing CCD cameras which allows to measure important characteristics of a camera like readout noise, total gain, dark current, ‘hot’ pixels, useful area, etc. The application makes a statistical analysis of images saved in files with FITS format, commonly used in astronomy. A graphical interface is based on the ROOT package, which offers high functionality and flexibility. The program was developed in a way to ensure future compatibility with different operating systems: Windows and Linux. The CCD Toolkit was created for the “Pi of the Sky” project collaboration [2].

Keywords: Gamma Ray Bursts (GRB), astronomy, CCD cameras, image processing

1. INTRODUCTION

The “Pi of the Sky” project [2] is conducted by Soltan Institute for Nuclear Studies in collaboration with Center of Theoretical Physics, Warsaw University and Warsaw University of Technology. Its main goal is continuous observation and real-time analysis of the sky in a search for optical flashes of cosmological origin. Sky images are taken by CCD cameras. Because of specific requirements, designing and developing the cameras is a part of the project. CCD sensors are available in dimensions with diameter bigger than half inch, which is crucial in professional astronomy applications. Low noise level and high sensitivity are also very important.

The main disadvantage of CCD matrix is a complicated control circuit. A specially formed clock signals, amplifier and analog to digital converter (ADC) are necessary. Therefore, the final result depends on many parameters like gain of the amplifier, shape of the clock signals and timings. There is a need for a tool which gives fast and easy way to estimate quality of camera images in a lab. It is required to know level of noise, gain of all camera blocks, dark current, active area of CCD matrix. It is possible to get most of that information by means of simple statistical analysis.

In our cameras the CCD matrix is controlled by an FPGA circuit. It allows us to modify easily the shape of CCD signals. After each change it is necessary to test performance of the camera. Then camera is placed in a light-tight, cold box (about -17°C) to reduce temperature drifts, thermal noise and light leak. Several exposures are taken and the data are stored on a disk with the format FITS (Flexible Image Transport System) commonly used in astronomy. Then, the CCD Toolkit is used to read the data and make the analysis.

The method of testing a camera is based on analysis of image histograms and their projections. For the noise measurement so called “dark images” needs to be taken. A dark image is a photo which is taken without the light from an environment. Hence, the image contains only results of dark current and noise. Initially all analysis of the images were done using Iris [3] application step by step but this way was very inconvenient. This is the reason why the CCD Toolkit application was created.

2. REQUIREMENTS AND FEATURES

The cameras are equipped with an ADC of 16 bits of resolution. Full size of the data is 2062 x 2048 pixels, but 16 pixels at each edge are hidden and the first 16 columns are due to a readout buffer. The illuminated area is thus 2032x2032 pixels. Hence, the size of a FITS file (raw data from the CCD) is around 8MB ($2k \times 2k \times 2\text{bytes}$).

Basic requirements for the CCD Toolkit applications are:

PART I – dark current, readout noise and uniformity estimates:

- ✓ reading FITS images from a disk
- ✓ displaying images
- ✓ making subtraction of two images
- ✓ plotting single pixel distribution and performing Gaussian fit to it
- ✓ plotting X-projection (profile) distribution (as above)
- ✓ plotting Y-projection (profile) distribution (as above)
- ✓ allowing to cut frame edges (full frame: 2062×2048 , visible part: 2032×2032)

PART II - gain measurement:

- ✓ cutting out frame edges
- ✓ dividing image into sub-regions (e.g. 100×100 or 1800×10)
- ✓ plotting distribution for each region
- ✓ plotting thumbnails of distributions and fitted Gaussians curves
- ✓ plotting sigma in function of $(\text{mean})^2$
- ✓ fitting straight line

System requirements for the tool:

- ✓ should operate on Windows NT family system
- ✓ should use the ROOT package for graphical analysis [4]
- ✓ should read images, stored in the FITS format
(one can use the FITSIO library [5] or the AstroROOT package [6])

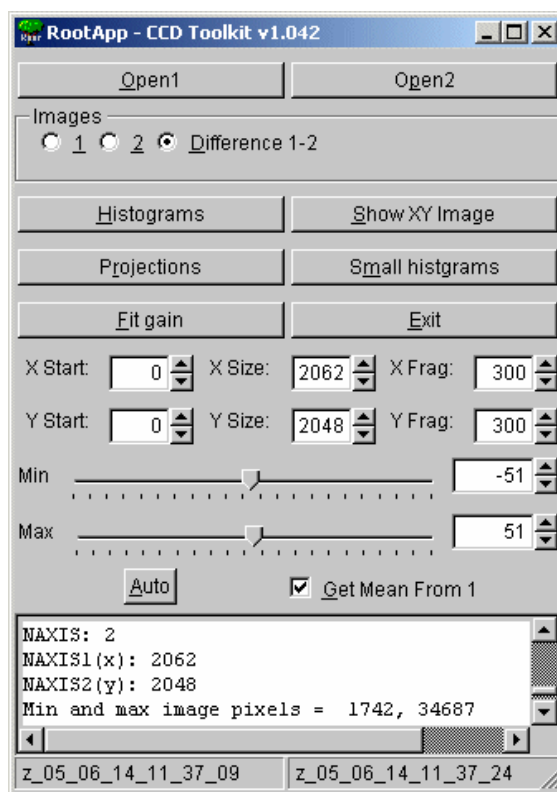


Figure 1. User interface of the CCD Toolkit.

Taking into account that main users of our application are electronics engineers used to work with PC in the Windows environment, the CCD Toolkit was designed for PC platform and Windows XP and 2000 operating systems. We decided to use the ROOT package library [4] which is well known in particle physics community (eg. CERN). To improve an overall efficiency the C/C++ language was chosen and the source code was compiled (instead of being interpreted by Cint). The Microsoft Visual C++ environment was used for development to ensure compatibility with Windows. It was the best solution for the ROOT package, because CERN supports that IDE and prepares pre-compiled DLL libraries for each new version of ROOT. To handle images from FITS files the FITSIO library [5] was used. Unfortunately we could not use AstroROOT, because Microsoft systems are not supported. Only versions for Linux are available. For a possible future portability of the application between Windows and Linux systems the graphical user interface (GUI) of our application was developed completely by using the ROOT package classes.

To eliminate the effect of the constant offset, which is somewhat different for each pixel and to reduce errors caused by 'hot' pixels (exhibiting high value in spite of darkness) it is convenient to subtract two successive frames (both read from the FITS files) (Figure 1). A lot of main errors of loaded picture(s) can be caught instantly by simple displayed preview (Figure 2). This feature allows user to check the image visually and to find incorrect parts (e.g. 'hot' pixels). To mark the best useful area of the CCD matrix the possibility of edge cutting was implemented.

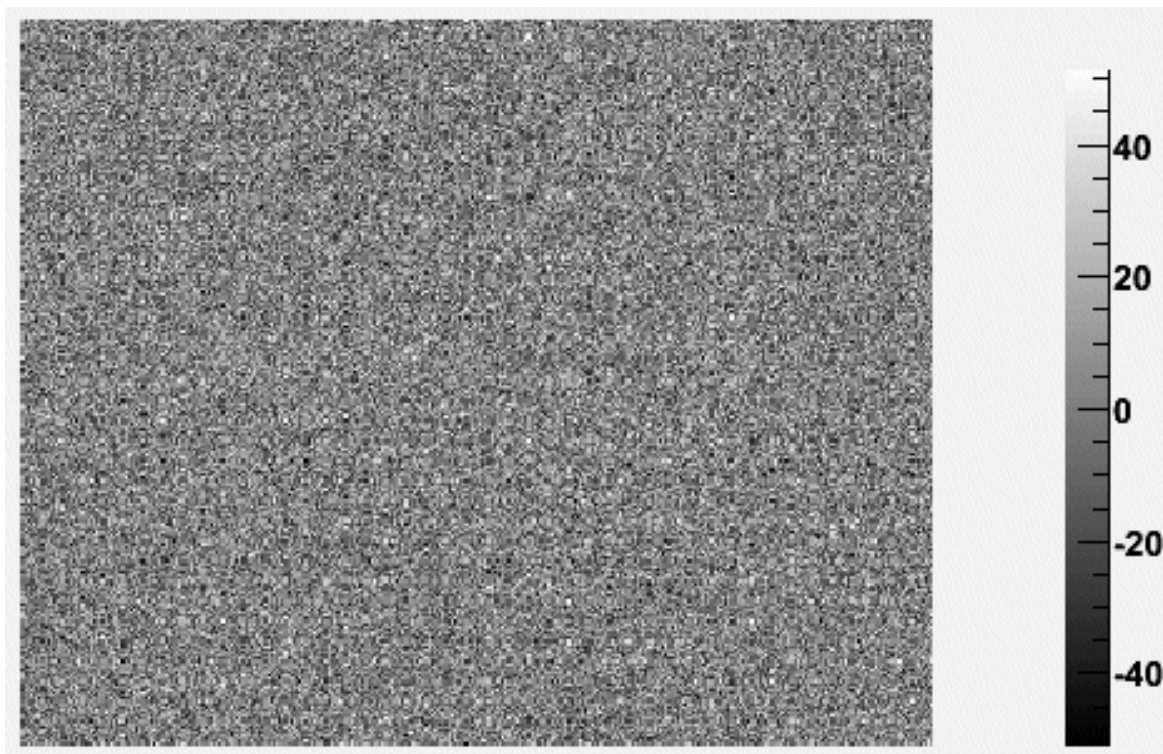


Figure 2. XY Image preview and grey scale palette

For the better image display an option is available to find automatically the optimum brightness range, using single pixel distribution from XY histogram (Figure 3). The range is defined as $\text{mean} \pm 3 \times \text{sigma}$.

Simple visual inspection cannot replace quantitative analysis of the image. To observe the dark current effect and possible nonuniformities the application makes X and Y profiles (projection) by counting average value for each column and each row (Figure 3). Details of the effects can be measured by histogram analysis of either all or parts of the profiles (Figure 4). Dividing the image into regions and making single pixel distribution for each region (Figure 5) allows us to find differences between regions (eg. sensitivity gradient).

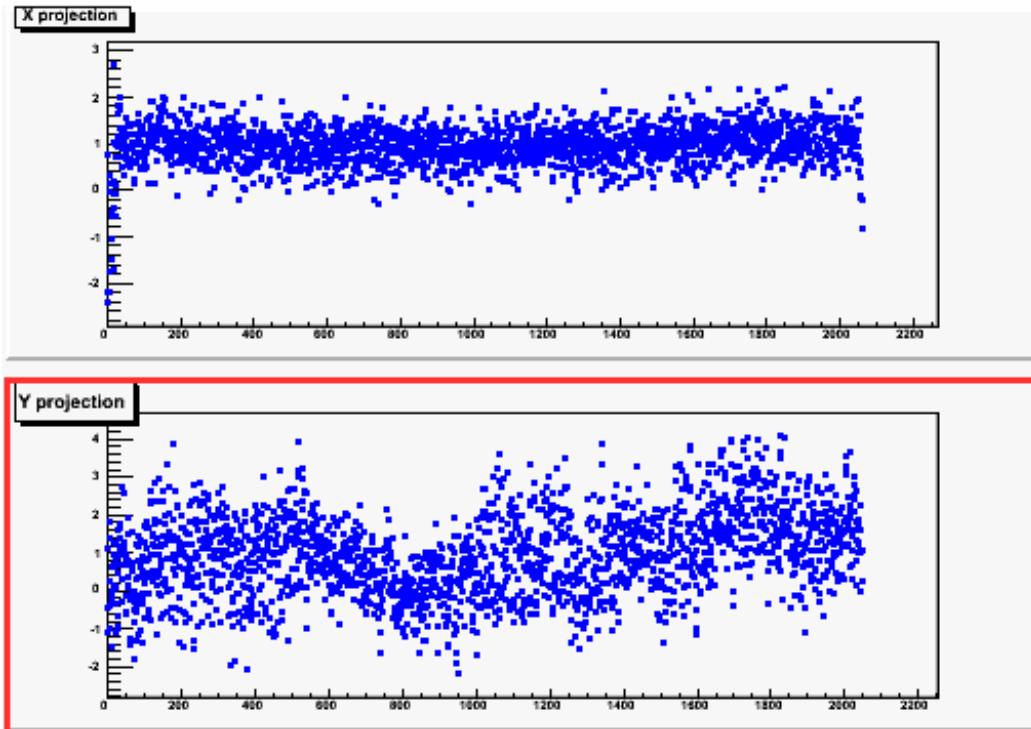


Figure 3. X and Y profiles (side-projections) Each point corresponds to an average value of all pixels in a given column (X) or row (Y)

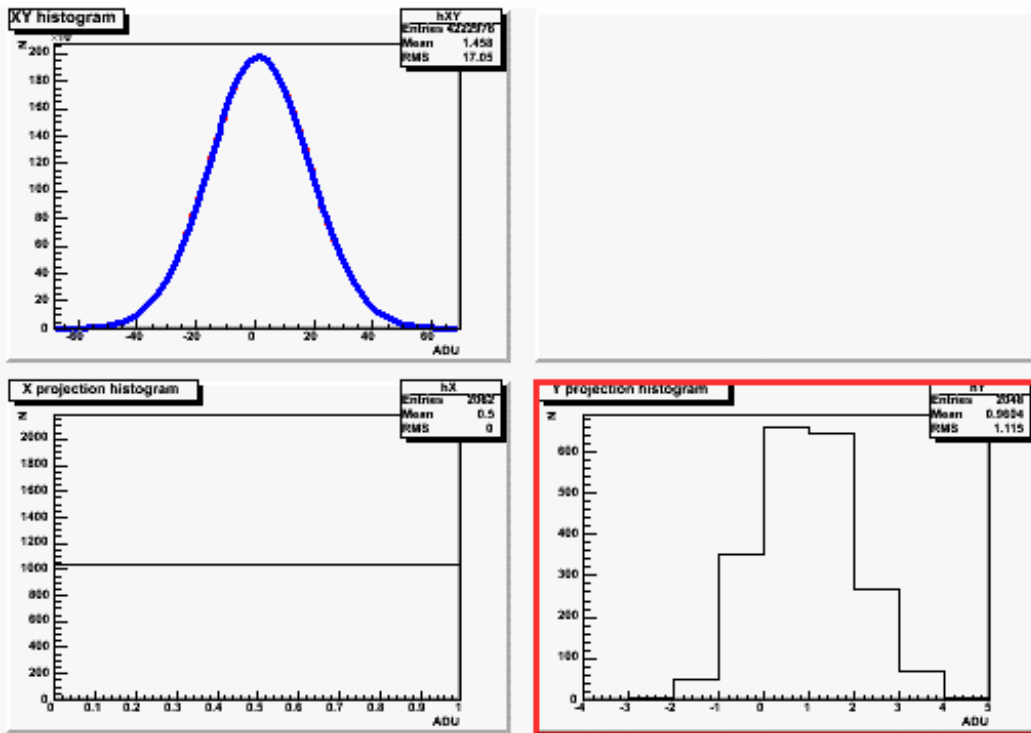


Figure 4. Distributions of single pixel values (XY), average column values (X), and average row values (Y).

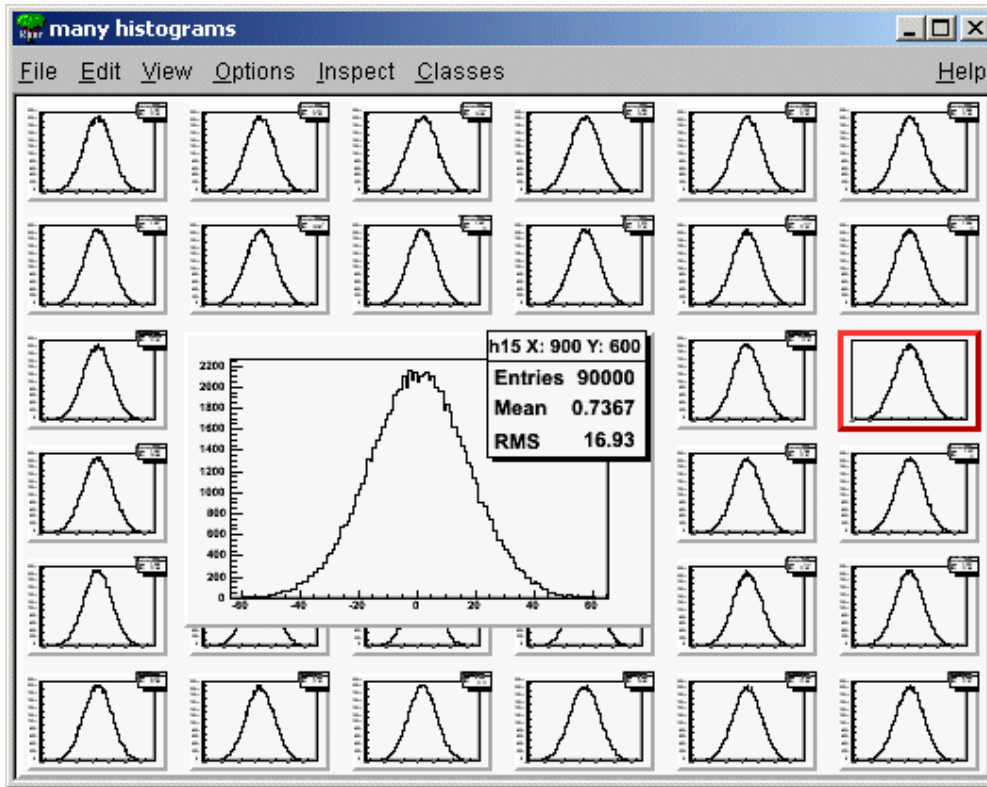


Figure 5. Histograms for each region after image division.

The most advanced part of the application is the gain measurement option. The method used is based on the fact, that the number of electrons collected in a single pixel obeys the Poisson distribution, which has only one free parameter, so that the variance is uniquely determined by the average. This fact does not depend off the source of electrons, which could be light conversion, thermal noise or dark current.

Let us denote the number of electrons collected in a single pixel by n . After readout, amplification and conversion it manifests as a certain number of ADC counts (or Arbitrary Digital Units – ADU) v . Conversion factor of the device (sometimes referred to as “gain”) can be defined as $k = n/v$. The number of electrons n obeys the Poisson distribution with the average $\langle n \rangle$ and the variance $\sigma_n^2 = \langle n \rangle$. The distribution after ADC conversion is characterised by

$$\begin{aligned} \text{the average} & \quad \langle v \rangle = v_0 + \langle n \rangle / k & \quad (\text{where } v_0 \text{ is some offset}) \\ \text{and the variance} & \quad \sigma_v^2 = \sigma_n^2 / k^2 = \langle n \rangle / k^2 = \langle v \rangle / k - v_0 / k. \end{aligned}$$

The distribution of v is no longer poissonian, but for large $\langle v \rangle$ it could be well approximated by a gaussian. Having several data samples with different average values $\langle v \rangle$ one can plot the variance σ_v^2 versus the average $\langle v \rangle$ and perform a straight line fit $\sigma_v^2 = a \cdot \langle v \rangle + b$. The resulting coefficient a will be just an inverse of the conversion factor:

$$a = 1 / k$$

The main advantage of this method is that it is enough to use the output data only and no knowledge about the hardware (like CCD and ADC conversion factors or preamplifier gain) is needed to obtain the result. In practice, different samples can be obtained by illuminating uniformly the CCD with different amount of light. One can also use a single frame with some illumination gradient and divide it into several regions. Each region should be small enough, that one can assume it is uniformly illuminated and large enough to ensure good statistics for estimating σ_v and $\langle v \rangle$. In such case the values v from pixels in a given region should obey gaussian distribution which parameters can be determined e.g. by least square fitting. One has to keep in mind, however, that each pixel might have slightly different offset, which makes the resulting distribution wider. Fortunately, this does not affect the method, because the v_0 distribution is usually also a gaussian.

Measured total variance σ_{tot}^2 is thus a sum of σ_v^2 and the offset variance σ_0^2 . It changes the b parameter of the fit, but leaves intact the $a = 1/k$:

$$\sigma_{\text{tot}}^2 = \sigma_v^2 + \sigma_0^2 = \langle v \rangle / k - v_0 / k + \sigma_0^2$$

In the CCD Toolkit the method described above is implemented as follows. The CCD matrix is divided into a number of regions. For each region a distribution of pixel values v is obtain in form of a histogram (Figure 5). A gaussian fit performed for each histograms gives a couple of parameters: the average $\langle v \rangle$ and the variance σ_v^2 . Plotting the variance σ_v^2 versus the average $\langle v \rangle$ (Figure 6) allows us to make a linear fit $\sigma_v^2 = a \langle v \rangle + b$. The parameter a determines the total gain $k = 1/a$ (conversion factor for the camera).

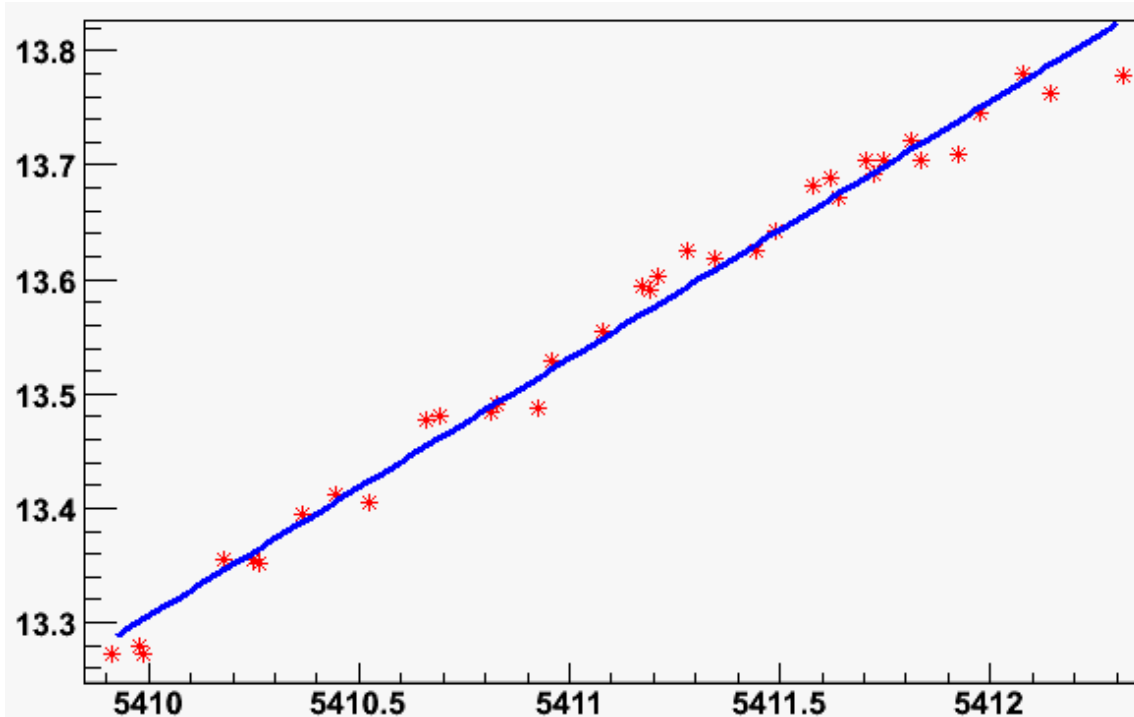


Figure 6. The gain plot: the variance σ_v^2 versus the average $\langle v \rangle$ together with a straight line fit.

3. PROBLEMS AND SOLUTIONS

The first attempts to write the program were done under Windows 98 SE system with the ROOT package, version 3.04.02. But the next version does not support Microsoft systems other than NT family. Moreover, there were some problems with compilation under Microsoft Visual Studio C++, version 6.0. Our final environment was: the ROOT package, version 4.03, Windows XP and Visual Studio .NET 2003. The change of programming IDE was caused by implementation of the graphical user interface from the ROOT package.

The ROOT installation is described in detail at the ROOT home page: <http://root.cern.ch>. It is important to remember to set the proper path to the ROOT directory. It can be done by modifying system settings: *system properties* | *advanced* | *environment variables* | *system variables*: \$ROOTSYS = C:\ROOT for example.

The CFITSIO library was used to load the image and perform all related calculations. Initially there were C functions but later it was moved to C++ TImgFits class. The TImgFits contains all data about one image and has methods for loading, subtraction, cutting picture and data converting (filling different histograms and graphs, dividing into regions etc.). In order to optimize the code, the FillN method was used to fill the histograms instead of simple Fill method.

In order to display 16-bit image properly, a scale conversion must have been performed. It was achieved by finding minimum and maximum brightness for each image and defining an optimized grey palette. For that, modification of the ROOT's TASImage was necessary. A new class TASFImage was created, which inherits TASImage and TImgFits classes. Some parts of the source code from TASImagePaletteEditor class have been found useful.

During the ROOT's interface implementation there was a problem with compilation under Visual Studio. A solution was using RootCINT to generate special dictionary files (which translate library function into API commands) for the ROOT's interface and classes. After proper configuration of Visual Studio all process is done automatically [7, 8]. The main disadvantage of the ROOT API interface for GUI is the difficulty to make window's drafts, buttons, etc. One has to make all event structure and connect by the ROOT dictionary of visual objects with events – make actions. ROOT GUI Builder was not used. Instead, all GUI was written manually as source code.

Using GUI from the ROOT turned out to be rather complicated. Careful memory management of ROOT's windows was necessary to avoid memory leaks. Finally, a lot of checks were added to guard the application stability. For example, an attempt to make fit to an empty data set should not crush the program.

4. CONCLUSIONS

Selection of the ROOT package as a base of the CCD Toolkit was very convenient, however, one has to keep in mind that the ROOT is primarily developed for Linux. Versions for Windows are less supported. On the other hand, the ROOT offers portability and high flexibility. One can use compiled languages like C and C++ to accelerate applications. The convenience of high level language – classes and interfaces also cannot be neglected. By using the FITS format, which is very popular in astronomy, one gains compatibility with a very large range of applications.

In the past, requirements for a PC hardware seemed to be larger than it utterly occurred. For three images in memory it is necessary to reserve about 50MB ($3 \times 8\text{MB} \times 2$, where the last factor 2 is because one has to have two copies of each image for manipulation). Today, a PC with 512MB is nothing extraordinary and we do not need to worry about RAM limitations.

ACKNOWLEDGMENTS

This work was supported by the grant nr 503 G 1050 0039 005 from the Dean of the Faculty of Physics of the Warsaw University of Technology. The authors acknowledge also the support of the Ministry of Science and Information Society Technologies under grant 1 P03B 064 29.

REFERENCES

- [1] CCD Toolkit homepage
<http://grb.fuw.edu.pl/pi/user/molak/ccdtoolkit/index.html>
- [2] Ćwiok M., Dominik W., Husejko M., Wrochna G., "Search for Optical Flashes Accompanying Gamma Ray Bursts", Proc. SPIE, 5484, s. 283-289, 2004.
- [3] IRIS homepage
<http://www.astrosurf.org/buil/us/iris/iris.htm>
- [4] ROOT homepage
<http://root.cern.ch/>
- [5] CFITSIO homepage
<http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html>
- [6] AstroROOT homepage
<http://isdc.unige.ch/Soft/AstroRoot/>
- [7] ROOT and CINT: example of compilation under Visual Studio for Windows systems
<http://d0.phys.washington.edu/~haas/windows%20compiling.html>
- [8] ROOT and CINT: dictionary generation
<http://root.cern.ch/root/CintGenerator.html#DictionaryNOIO>