Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Krzysztof Goliński
Nr albumu: 245284

Narzędzie graficzne do testowania i obsługi kamer CCD w eksperymencie „Pi of the Sky”

Praca magisterska
na kierunku INFORMATYKA

Praca wykonana pod kierunkiem
dra Marcina Sokołowskiego
Instytut Problemów Jądrowych
im. Andrzeja Sołtana

Grudzień 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Główną rolę w eksperymencie astronomicznym „Pi of the Sky" odgrywają kamery CCD. Ich zestaw w ciągu nocy wykonuje tysiące zdjęć różnych części nieba. Zamiarem poniżej pracy było stworzenie graficznego narzędzia do obsługi oraz automatycznego testowania kamer, budowanych do celów astronomicznych. By zrealizować założenia eksperymentu „Pi of the Sky" zostaną zbudowane 32 kamery CCD, dlatego automatyzacja procedury ich testowania jest bardzo ważna. Praca zawiera opis wszystkich funkcji programu, a także informacje na temat technik i narzędzi wykorzystanych do jego stworzenia.

Słowa kluczowe

kamera CCD, oprogramowanie do kamer CCD, testowanie kamer CCD, test2K2Kgui, ciągła obserwacja nieba

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.3 Informatyka

Klasyfikacja tematyczna

D. Software
D.1 Programming techniques
D.1.5 Object-oriented Programming

Tytuł pracy w języku angielskim

Graphical tool for testing and operating CCD cameras in the „Pi of the Sky" project

Spis treści

Wprowadzenie

Celem eksperymentu „Pi of the Sky” ([1],[2]) jest badanie błysków optycznych związanych z rozbłyskami promieniowania gamma (ang. *Gamma Ray Burst*) [3] oraz innych optycznych rozbłysków pochodzenia kosmicznego. W odróżnieniu od klasycznych zjawisk astronomicznych, przy których zmiany zachodzą w skali miesięcy, lat, błyski gamma są bardzo spektakularne. Czasy ich trwania wynoszą od ułamków do kilkuset sekund. Jednak ulotność tych zjawisk stanowi dla badaczy poważny problem. Jeśli zauważą rozbłysk gamma, muszą oni w bardzo krótkim czasie skierować swoje teleskopy optyczne by go zarejestrować.

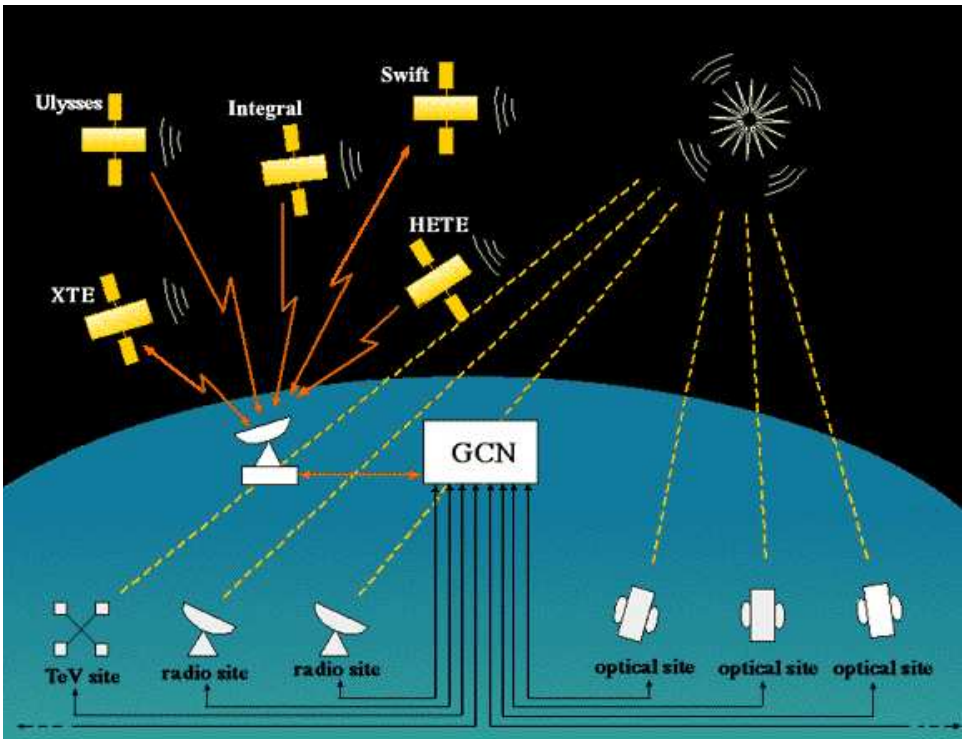
Eksperyment „Pi of the Sky” próbuje rozwiązać ten problem poprzez obserwację jak największego obszaru nocnego nieba. Docelowo w przedsięwzięciu mają być użyte dwa zbiory złożone z 16 kamer CCD (zob. rozdział 1), oddalonych od siebie o około 100km. Łącznie urządzenia mają obserwować około 1/6 nieba. Ten obszar będzie pokrywał się z polem widzenia satelity Swift [4], który jeśli namierzy rozbłysk gamma, poinformuje o tym fakcie stacje naziemne za pośrednictwem sieci GCN (ang. *The Gamma ray bursts Coordinates Network*) [5] (rys. 1). W Las Campanas (Chile) od roku 2004 działa już prototypowa wersja systemu złożona z dwóch kamer (rys. 2). Podczas jednej z faz eksperymentu, trwającej ponad rok, wykonano około 600 tysięcy zdjęć. Potwierdzeniem, że przyjęta strategia obserwacji nieba jest słuszna było zaobserwowanie błysku z dnia 19 marca 2008 roku – GRB080319B [6].

Eksperyment „Pi of the Sky” to nie tylko same kamery. W jego skład wchodzi także cały system zarządzania składający się z wielu programów i modułów. Całe oprogramowanie działa na systemie operacyjnym *Fedora* [7] – jednej z dystrybucji linuksa. Najważniejsze programy i moduły to:

- PIMAN
  Główny program zarządzający całym systemem i wszystkimi modułami
- MOUNT
  Moduł odpowiedzialny za sterowanie montażem i odpowiednim ustawieniem kamery astronomicznej względem obserwowanego obiektu
- DAQ (ang. *Data Acquisition System*)
  Moduł odpowiedzialny za pobieranie i analizę zdjęć z kamer CCD
- PISHELL i RUNSCRIPT
  Programy klienckie umożliwiające wysyłanie poleceń do zarządcy systemu (PIMAN-a).

Większość wyżej opisanych elementów systemu powstała dzięki pracy członków zespołu „Pi of the Sky”, w którego skład wchodzą pracownicy i studenci kilkunastu instytucji.

Przed autorem niniejszej pracy postawiono zadanie dołożenia kolejnej cegiełki do eksperymentu. Jego celem było stworzenie programu z interfejsem graficznym do testowania i obsługi kamer CCD. Aplikacja miała umożliwić łatwą obsługę urządzenia, zarówno zaawansowanym jak i początkującym użytkownikom. Ważnym zadaniem programu miała być także możliwość



Rysunek 1: Komunikacja w sieci GCN

Rysunek 2: Prototypowy system złożony z 2 kamer działający od czerwca 2004 w obserwatorium Las Campanas w Chile

automatycznego testowania kamer. W tym celu dla potrzeb eksperymentu „Pi of the Sky" powstał prosty składniowo język skryptowy.

Plan poniżej pracy przedstawia się następująco. W rozdziale 1 zostały zawarte najważniejsze informacje potrzebne do zrozumienia działania kamer użytych w eksperymencie oraz obsługującego je sterownika. Rozdział 2 opisuje narzędzia, które zostały użyte do stworzenia programu *test2K2Kgui* będącego przedmiotem poniższej pracy. Kolejny, 3 rozdział, to szczegółowy opis programu *test2K2Kgui*, jego konfiguracji jak również opis tworzenia skryptów do automatycznego testowania kamer CCD.

Rozdział 1

Kamery CCD w eksperymencie „Pi of the Sky”

Z faktu, że kamery CCD (rys. 1.1) wykorzystywane w eksperymencie będą zamontowane w obserwatorium Las Campanas (Chile) wynika, iż muszą one spełniać specyficzne wymagania. Użytkownik nie będzie miał do nich bezpośredniego dostępu, dlatego jednym z najważniejszych założeń jest pełna zdalna kontrola urządzenia przez Internet. Z tego samego powodu kamery muszą być wyposażone w czujniki temperatury i wilgotności. Kolejnym równie istotnym założeniem jest jak najdłuższa żywotność migawki oraz innych podzespołów kamery. Im rzadziej urządzenie będzie podlegało awariom, tym dłużej będzie pracowało na rzecz eksperymentu i nie będzie konieczne częste sprowadzanie jego do Polski, by dokonywać napraw. Również kwestia ceny kamery była istotnym parametrem wyboru produktu. Ostatecznie zdecydowano, że członkowie eksperymentu „Pi of the Sky” sami zaprojektują i zbudują urządzenia spełniające ich wymagania. Poniżej znajduje się lista zawierająca opis najważniejszych elementów kamery:

- matryca CCD (ang. *Charge Coupled Device*)
  Składa się z układu elementów światłoczułych, które zbierają informacje o natężeniu padającego na nie światła. Jest to najważniejszy element urządzenia, stąd jego nazwa – kamera CCD. Zastosowano w eksperymencie „Pi of the Sky” matryce CCD firmy *Fairchild* i *STA* o wymiarach 2000×2000 pikseli i rozmiarze piksela $15\mu m$.

- przetwornik analogowo-cyfrowy (ang. *Analog to Digital Controller – ADC*)
  Przetwarza sygnał odczytany z matrycy CCD na postać cyfrową

- interfejs USB (ang. *Universal Serial Bus*)
  Pozwala połączyć się z lokalną kamerą za pomocą uniwersalnej magistrali szeregowej. Maksymalna prędkość transmisji wynosi 52MB/s.

- interfejs Gigabit Ethernet
  Pozwala połączyć się ze zdalną kamerą za pomocą Gigabitowego Ethernetu, protokół UDP. Maksymalna prędkość transmisji wynosi 100MB/s.

- układ FPGA firmy *Altera*
  Steruje prędkością odczytu z matrycy CCD i generuje sygnały sterujące jej pracą.

- układ *Cypress FX2 USB CY7C68013*
  Odpowiada za transmisję danych przez USB.

Rysunek 1.1: Kamera CCD skonstruowana na potrzeby eksperymentu „Pi of the Sky”

- LNA (ang. *Low Noise Amplifier*)
  Niskoszumowy przedwzmacniacz, którego zadaniem jest wzmocnienie sygnału odczytywanego z matrycy CCD.

- ogniwo Peltiera
  Służy do chłodzenia matrycy CCD na zasadzie efektu termoelektrycznego.

Szczegółowy opis zbudowanych kamer CCD można znaleźć w [8].

Aby wszystko działało zgodnie z założeniami eksperymentu, członkowie zespołu musieli również napisać oprogramowanie obsługujące stworzone przez nich kamery CCD. Sterowniki urządzeń zostały napisane w dwóch językach: *C* i *C++*.

Zgodnie z konwencją programowania obiektowego wydzielono klasę *DeviceBase*, która jest abstrakcją dowolnego typu kamery użytej w eksperymencie. Zawiera ona podstawowe definicje stałych, implementacje ogólnych funkcji oraz deklaracje metod wirtualnych – interfejsu urządzenia. Wszystkie klasy reprezentujące poszczególne rodzaje kamer użytych w eksperymencie muszą po niej dziedziczyć.

Klasa *Device2K2K* jest implementacją kamery korzystającej jedynie z interfejsu USB. Do poprawnego działania wymaga załadowanego do pamięci modułu jądra systemu Linux o nazwie *pikam.ko*. Klasa *Device2K2K* otwiera pliki urządzenia (/dev/usb/pikapN) i obsługuje komunikację z kamerami.

Klasa *DeviceEth2K2K* dziedziczy publicznie po *Device2K2K* i służy do obsługi kamer korzystających z gigabitowego interfejsu Ethernet. W odróżnieniu od klasy nadrzędnej, nie wymaga załadowania modułu jądra *pikam.ko* do pamięci. Jest tak dlatego, iż do komunikacji z urządzeniami wykorzystuje protokół NUDP [9] zaimplementowany z pomocą biblioteki *Sockets* [10]. Sterownik kamery został skompilowany do postaci biblioteki *libpicamdriver.a*.

Szczegółowe informacje dotyczące sterownika kamer CCD można znaleźć w [1].

Rozdział 2

Opis zastosowanych narzędzi

Cały system do obsługi eksperymentu „Pi of the Sky" działa pod kontrolą *Fedory* [7] – dystrybucji linuksa. Do tworzenia kolejnych programów, modułów w eksperymencie korzysta się z oprogramowania dostępnego na wolnej licencji: Powszechnej Licencji Publicznej GNU (ang. *GNU General Public License – GNU GPL*). Niesie to ze sobą wiele korzyści. Przede wszystkim kod źródłowy wszystkich programów wydanych na tej licencji jest otwarty. Można go czytać, modyfikować i w dowolny sposób wykorzystywać dla potrzeb projektu. Dla uczelni, a także członków zespołu „Pi of the Sky", nie bez znaczenia jest również fakt, iż oprogramowanie na licencji GNU GPL jest bezpłatne. Również przy tworzeniu programu będącego przedmiotem tej pracy nie odstąpiono od tej konwencji. W rozdziale zostały opisane narzędzia wykorzystane do stworzenia *text2K2Kgui*.

2.1. System *ROOT*

System ROOT [11] to zaawansowane narzędzie służące do analizy dużej ilości danych tworzone w Europejskiej Organizacji Badań Jądrowych CERN (*fr. Conseil Européen pour la Recherche Nucléaire)*. Pozwala między innymi na: prowadzanie dokładnych obliczeń numerycznych, rysowanie grafów, histogramów, modeli trójwymiarowych. Cały system jest stworzony w pełni obiektowo. W prosty sposób integruje się z językami programowania *C++* i *Python*.

W tym punkcie skupimy się jedynie na bardzo małym fragmencie systemu wykorzystanym przy tworzeniu programu a mianowicie na interfejsie graficznym użytkownika.

Obiekty graficzne

Aby w programie *C++* móc korzystać z obiektów graficznych ROOT-a, muszą być spełnione następujące warunki:

- deklaracja klasy, reprezentująca obiekt graficzny, powinna znajdować się w pliku nagłówkowym, a definicja w pliku programu.
- w pliku nagłówkowym załączamy dekarację:

  ```
  #include<RQ_OBJECT.h>
  ```

- na początku deklaracji klasy umieszczamy linię:

  ```
  RQ_OBJECT("NazwaKlasy")
  ```

a na końcu:

```
ClassDef(NazwaKlasy,0)
```

Słowo „NazwaKlasy” zastępujemy nazwą deklarowanej klasy.

Instancje tej klasy będą traktowane jako obiekty graficzne. Jeśli nie chcemy od podstaw konstruować nowej klasy obiektu, możemy dziedziczyć po już istniejących klasach, np.: *TGMainFrame*, *TGTransientFrame*.

Słownik

Przed przystąpieniem do kompilacji programu należy utworzyć słownik. Będzie on zawierał informacje o wszystkich klasach korzystających z graficznego interfejsu ROOT-a oraz powiązania między nimi np. sygnały.

W tym celu powinniśmy w katalogu projektu programu utworzyć plik o nazwie *LinkDef.h*. Następnie, w nowo utworzonym pliku, od nowej linii wpisujemy według poniższego schematu informacje dla preprocesora o wszystkich nazwach klas korzystających z ROOT-a:

```
#pragma link C++ class NazwaKlasy;
```

gdzie słowo „NazwaKlasy” zastępujemy odpowiednią nazwą klasy.

Pozostaje nam jedynie wygenerowanie pliku słownika *Dictionary.cpp*, co czynimy w następujący sposób:

```
rootcint -f Dictionary.cpp -c LISTA LinkDef.h
```

W powyższym schemacie w miejsce słowa „LISTA” wstawiamy listę plików nagłówkowych korzystających z ROOT-a.

Kompilacja

Podczas wstępnej kompilacji plików programu (o rozszerzeniu „cpp”) do plików obiektów, musimy przekazać kompilatorowi informacje o flagach i ścieżkach do plików nagłówkowych ROOT-a. Najprościej jest to zrobić korzystając z polecenia:

```
‘root-config --cflags‘
```

Przy tworzeniu programu wynikowego musimy również wskazać odpowiednie biblioteki dla ROOT-a, które zostaną do niego dołączone, co najprościej uzyskać przez:

```
‘root-config --cflags --glibs‘
```

2.2. Biblioteka *Libconfig*

Biblioteka *Libconfig* jest darmową i otwartą biblioteką do obsługi plików konfiguracyjnych. Umożliwia zarówno odczytywanie jak i modyfikację ustawień z pliku. Jest napisana w dwóch wersjach: dla języka programowania *C* oraz pokrewnego mu obiektowego języka programowania *C++*. My skupimy się tylko i wyłącznie na wersji dla *C++*, gdyż w tym języku powstał program będący przedmiotem tej pracy. Nie jest to również dokładny opis wszystkich możliwości biblioteki. W pracy zostaną poruszone jedynie niezbędne elementy, z których skorzystano przy tworzeniu programu *test2K2Kgui*. Szczegółowy opis biblioteki *Libconfig* można znaleźć na stronie głównej projektu [12].

Użycie biblioteki w *C++*

By móc skorzystać z biblioteki *Libconfig*, w programie napisanym w języku *C++*, należy w pliku źródłowym programu dodać dyrektywę preprocesora:

```
#include <libconfig.h++>
```

lub alternatywnie:

```
#include <libconfig.hh>
```

Dla skrócenia nazw wywoływanych funkcji i poprawienia czytelności kodu, dobrym pomysłem jest również dodanie przestrzeni nazw *libconfig*:

```
using namespace libconfig;
```

Podczas kompilacji programu należy także poinformować konsolidator o tym, że chcemy skorzystać z zewnętrznej biblioteki poprzez przekazanie opcji "-lconfig++".

Składnia pliku konfiguracyjnego

Opcja w pliku konfiguracyjnym ma następującą składnię:

```
nazwa = wartość;
```

lub alternatywanie:

```
nazwa : wartość;
```

Gdzie nazwa może składać się z dużych i małych liter alfabetu łacińskiego oraz cyfr. Wartość natomiast może być typu: int, float, string, bool i jest zapisywana zgodnie ze składnią języka *C++*.

W pliku konfiguracyjnym można także korzystać z komentarzy, których składnia pochodzi z języka *C++*. Dodatkowo wprowadzono jeszcze jeden sposób oznaczania komentarza, ważny od znaku *hash* ("#") do końca linii.

Przykładowy plik konfiguracyjny znajduje się w punkcie 3.2.

Wyjątki

Biblioteka o nieoczekiwanych zdarzeniach informuje poprzez rzucanie wyjątków. Oto ich lista:

- SettingTypeException
  wyjątek rzucany gdy zmienna na którą próbujemy wczytać wartość jest innego typu niż ten z pliku konfiguracyjnego.
- SettingNotFoundException
  wyjątek rzucany gdy nie znaleziono szukanej opcji.
- SettingNameException
  wyjątek rzucany gdy próbujemy utworzyć opcję o identycznej nazwie, jak już istniejąca opcja.
- ParseException
  wyjątek rzucany gdy w pliku konfiguracyjnym wystąpił błąd składniowy.
- FileIOException
  wyjątek rzucany gdy wystąpią problemy podczas pisania/czytania z pliku konfiguracyjnego.

Klasą nadrzędną dla wszystkich wymienionych wyjątków jest *ConfigException*.

Kompilacja oraz instalacja biblioteki

Aby zbudować bibliotekę *libconfig*, należy pobrać jej najnowszą wersję z głównej strony projektu [12] lub skorzystać z zamieszczonej na płycie CD dołączonej do pracy – plik *libconfig-1.3.tar.gz*. Spakowaną bibliotekę umieszczamy w katalogu */opt/pi/ext/dload/* i rozpakowujemy poleceniem:

```
tar -zxvf libconfig-1.3.tar.gz
```

Następnie w kolejności wykonujemy:

```
cd /opt/pi/ext/dload/libconfig-1.3
./configure --prefix=/opt/pi/ext/
make
sudo make install
```

co spowoduje zbudowanie i zainstalowanie biblioteki w katalogu */opt/pi/ext/*. Wskazany folder jest miejscem, w którym są umieszczane wszystkie zewnętrzne biblioteki oraz programy wykorzystywane w eksperymencie „Pi of the Sky".

Przykład użycia

Dla czytelności kodu część odpowiedzialna za przechwytywanie wyjątków została pominięta. Przykład odczytu opcji *fits_viewer_name*:

```
Config cfg;
cfg.readFile(pathConfig);

if (cfg.exists("fits_viewer_name")) {
    Setting& sname = cfg.lookup("fits_viewer_name");
    const char* name = sname;
    strcpy(fitsViewer,name);
    printf("fits_viewer_name: %s\n",name);
}
```

Przykład modyfikacji wartości opcji *fits_viewer_name*:

```
Config cfg;
cfg.readFile(pathConfig);

if (cfg.exists("fits_viewer_name")) {
    Setting& sname = cfg.lookup("fits_viewer_name");
    sname = fitsViewer;
    cfg.writeFile(pathConfig);
    printf("Update configuration: fits_viewer_name = %s\n",fitsViewer);
}
else {
    printf("Config Error: fits_viewer_name not exist\n");
    return false;
}
```

2.3. Flex i Bison

Jednym z ważniejszych założeń, które miał zrealizować *test2K2Kgui* było wykonywanie skryptów umożliwiających automatyczne testowanie kamer CCD. By spełnić ten warunek, powstał prosty składniowo język skryptowy, pozwalający wykonywać najważniejsze operacje na urządzeniu. Jego szczegółowy opis znajduje się w punkcie 3.4.

Od razu założyłem, że cześć programu odpowiedzialna za analizę języka skryptowego będzie tworzona przy pomocy odpowiednich narzędzi. Było ku temu kilka przesłanek. Wykonanie własnego analizatora składniowego jest pracą żmudną i czasochłonną. Podczas jego tworzenia można popełnić wiele błędów, które mogą ujawnić się dopiero po jakimś czasie. Automatyczne generatory kodu minimalizują ten problem i pozwalają skupić się na zadaniu. Kolejnym powodem było to, iż składnia i ilość operacji w nowo powstającym języku skryptowym mogła podlegać zmianom. Duża ilość poprawek w kodzie może skutkować brakiem jego czytelności, czego konsekwencją mogą być oczywiście błędy. Jak się później okazało, zmiany w języku skryptowym miały miejsce, co potwierdziło wcześniejsze przypuszczenia.

Do stworzenia kodu programu potrafiącego zrozumieć wspomniany już język użyłem dwóch popularnych narzędzi linuksowych. Pierwszym z nich jest *Flex* [13], program który generuje analizatory leksykalne, czyli programy potrafiące rozpoznawać w tekście: zarezerwowane słowa, napisy, liczby, operatory. Drugi to *Bison* [14], generator analizatorów składniowych, zwanych również parserami. Programy tego typu dokonują analizy składniowej – sprawdzają, czy przewarzany plik został zapisany zgodnie z regułami pewnej gramatyki. Ich wynikiem jest najczęściej pewna struktura, którą można później w łatwy sposób przetwarzać w programie. W przypadku *Test2K2Kgui* tą strukturą jest lista obiektów zawierająca informacje o kolejnych operacjach, które należy wykonać. Każdy taki obiekt, zwany w liście węzłem, jest instancją klasy *ScriptNode*. Zawiera on informacje o typie akcji do wykonania, jej parametrach oraz wskaźnik do następnego węzła.

Rozdział 3

Program test2K2Kgui

Przed przystąpieniem do prac mających na celu stworzenie graficznego programu do obsługi kamer CCD postanowiłem dowiedzieć się, jak członkowie zespołu „Pi of the Sky" oceniają dotychczasowy program z interfejsem tekstowym do obsługi tych urządzeń – *test2K2K*. Poprosiłem więc promotora mojej pracy o wysłanie do osób obsługujących kamery e-mali, w moim imieniu, z zapytaniem o wady, zalety aktualnego programu, listę najczęściej wykonywanych operacji na kamerze oraz propozycje nowych funkcji. Poniżej przedstawiam listę uwag jakie otrzymałem od członków zespołu „Pi of the Sky":

- Największy problem użytkownikom programu *test2K2K* sprawia mało przyjazny interfejs użytkownika. Wykonanie dowolnej akcji polega na wpisaniu jej numeru z listy wszystkich możliwych. Jednak spis poleceń jest tak długi, że nie mieści się na ekranie monitora i potrzebne jest przewijanie zawartości konsoli by zobaczyć akcje z początku listy.

- Jeśli jakaś akcja na kamerze została wykonana, jej wynik był wypisywany na ekranie monitora i od razu po nim następowała lista wszystkich akcji. By obejrzeć wynik ponownie potrzebne było przewinięcie zawartości konsoli.

- Jako wadę programu podano również brak możliwości oglądania na bieżąco podstawowych parametrów kamery, takich jak temperatura. By uzyskać ten efekt użytkownik musiał wielokrotnie wybierać interesującą go akcję.

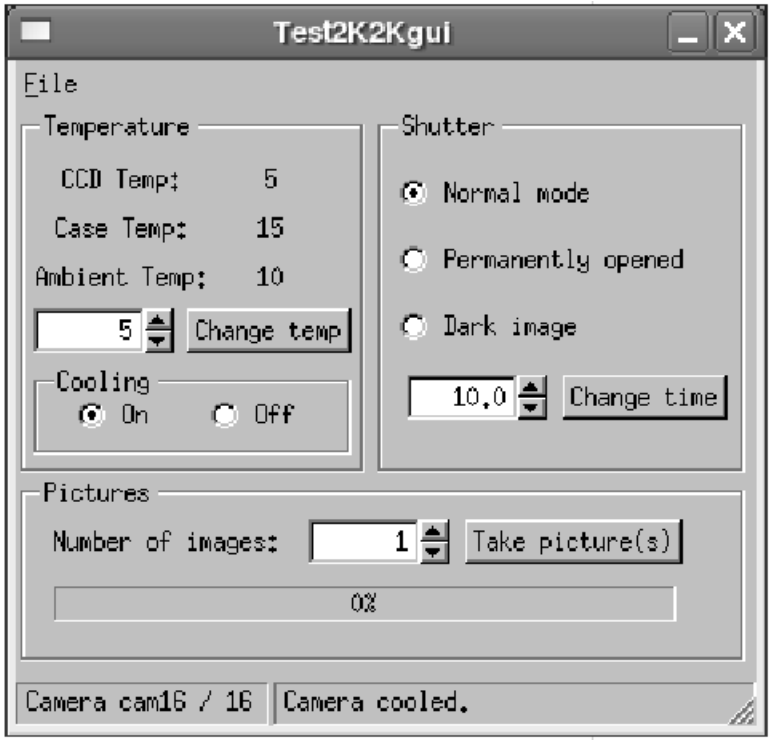
- Zaproponowano nową funkcję, której nie posiadał *test2K2K*: możliwość automatycznego testowania kamer poprzez wykonywanie skryptów.

Po analizie powyższych uwag i propozycji doszedłem do następujących wniosków:

- Program *test2K2K* ma możliwość podawania pewnych parametrów z linii komend. Szanując przyzwyczajenia użytkowników postanowiłem, że graficzny program również będzie dawał tę możliwość, zachowując zgodność co do nazw parametrów – w miarę możliwości. Również nazwa tworzonego programu *test2K2Kgui* miała przekonać użytkowników, że jest to ulepszona wersja dobrze im znanego programu.

- Przesłane listy najczęściej wykonywanych akcji pozwoliły na wyodrębnienie operacji i informacji, do których dostęp powinien być prosty i szybki.

- Zaakceptowałem pomysł automatycznego testowania kamer i podjąłem się jego rozwiązania.



Rysunek 3.1: Wygląd okna głównego programu test2K2Kgui

- Zaakceptowałem pomysł wyświetlania na bieżąco najważniejszych parametrów kamery.
- Zaproponowałem stworzenie pliku konfiguracyjnego dla programu. Pozwoli on na zapamiętanie wartości parametrów podawanych z linii komend, podczas uruchomienia program.

Przy tworzeniu *test2K2Kgui* bardzo pomocny okazał się symulator kamery CCD – *nudpsim* [15]. Powstał on na potrzeby projektu „Pi of the Sky" i imituje działanie urządzenia komunikującego się przez gigabitowy interfejs Ethernet. Dzięki *nudpsim* mogłem pracować nad programem w domu, a co najważniejsze nie musiałem przejmować się, że ewentualne błędy w aplikacji mogą uszkodzić kamerę.

3.1. Opis programu

Każdy z poniższych punktów opisuje jedno okno programu *test2K2Kgui* lub jedną ważną jego funkcję. Wszystkie okna aplikacji składają się z ramek, w których zostały zgrupowane podobne operacje wykonywane na kamerze.

Okno główne

Główne okno programu *test2K2Kgui* (rys. 3.1) prezentuje podstawowe informacje o kamerze oraz najczęściej używane operacje wykonywane na urządzeniu. Dla czytelności, podobne akcje zostały umieszczone blisko siebie i zgrupowane w ramki. Poniżej znajduje się opis tychże ramek:

- ramka *Temperature*
  Wyświetla informacje związane z temperaturą kamery:

  - *CCD Temp* – temperatura matrycy CCD
  - *Case Temp* – temperatura wewnątrz obudowy kamery
  - *Ambient Temp* – temperatura otoczenia

  Wyżej wymienione wartości są aktualizowane przez aplikację co 5 sekund.

  Użytkownik może również nakazać kamerze zmianę jej aktualnej temperatury. W tym celu należy wprowadzić wartość żądanej temperatury (w stopniach Celsjusza) do okna edycyjnego i zatwierdzić akcję przyciskiem *change temp*. Jeśli chłodzenie kamery jest wyłączone, ukaże się komunikat proponujący jego włączenie.

  Jeśli kamera z różnych powodów nie może osiągnąć żądanej przez użytkownika temperatury, zostanie on o tym poinformowany komunikatem: *Warning: Camera not cooling!* W przypadku gdy wskazana temperatura zostanie osiągnięta, otworzy się okno dialogowe z informacją: *Camera cooled*.

  Ostatnią opcją w ramce jest możliwość włączenia/wyłączenia wspomnianego już chłodzenia kamery.

- ramka *Shutter*
  W ramce *Shutter* zostały zawarte wszystkie operacje związane z migawką kamery. Jedną z takich akcji jest zmiana czasu otwarcia migawki. Aby tego dokonać, należy w polu edycyjnym wprowadzić żądaną wartość czasu otwarcia przesłony, wyrażoną w sekundach i potwierdzić zmianę przyciskiem *Change time*. Możliwa jest również zmiana sposobu pracy migawki. Do wyboru są trzy tryby:

  - *Permanently opened* – migawka podczas wykonywania zdjęcia jest cały czas otwarta
  - *Normal mode* – migawka podczas wykonywania zdjęcia jest otwierana na określony czas
  - *Dark image* – migawka podczas wykonywania zdjęcia jest cały czas zamknięta. Tak zwana ciemna klatka służy do kalibracji pomiarów i wyznaczania poziomu szumów

- ramka *Pictures*
  Pozwala na wykonywanie zdjęć kamerą CCD. Użytkownik w polu edycyjnym wprowadza liczbę zdjęć jaką chce wykonać i zatwierdza akcję wybierając przycisk *Take picture(s)*. Kamera zaczyna pracę informując jednocześnie użytkownika o przebiegu działań na pasku postępu. Możliwe jest, lecz nie wskazane wykonywanie w tym momencie innych działań związanych z kamerą. Gdy zostaną wykonane wszystkie zdjęcia, pasek postępu pokaże wartość 100% a na pasku stanu ukaże się stosowna informacja o zakończeniu akcji. Wszystkie wykonane zdjęcia zostaną zapisane w formecie FITS (ang. *Flexible Image Transport System*) [19] i umieszczone w katalogu wskazanym w oknie opcji.

- Menu
  Z menu okna głównego uruchamiamy programy zewnętrzne i otwieramy inne okna *test2K2Kgui* zawierające bardzje zaawansowane akcje związane z kamerą. Szczegółowy opis okien i odpowiadających im opcji z menu znajduje się w dalszej części pracy.

- pasek stanu
  Pasek stanu jest podzielony na dwie części. W pierwszej (lewej) znajduje się nazwa kamery CCD z którą pracujemy oraz jej numer identyfikacyjny. W prawej części paska stanu ukazują się informacje o postępie akcji wykonywanych w oknie głównym.

Rysunek 3.2: Okno opcji

Okno opcji

Dzięki oknu opcji (rys. 3.2) użytkownik może w prosty sposób edytować podstawowe ustawienia programu. Wszystkie wartości pól z tego okna są wczytywane z pliku konfiguracyjnego aplikacji (zob. punkt 3.2). Wszelkie zmiany wartości w tym oknie są zapisywane od razu w konfiguracji i przy ponownym uruchomieniu *test2K2Kgui* są uznawane za domyślne.

Okno opcji otwieramy wybierając z menu okna głównego (zob. punkt 3.1) pole *Options*. Poniżej znajduje się opis wszystkich elementów okna opcji.

- ramka *fits directory*
  Wyświetla nazwę katalogu, w którym zostaną zapisane zdjęcia wykonane kamerą. Po wciśnięciu przez użytkownika przycisku *change* otworzy się okno dialogowe, w którym można wybrać nowy katalog do zapisywania zdjęć.

- ramka *Fits viewer program*
  Pozwala na wybór programu, który będzie wyświetlał zdjęcia wykonane kamerą po wybraniu opcji *Show last FITS*. W polu edycyjnym należy wpisać nazwę przeglądarki obsługującej format *fit* i zatwierdzić zmianę przyciskiem enter, bądź wciskając przycisk *change*. Ważne jest by wybrany program potrafił obsługiwać pliki podane jako parametr programu oraz by jego katalog znajdował się w zmiennej środowiskowej linuksa *PATH*.

- ramka *Fits compression*
  Umożliwia wybór sposobu kompresji zdjęć wykonywanych przez kamerę. Do wyboru są dwie wartości:

  – None – brak kompresji zdjęć

– ASAS – kompresja zdjęć ASAS

- ramka *Script output*
  Pozwala użytkownikowi na wybór pliku, w którym zostanie zapisany dziennik zdarzeń, czyli wszystkie akcje, które zaszły podczas wykonywania skryptów (zob. punkt 3.4). Jeśli plik nie został określony, domyślnie stanie się nim *script.out* i zostanie utworzony w katalogu skąd uruchomiono program.

- ramka *Cooling dialog*
  Po osiągnięciu przez kamerę żądanej temperatury ustawionej przez użytkownika lub gdy urządzenie nie może jej osiągnąć, jesteśmy o tym informowani stosownym komunikatem w oknie dialogowym. W tej ramce możemy zadecydować, czy chcemy by to okno się wyświetlało, bądź nie.

- pasek stanu
  Wyświetla informacje o postępie, ukończeniu, bądź błędach akcji wykonywanych w oknie opcji

Okno kluczy/wartości

Jak już wspomniano, zdjęcia wykonywane kamerą CCD są zapisywane w formacie graficznym FITS. Jedną z ciekawszych opcji tego formatu jest możliwość zapisywania dodatkowych informacji w kodzie ASCII. Pliki FITS posiadają nagłówek, który pozwala na przechowywanie w nim par: klucz=wartość. Znaczenie klucza i wartości jest ograniczone jedynie fantazją osoby wykonującej zdjęcia. Typowym przykładem może być para: ID_CAM=16. Większość kluczy jest wypełniana automatycznie przez sterownik kamery, na podstawie jej ustawień.

Aby dodać klucze i wartości , opisujące zdjęcia wykonane w ramach danego testu kamery, należy z menu wybrać opcję *Fits keys/values*. Poniżej znajduje się opis ramek znajdujących się w oknie kluczy/wartości (rys. 3.3):

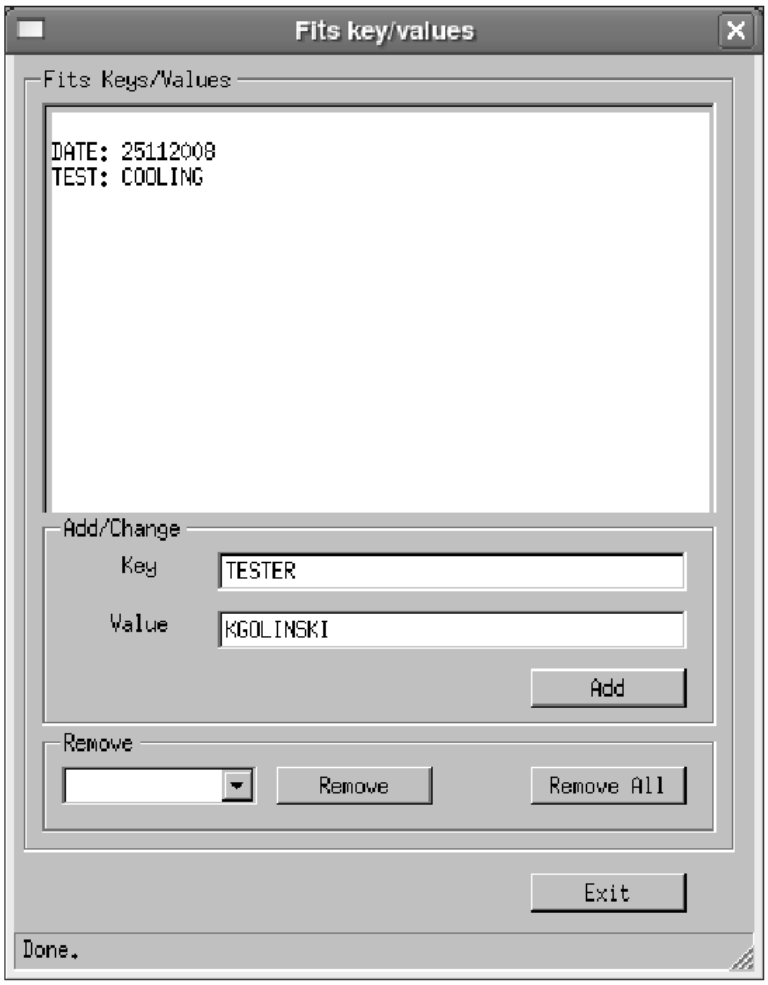
- ramka *Add/Change*
  Zawiera dwa pola tekstowe: *key* i *value*. W pierwszym z nich wpisujemy nazwę klucza, w drugim jego wartość. Wypełnienie pola *value* jest opcjonalne. Aby dodać nową parę wystarczy w jednym z okienek wcisnąć przycisk enter, bądź wybrać przycisk *Add*. Nowo dodana para ukaże się w oknie tekstowym. Jeśli nazwa klucza jest identyczna z nazwą klucza już istniejącego, nowa wartość zostaje zapisana w miejsce starej.

- ramka *Remove*
  Pozwala na usunięcie poszczególnych par klucz/wartość, bądź wszystkich. W pierwszym przypadku wybieramy z listy wszystkich kluczy nazwę klucza, którą chcemy usunąć i wybieramy przycisk *Remove*. Jeśli chcemy usunąć wszystkie pary wciskamy przycisk *Remove All*. Przed akcją skasowania wszystkich par zostaniemy ostrzeżeni komunikatem: *Warning: Do you want remove ALL fits keys/values?*

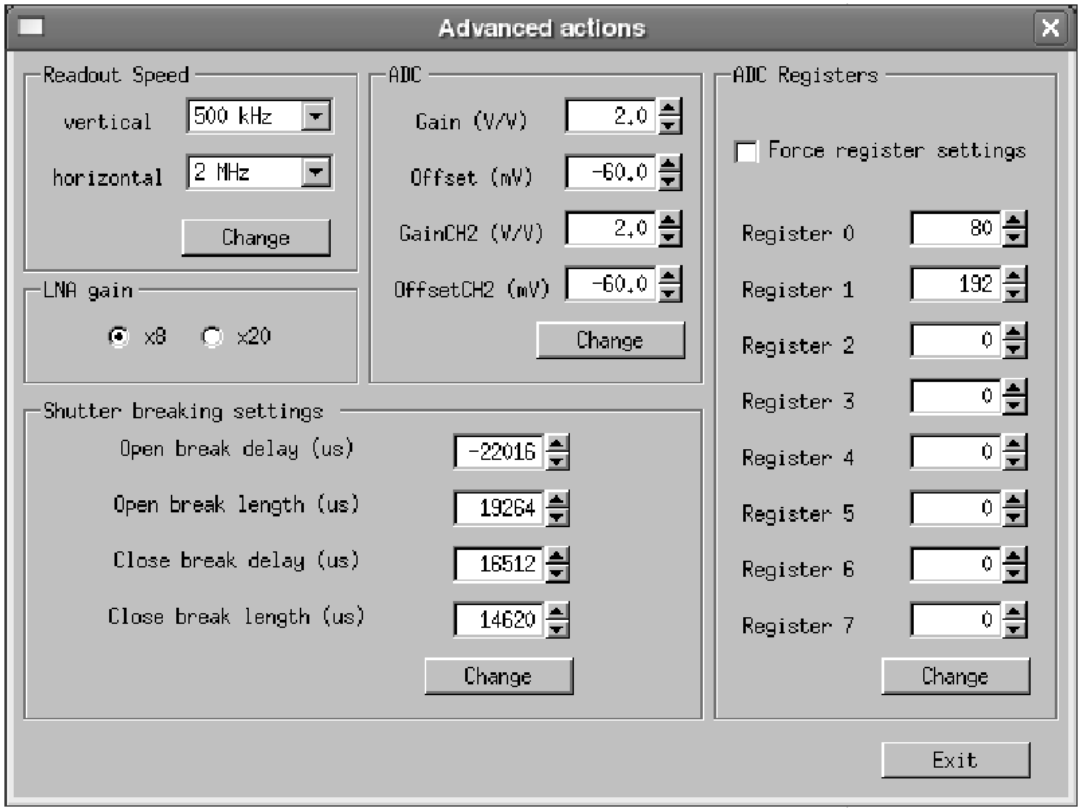
Należy zwrócić uwagę na to, iż klucze/wartości ustawione w tym oknie nie są dodawane do zdjęć wykonywanych podczas automatycznego testowania (zob. punkt 3.4).

Okno zaawansowanych akcji

Okno zaawansowanych akcji (rys. 3.4) zawiera opcje pozwalające zmienić wartości wewnętrznych ustawień poszczególnych podzespołów kamery CCD. Dla wygody użytkownika odpowiednie wartości są przeliczane i wypisywane w jednostkach fizycznych. Okno zaawansowa-



Rysunek 3.3: Okno kluczy/wartości



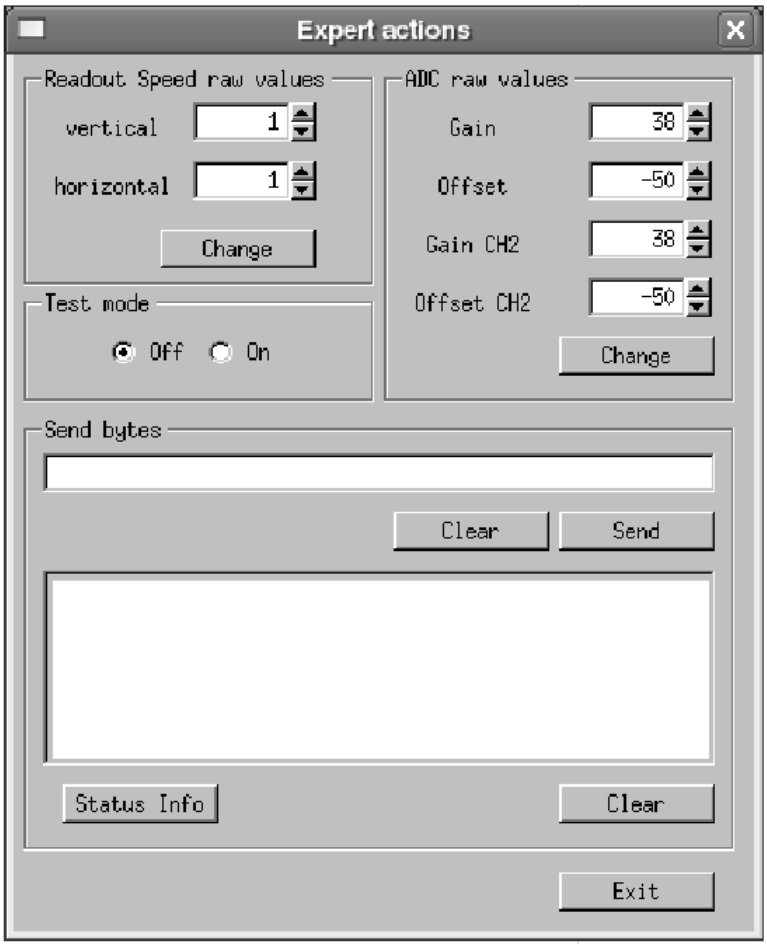
Rysunek 3.4: Okno zaawansowanych akcji

nych akcji otwieramy wybierając z menu okna głównego opcję *Advanced actions*. Poniżej znajduje się opis wszystkich ramek z tego okna:

- ramka *Readout Speed*
  Pozwala na ustawienie szybkości odczytu danych z matrycy CCD
- ramka *ADC*
  Umożliwia zmianę ustawień przetwornika analogowo-cyfrowego, aby dokonywać korekty otrzymywanych zdjęć.
- ramka *ADC Registers*
  Pozwala na bezpośrednią zmianę wartości rejestrów przetwornika analogowo-cyfrowego.
- ramka *LNA gain*
  Daje możliwość zmiany wartości wzmocnienia przedwzmacniacza.
- ramka *Shutter breaking settings*
  Umożliwia zmiany prędkości otwierania, zamykania przesłony kamery. Dzięki umiejętnemu sterowaniu tymi wartościami, możliwe jest przedłużenie żywotności działania migawki.

Okno eksperta

Dla najbardziej zaawansowanych i wymagających użytkowników kamer CCD powstało okno eksperta (rys. 3.5). Wszystkie wartości w nim wyświetlane są pobierane bezpośrednio ze sterownika urządzenia i nie są poddawane żadnej analizie, na przykład przeliczaniu jednostek.



Rysunek 3.5: Okno eksperta

Okno eksperta otwieramy wybierając z menu okna głównego opcję *Expert actions*. Poniżej znajduje się opis ramek z tego okna oraz informacje o akcjach, które można w nich wykonać:

- ramka *Send bytes*
  Pozwala na wysyłanie bezpośrednio do sterownika kamery instrukcji sterujących w postaci ciągu bajtów. Aby wysłać instrukcję do kamery, jej treść wpisujemy w oknie edycyjnym i wciskamy enter lub wybieramy przycisk *Send*. Zwrotna odpowiedź od urządzenia zostanie wypisana w oknie tekstowym. Wartości wysyłane do kamery mogą być podawane w trzech różnych systemach liczbowych:

  – dziesiętnym, np.: 16
  – ósemkowym (oktagonalnym), np.: 020
  – szesnastkowym (hexadecymalnym), np.: $0x10$

  Jeśli chcemy obejrzeć aktualne ustawienia kamery wybieramy przycisk *Status info*. Urządzenie pobierze na nowo informacje ze wszystkich swoich sensorów i wynik operacji będziemy mogli obejrzeć w oknie tekstowym.

- ramka *Test mode*
  Umożliwia włączanie/wyłączanie testowego trybu pracy kamery, w którym urządzenie zwraca obraz o znanej i dobrze określonej postaci. Umożliwia to np. badanie błędów transmisji.

- ramka *Readout speed raw values*
  Pozwala na zmianę prędkości odczytu z układu CCD do pamięci wewnętrznej kamery.

- ramka *ADC raw values*
  Daje możliwość sterowania ustawieniami przetwornika analogowo-cyfrowego.

Programy zewnętrzne

Dla wygody użytkowników *test2K2Kgui* pozwala na prostą współpracę z programami zewnętrznymi.

- Przeglądarka plików FITS
  Po pobraniu zdjęcia z kamery CCD w menu okna głównego uaktywnia się opcja *Show last fits*. Wybranie jej powoduje otwarcie przeglądarki plików FITS zadeklarowanej w oknie opcji (zob. punkt 3.1). Zostanie ona uruchomiona z parametrem będącym ścieżką do ostatnio wykonanego zdjęcia. Domyślnym programem wyświetlającym zdjęcia jest *ds9* [17].

  Dokonując wyboru przeglądarki plików FITS należy zwrócić uwagę, jak program zachowuje się w przypadku otwierania zdjęć bez kompresji i z kompresją ASAS. Domyślnie ustawiona przeglądarka nie potrafi obsłużyć plików poddanych kompresji.

- *CCDToolkit*
  Program *CCDToolkit* [18] autorstwa Marcina Molaka i Janusza Użyckiego powstał na potrzeby eksperymentu „Pi of tke Sky". Umożliwia on analizę obrazów z kamery, potrafi między innymi: tworzyć histogramy, próbkować i obliczać transformatę Fouriera.

Zmiana oprogramowania wbudowanego w kamerę CCD

Kamery CCD wykorzystywane w eksperymencie „Pi of the Sky" pozwalają na programową zmianę wbudowanego oprogramowania. Jest to bardzo wygodna metoda wprowadzania nowych opcji oraz poprawek w sterowniku, nie wymagająca wykonywania fizycznych operacji przy urządzeniu.

Aby zmienić wbudowane oprogramowanie, należy z menu okna głównego programu wybrać podmenu *Firmware*. Następnie należy określić, który sterownik chcemy zmienić. Do wyboru są dwa: *Change Cypress firmware* – odpowiedzialny za transmisję danych przez interfejs USB i *Change Altera firmware* – sterownik odpowiedzialny za odczyt matrycy CCD. Po dokonaniu wyboru otworzy się okno, w którym wskazujemy plik zawierający nowe oprogramowanie. Przed zatwierdzeniem ostatecznej akcji zmiany sterownika, zostaniemy ostrzeżeni następującym komunikatem:

```
Warning: You try load new firmware for Cypress.
Are you REALLY SURE?
```

Zmiana oprogramowania kamery CCD potencjalnie może być ryzykowna, jeśli podejmie się jej osoba niepowołana. Zastosowanie nieodpowiedniego sterownika, bądź zamazanie sterego przypadkowymi danymi, może spowodować nieprzewidziane skutki w działaniu urządzenia. Dlatego przed zmianą oprogramowania kamery należy się upewnić, czy posiadamy kopię zapasową aktualnie działającego sterownika.

3.2. Plik konfiguracyjny

Program *Test2K2Kgui* pozwala użytkownikom na zapamiętywanie podstawowych i najważniejszych ustawień w pliku konfiguracyjnym. Dzięki niemu pracująca z programem osoba nie musi żmudnie, wielokrotnie wprowadzać tych samych danych. Zastosowanie pliku konfiguracyjnego umożliwia również proste przeniesienie konfiguracji na inny komputer. Wartości ustawione w pliku konfiguracyjnym mogą być przesłaniane przez parametry programu podawane z linii poleceń (zob. punkt 3.3). Dla wygody użytkownika powstało również okno opcji (zob. punkt 3.1), dzięki któremu wprost z interfejsu graficznego można zmieniać wartości podstawowych ustawień.

Plik konfiguracyjny programu powinien znajdować się w katalogu *∼/.pisoft* i nosić nazwę *test2K2Kgui.conf*. Jeśli aplikacja nie znajdzie go we wskazanym powyżej miejscu, automatycznie zostanie utworzony nowy plik o odpowiedniej nazwie. Jego zawartość, jako przykład, została podana na końcu punktu.

Poniżej znajduje się lista wszystkich opcji, które można ustawić w pliku konfiguracyjnym, wraz z ich objaśnieniem. Obok każdej opcji w nawiasie znajduje się informacja o typie wartości jaką ona przyjmuje.

- path_fits_directory (string)
  Pełna ścieżka do katalogu, w którym będą zapisywane zdjęcia pobrane z kamery. Możliwe jest również skorzystanie ze skróconej nazwy aktualnego katalogu: "./". Jeśli opcja nie jest ustawiona, zdjęcie zostanie zapisane do katalogu, w którym został uruchomiony program.

- camera_type (string)
  Rodzaj kamery, z którą program nawiązuje połączenie.
  Możliwe wartości:

  - "usb" – dla urządzenia bezpośrednio podłączonego do portu usb danego komputera
  - "eth" – dla kamery, z którą program łączy się za pomocą sieci komputerowej

  Jeśli opcja nie jest ustawiona, domyślnym urządzeniem jest kamera usb.

- eth_camera_ip (string)
  Adres IPv4 kamery jeśli urządzenie z którym się łączymy jest typu sieciowego. Jeśli korzystamy z kamery usb opcja jest pomijana.

- eth_camera_port (int)
  Port pod którym nasłuchuje kamera sieciowa. Jeśli korzystamy z urządzenia komunikującego się przez port usb, to opcja jest pomijana.

- init_params (bool)
  Możliwe wartości:

  - true – wewnętrze parametry kamery zostaną ustawione z plików konfiguracyjnych sterownika: *ccd.cfg, device.cfg*
  - false – kamera nie jest inicjowana wartościami z wyżej wymienionych plików konfiguracyjnych sterownika

  Domyślna wartość: false.

- fits_viewer_name (string)
  Nazwa programu wyświetlającego zdjęcia w formacie *fit*. Katalog, w którym znajduje się program, musi należeć do zmiennej środowiskowej linuksa – *PATH*. Przeglądarka zdjęć musi także obsługiwać nazwy plików do wyświetlenia podane jako parametr programu.

- fits_compression (string)
  Rodzaj kompresji jaka zostanie zastosowana przy tworzeniu zdjęć. Możliwe wartości to:

  - "eFITSComprNone" – brak kompresji
  - "eFITSComprASAS" – kompresja ASAS

- path_script_output (string)
  Ścieżka wskazująca miejsce zapisu dziennika zdarzeń – wszystkich akcji wykonywanych podczas automatycznego testowania kamery CCD (zob. punkt 3.4). Domyślnie opcja wskazuje na plik *script.out*. Jest on tworzony w katalogu, w którym został uruchomiony program *test2K2Kgui*. Jeśli plik o wskazanej nazwie już istniał, nowe informacje będą dopisywane na jego końcu.

- show_cooling_dialog (bool)
  Opcja pozwalająca włączyć/wyłączyć okno dialogowe informujące o chłodzeniu kamery. Domyślna wartość – true – powoduje wyświetlanie okna w programie.

Poniżej znajduje się przykładowy plik konfiguracyjny programu *test2K2Kgui*. Każda z opcji powinna znajdować się w osobnej linii zakończonej średnikiem. Istnieje również możliwość wprowadzania komentarzy do pliku konfiguracyjnego. Komentarz rozpoczyna znak *hash* ("#") i trwa do końca wiersza.

```
path_fits_directory = "/home/users/krzysztof/";
camera_type = "eth";
```

```
eth_camera_ip = "127.0.0.1";
eth_camera_port = 1234;
init_params = true;
fits_viewer_name = "ds9";
# uwaga: duze rozmiary zdjec
fits_compression = "eFITSComprNone";
path_script_output = "script.out";
show_cooling_dialog = true;
```

Do obsługi pliku konfiguracyjnego wykorzystano bibliotekę *Libconfig*. Szczegółowy, techniczny opis jak to zrealizowano w *test2K2Kgui* można przeczytać w punkcie 2.2.

3.3. Parametry programu

Parametry programu, podobnie jak plik konfiguracyjny (zob. punkt 3.2), pozwalają na rozpoczęcie pracy z kamerą przygotowaną wedle preferencji użytkownika.

Jeśli chcemy skorzystać z możliwości przekazania parametrów do programu *test2K2Kgui*, pierwszym z nich powinna być nazwa kamery, na przykład:

```
test2K2Kgui kam1
```

Następnie, w dowolnej kolejności, mogą być użyte parametry z poniższej listy:

- -usb
  Wskazujemy ten parametr gdy kamera, z którą pracujemy, komunikuje się za pomocą interfejsu USB. Urządzenie musi być widoczne w systemie Linux jako */dev/usb/pikamN*, gdzie N jest liczbą od 0 do 15.

- -eth=<adres IP:Port>
  Oznacza, że należy połączyć się z kamerą sieciową o wskazanym adresie IP i porcie, na przykład:

  ```
  -eth=127.0.0.1:1234
  ```

- -compr
  Zdjęcia wykonywane przez kamerę są poddawane kompresji ASAS [19].

- -nocompr
  Zdjęcia wykonywane przez kamerę nie będą poddawane kompresji.

- -init_params
  Wewnętrze ustawienia kamery są inicjowane wartościami z pliku sterownika.

- -outdir=<ścieżka>
  Pozwala na ustawienie katalogu, w którym będą zapisywane zdjęcia wykonane przez kamerę. Przykład użycia:

  ```
  -outdir=/home/users/myhomedir/images/
  ```

- -save_to_conf
  Wartości parametrów ustawione z linii poleceń są zapisywane do pliku konfiguracyjnego programu.

- -help
  Wypisuje w powłoce linuksa informacje na temat parametrów przyjmowanych przez program i kończy działanie aplikacji.

- -verb
  W powłoce linuksa są wypisywane szczegółowe informacje na temat aktualnych akcji wykonywanych przez program.

- -outscript=<ścieżka>
  Pozwala na ustawienie ścieżki oraz nazwy pliku, w którym zostanie zapisany dziennik zdarzeń (zob. punkt 3.4). Przykład użycia:

  ```
  -outscript=/home/users/myhome/script.out
  ```

- -local_ip=<adres IP>
  W przypadku gdy komputer, z którego korzystamy posiada więcej niż jedną kartę sieciową, możemy wskazać adres podsieci przez którą należy wysyłać komunikaty do kamery, na przykład:

  ```
  -local_ip=192.168.1.1
  ```

- -local_port=<numer portu>
  Analogicznie do poprzedniej opcji, wskazujemy numer portu w odpowiedniej podsieci, na przykład:

  ```
  -local_port=10000
  ```

Poniżej znajduje się przykładowe wywołanie programu z kilkoma parametrami:

```
test2K2Kgui kam1 -eth=192.168.1.2:1234 -outscript=script.out -save_to_conf
```

Nakazujemy w nim programowi połączyć się z kamerą, korzystającą z gigabitowego interfejsu Ethernet, dostępną pod adresem IP: *192.168.1.2* oraz portem: *1234*. Urządzenie ma otrzymać nazwę *kam1*, a dziennik zdarzeń automatycznych testów zostanie zapisany do pliku *script.out* w katalogu, skąd uruchomiono program. Dzięki skorzystaniu z *-save_to_conf* wszystkie parametry wywołania programu zostaną zapisane do pliku konfiguracyjnego aplikacji.

Ważną cechą parametrów podawanych z linii komend jest to, iż są one ustawiane po przeczytaniu pliku konfiguracyjnego. Umożliwia to przesłanianie uprzednio ustawionych wartości. Tę własność można wykorzystać do szybkiego testowania kamery, gdy jednorazowo chcemy sprawdzić, jak zachowa się urządzenie z nowymi parametrami.

3.4. Automatyzacja testowania kamer CCD

Wielokrotne powtarzanie tych samych czynności podczas procedury testowania kamery bywa nużące. Osoba obsługująca kamerę może przeoczyć, bądź zapomnieć o niektórych zadaniach do wykonania. Wtedy często należy całą procedurę sprawdzania urządzenia zacząć od nowa.

Biorąc pod uwagę powyższe fakty, jednym z ważniejszych zadań jakie ma realizować program *test2K2Kgui* jest automatyzacja testowania kamer CCD. Daje ona wiele korzyści. Zapewnia, że wszystkie testy wykonywane na różnych kamerach będą przeprowadzane dokładnie w ten sam sposób. Pozwala również jednej osobie na realizację kilku testów w tym samym

momencie. Wystarczy włączyć skrypty testujące na kilku komputerach i cierpliwie czekać na wyniki.

Pierwszym krokiem do realizacji automatycznego testowania kamer było stworzenie nieskomplikowanego składniowo języka skryptowego. Jego prostota wynika z faktu, iż skrypty w tym języku będą tworzone, czytane i modyfikowane przez osoby niekoniecznie zaznajomione z językami programowania. Na podstawie listy najczęściej wykonywanych operacji testowych wskazanych przez członków zespołu „Pi of the Sky" zaproponowałem składnię języka skryptowego. Otrzymałem w ten sposób listę operacji oraz prostą i elegancką propozycję ich reprezentacji. Do stworzenia analizatora leksykalnego i składniowego skorzystałem z dwóch narzędzi: *Flex* i *Bison* (zob. punkt 2.3).

Kolejnym wymaganiem było, by wszystkie akcje wykonywane w skrypcie oraz ich wyniki były zapisywane do pliku. W tym celu został stworzony dziennik zdarzeń. Odnotowywane są w nim, oprócz już wspomnianych akcji i ich wyników, informacje o dacie i godzinie rozpoczęcia/zakończenia wykonywania skryptu. Trafiają do niego również dodatkowe uwagi w postaci komentarzy, pozwalające namierzyć ewentualne błędy w skryptach. Wyniki uruchomienia nowego pliku z testami są zapisywane zawsze na końcu dziennika zdarzeń. Nie musimy więc za każdym razem kopiować wyników by ich nie utracić. Wystarczy uruchomić grupę testów i po wykonaniu wszystkich obejrzeć dziennik zdarzeń.

Miejsce zapisu i nazwę dziennika dziennika zdarzeń można ustalić w oknie opcji.

Tworzenie skryptu

Nazwa pliku skryptu powinna mieć rozszerzenie „sc". Każda operacja powinna znajdować się w osobnej linii. Przykładowy skrypt i jego dziennik zdarzeń znajduje się w dodatku, punkt A i B.
Akcje związane z pobieraniem zdjęć:

- take_n_pictures(integer n)
  pobranie n zdjęć z kamery. Liczba n powinna być liczba naturalną, $n > 0$. Po pobraniu każdego zdjęcia w dzienniku zdażeń zapisywana jest pełna ścieżka pobranego pliku.
  Przykład:

  ```
  take_n_pictures(5)
  ```

- set_fits_dir(string s)
  s jest pełną ścieżką do katalogu w którym chcemy zapisać zdjęcia. Jeśli podany katalog nie istnieje, w dzienniku zdarzeń zostanie zapisana stosowna informacja a pliki zdjęcia trafią do katalogu określonego w konfiguracji.
  Przykład:

  ```
  set_fits_dir("/home/users/myhomedir/pictures/")
  ```

- set_fits_key(string key, string value)
  dodaje do pliku FITS klucz *key* o wartości *value*. Jeśli podany klucz już istnieje, zmienia jego wartość. Po wykonaniu operacji do dziennika zdarzeń zapisywane są wszystkie ustawione klucze i ich wartości.
  Przykład:

  ```
  set_fits_key("TESTTYPE","COOL_TEST")
  ```

- clear_fits_keys
  usuwa wszystkie, dodatkowo zdefiniowane przez użytkownika, klucze FITS dodane poleceniem *set_fits_key*

Akcje związane z temperaturą:

- wait_for_temp(integer n)
  ustawia temperaturę CCD na wartość n°C i czeka aż zostanie ona osiągnięta. Co 5 sekund zapisuje do dziennika zdarzeń informacje o aktualnej temperaturze. Zobacz również *set_temp*
  Przykład:

  ```
  wait_for_temp(-5)
  ```

- set_temp(integer n)
  wysyła do kamery prośbę o ustawienie temperatury CCD na wartość n°C. W przeciwieństwie do *wait_for_temp* nie czeka aż zostanie osiągnięta żądana temperatura.
  Przykład:

  ```
  set_temp(0)
  ```

- set_cooling_on
  Włącza chłodzenie kamery. Zobacz również *set_cooling*

- set_cooling_off
  Wyłącza chłodzenie kamery. Zobacz również *set_cooling*

- set_cooling(integer n)
  Dla $n = 1$ włącza chłodzenie kamery; $n = 0$ – wyłącza. Zobacz również: *set_cooling_on, set_cooling_off*

Akcje związane z migawką:

- set_shutter_time(integer n)
  Ustawia czas otwarcia migawki na n sekund.
  Przykład:

  ```
  set_shutter_time(5)
  ```

- shutter_normal_node
  ustawia tryb normalny pracy migawki. Zobacz również: *shutter_permanently_opened, dark_image*

- shutter_permanently_opened
  migawka jest cały czas otwarta. Zobacz również: *shutter_normal_node,dark_image*

- dark_image
  migawka podczas wykonywania zdjęć jest zamknięta. Zobacz również: *shutter_normal_node, shutter_permanently_opened*

Pozostałe akcje:

- get_status
  Zapisuje do dziennika zdarzeń informacje o stanie kamery: temperaturę CCD, obudowy kamery, otoczenia oraz chłodzenia.

- system(string s)
  Wykonuje polecenie powłoki systemu operacyjnego Linux, s definiuje polecenie. Przykład:

  ```
  system("mv /images/*.fit /images/old/")
  ```

- sleep(integer n)
  nie wykonuje żadnych czynności przez n sekund. Wartość zmiennej n powinna być liczbą naturalną, $n > 0$.
  Przykład:

  ```
  sleep(10)
  ```

- #
  komentarz rozpoczyna znak hash ("#") i jest ważny do końca linii.
  Przykład:

  ```
  sleep(3600) # czekaj 1 godzine
  ```

Kilka dobrych rad odnośnie tworzenia skryptów

Skrypt powinien rozpoczynać się od komentarza wyjaśniającego czego on dotyczy i jaki jest jego cel. W przyszłości ułatwi to rozpoznawanie podobnych, aczkolwiek różnych plików bez czytania zawartych w nim wszystkich akcji.

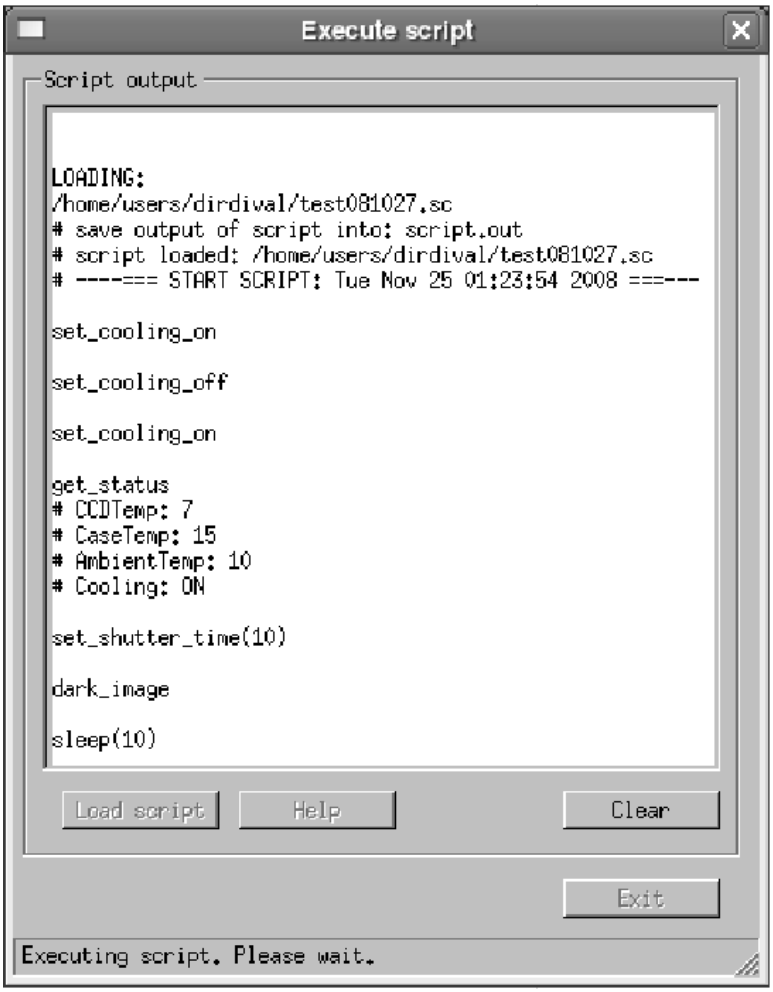
Nie należy nigdy zakładać, że przed rozpoczęciem procedury testowania kamera znajduje się w jakimś konkretnym stanie. Przed wykonaniem zdjęć powinniśmy ustawić: tryb pracy migawki i czas jej otwarcia, wybrać tryb chłodzenia, jak również temperaturę CCD oraz ustalić katalog, do którego będą zapisywane zdjęcia.

Jeśli korzystamy z opcji dodawania kluczy i wartości do plików FITS, powinniśmy grupować w spójne bloki wywołania akcji *set_fits_key*. Zdecydowanie zwiększy to czytelność skryptu. Dobrym pomysłem wydaje się również rozpoczęcie każdego takiego bloku od wyczyszczenia wszystkich kluczy. Dzięki temu unikniemy sytuacji, w których stare wartości z innych bloków trafią do aktualnie tworzonego.

Korzystając z akcji *system* powinniśmy pamiętać aby zawsze stosować pełne ścieżki do plików, katalogów. Nigdy nie zakładajmy, że program został uruchomiony z konkretnego katalogu. Wypada również wspomnieć, iż akcja *system* nie daje wszystkich możliwości powłoki z Linuksa. Nie możemy na przykład przekierować strumienia wyjściowego polecenia do innego polecenia lub komendy. Również w dzienniku zdarzeń nie zobaczymy wyniku akcji. Zostanie w nim jedynie informacja o operacji, którą wykonaliśmy.

W przypadku, gdy z jakichś powodów utracimy skrypt, możemy spróbować odzyskać go z dziennika zdarzeń. W tym celu odszukujemy w dzienniku interesujący nas fragment i kopiujemy do nowego pliku. W zasadzie w tym momencie moglibyśmy już poprzestać, właśnie odzyskaliśmy skrypt. Dziennik zdarzeń został tak pomyślany, by wszystkie akcje w nim zapisane były w pełni zgodne ze składnią języka skryptu. Tak odzyskany plik może zawierać dużą liczbę komentarzy generowanych przez różne akcje. Jeśli chcemy przywrócić czytelność skryptu wystarczy je usunąć, co możemy uczynić wykonując:

```
$ cat odsyskany.sc | grep -v -e "#" > bezkomentarzy.sc
```



Rysunek 3.6: Okno *Execute Script* podczas wykonywania skryptu

Wykonanie skryptu

Aby wykonać skrypt z menu okna głównego należy wybrać opcję *Execute script*. Następnie w oknie, które się otworzy, wybieramy przycisk *Load* i w oknie wyboru plików zaznaczamy żądany skrypt. Jeśli plik, który wybraliśmy, jest poprawnym skryptem to zacznie się on wykonywać. Postępy jego wykonania będą widoczne w oknie tekstowym, w którym zostaną zapisane kolejno wykonywane akcje i ich rezultaty. Równolegle wszystkie informacje będą także zapisywane do dziennika zdarzeń.

Jeśli plik który wybraliśmy zawierał błędy składniowe, ujrzymy poniższy komunikat:

```
# syntax error in script:
# (line: 10) = take_npictures(3)
```

Komunikat poinformuje nas o linii w skrypcie, w której popełniliśmy błąd składniowy oraz wypisze jej zawartość. Natomiast, jeśli spróbujemy otworzyć plik do którego nie mamy uprawnień odczytu, ujrzymy następujący komunikat:

```
# you have no access privileges to file
```

W oknie *Execute Script* (rys. 3.6) znajduje się również prosta pomoc dotycząca akcji w skryptach. Wystarczy wybrać przycisk *help* a w oknie tekstowym zostanie wypisana lista wszystkich

akcji, wraz z ich opisem i przykładami użycia.

3.5. Opis instalacji programu

Aplikacja *test2K2Kgui* była tworzona i testowana na dwóch dystrybucjach linuksa: *Fedora* [7] i *PLD* [20]. Poniżej znajduje się lista wymaganych programów, narzędzi do kompilacji i poprawnego działania *test2K2Kgui*:

- program *make*, wersja 3.80 lub nowsza
- kompilator *g++*, wersja 3.3.6 lub nowsza
- program *flex*, wersja 2.5.31 lub nowsza
- program *bison*, wersja 2.1 lub nowsza
- biblioteka *libconfig*, wersja 1.3 lub nowsza
- system *ROOT*, wersja 5.20/00 lub nowsza
- biblioteka *libpicamdriver.a* – sterownik do kamer CCD wykonany przez zespół „Pi of the Sky"

Programy: *make, g++, flex* oraz *bison* powinny znajdować się w repozytoriach wspomnianych już dystrybucji linuksa. *ROOT* i *libconfig* można pobrać z głównych stron ich projektów: [11],[12]. Opis instalacji biblioteki *libconfig* znajduje się w punkcie 2.2. Niezbędna jest również biblioteka *libpicamdriver.a* – zawierająca sterownik do obsługi kamer CCD. Powinna ona znajdować się w katalogu */opt/pi/dev/pisys/daq/src/standalone/driver*.

Po upewnieniu się, że wszystkie wymagane programy i biblioteki znajdują się w systemie, możemy przystąpić do budowy i instalacji programu *test2K2Kgui*. W tym celu kopiujemy z płyty CD, dołączonej do pracy, katalog *test2K2Kgui* do folderu */opt/pi/dev/pisys/daq/-src/standalone/driver*. Następnie w katalogu budowanego programu wykonujemy:

```
make all
```

Jeśli kompilacja przebiegnie bezbłędnie, ujrzymy poniższy komunikat:

```
COMPILATION test2K2Kgui COMPLETED
```

Aby dokończyć proces instalacji programu wystarczy wydać polecenie:

```
make install
```

Spowoduje ono, że program *test2K2Kgui* znajdzie się w katalogu */opt/pi/dev/pisys/daq/-ndir/bin*.

Rozdział 4

Podsumowanie

W momencie powstawania pracy (koniec roku 2008) zespół „Pi of the Sky” przygotowuje się do kolejnej fazy eksperymentu, trwa budowana 32 nowych kamer CCD. Zanim trafią do obserwatorium w Chile muszą być dokładnie przetestowane. Zadaniem programu *test2K2Kgui* jest ułatwienie tej procedury. Członkowie zespołu będą mogli w wygodny sposób sprawdzić poprawne działanie kamery. Podgląd wszystkich istotnych parametrów urządzenia oraz możliwość ich modyfikacji, może spowodować szybsze wykrycie ewentualnych błędów sprzętu. Opcja automatycznego testowania kamer sprawi, że wszystkie testy przeprowadzane będą dokładnie w ten sam sposób. Dzięki temu zostanie wyeliminowany ludzki czynnik błędu z procedury badania sprzętu. Powinno przełożyć się to na bardziej miarodajne wyniki testów kamer. Dodatkowo, automatyczne testowanie urządzeń może być przeprowadzane przez mniejszą liczbę osób. Nie jest już potrzebna obecność osoby badającej kamerę przy każdym konkretnym jej egzemplarzu. Wystarczy uruchomić jednocześnie na kilku komputerach skrypty testujące urządzenia, a następnie porównać ich wyniki. Pozwoli to również na zmniejszenie czasu badania sprzętu, bądź przeprowadzenie większej ilości testów.

Ponieważ do budowy programu *test2K2Kgui* wykorzystano symulator kamery, przed złożeniem poniższej pracy, przeprowadzono testy aplikacji z prawdziwym urządzeniem korzystającym z interfejsu USB (rys. 4.1). Wykonane badanie programu nie wykazało żadnych niezgodności między wyświetlanymi informacjami przez aplikację, a ustawieniami kamery CCD.

Rysunek 4.1: Testy programu z kamerą CCD

Podziękowania

Chciałbym serdecznie podziękować mojemu promotorowi dr Marcinowi Sokołowskiemu, który z wielką cierpliwością odpowiadał na wszystkie pytania związane z funkcjonowaniem kamer CCD.

Dodatek A

Przykładowy skrypt do testowania kamer CCD

```
# opis testu:
# chlodzenie wlaczane, temperatura CCD = -10
# czas otwarcia przeslony 10 sekund
# katalog dla fitsow: /home/users/dirdival/fits/
#
# Wykonujemy po 2 zdjecia dla kazdego trybu migawki:
# tryb normalny, migawka otwarta i zamknieta

set_cooling_on
set_shutter_time(10)
set_fits_dir("/home/users/dirdival/fits/")

# usuwamy wszystkie stare zdjecia z katalogu
system("rm -f /home/users/dirdival/fits/*.fit")

# tryb migawki: normalny

shutter_normal_mode
wait_for_temp(-10)

clear_fits_keys
set_fits_key("MIGAWKA","NORMALNA")

take_n_pictures(2)

# tryb migawki: otwarta caly czas

shutter_permanently_opened
wait_for_temp(-10)

clear_fits_keys
set_fits_key("MIGAWKA","OTWARTA")
```

```
take_n_pictures(2)

# tryb migawki: zamknieta

dark_image
wait_for_temp(-10)

clear_fits_keys
set_fits_key("MIGAWKA","ZAMKNIETA")

take_n_pictures(2)
```

Dodatek B

Dziennik zdarzeń przykładowego skryptu

```
# save output of script into: /home/users/dirdival/script.out
# script loaded: /home/users/dirdival/test5.sc
# ---=== START SCRIPT: Tue Nov  4 02:17:04 2008 ===---

set_cooling_on
set_shutter_time(10)

set_fits_dir("/home/users/dirdival/fits/")
# Check directory: exist
# directory changed

system("rm -f /home/users/dirdival/fits/*.fit")
shutter_normal_mode

wait_for_temp(-10)
# wait_for_temp(-10)
# CCDTemp: -9
# CaseTemp: 15
# AmbientTemp: 10
# Cooling: ON

# wait_for_temp(-10)
# CCDTemp: -10
# CaseTemp: 15
# AmbientTemp: 10
# Cooling: ON

clear_fits_keys

set_fits_key("MIGAWKA","NORMALNA")
# all keys/values:
# MIGAWKA: NORMALNA
```

```
take_n_pictures(2)
# 1: /home/users/dirdival/fits//cam16_081103_00001.fit
# 2: /home/users/dirdival/fits//cam16_081103_00002.fit

shutter_permanently_opened

# wait_for_temp(-10)
# CCDTemp: -10
# CaseTemp: 15
# AmbientTemp: 10
# Cooling: ON

clear_fits_keys

set_fits_key("MIGAWKA","OTWARTA")
# all keys/values:
# MIGAWKA: OTWARTA

take_n_pictures(2)
# 1: /home/users/dirdival/fits//cam16_081103_00003.fit
# 2: /home/users/dirdival/fits//cam16_081103_00004.fit

dark_image

# wait_for_temp(-10)
# CCDTemp: -10
# CaseTemp: 15
# AmbientTemp: 10
# Cooling: ON

clear_fits_keys

set_fits_key("MIGAWKA","ZAMKNIETA")
# all keys/values:
# MIGAWKA: ZAMKNIETA

take_n_pictures(2)
# 1: /home/users/dirdival/fits//DARK/cam16_081103_00005_d.fit
# 2: /home/users/dirdival/fits//DARK/cam16_081103_00006_d.fit
# -------- STOP SCRIPT: Tue Nov  4 02:28:53 2008 ===---
```

Bibliografia

[1] Marcin Sokołowski, *Investigation of astrophysical phenomena in short time scales with „Pi of the Sky" apparatus*, rozprawa doktorska, http://arxiv.org/abs/0810.1179

[2] Strona główna eksperymentu „Pi of the Sky", http://grb.fuw.edu.pl

[3] Marek Biskup, *Poszukiwanie gwiazd zmiennych w eksperymencie „Pi of the Sky"*; punkt 1.1 – *Błyski gamma*, praca dyplomowa, http://www.mimuw.edu.pl/~mbiskup/documents/MarekBiskup-Fizyka-mgr.pdf

[4] Strona NASA poświęcona satelicie Swift, http://heasarc.gsfc.nasa.gov/docs/swift

[5] Strona NASA poświęcona sieci GCN, http://gcn.gsfc.nasa.gov

[6] Szczegółowy opis rozbłysku GRB080319B, http://grb.fuw.edu.pl/pi/ot/grb080319b/normal.html

[7] Strona główna dystrybucji linuksa *Fedora*, http://fedoraproject.org

[8] G. Kasprowicz, H. Czyrkowski, R. Dąbrowski, W. Dominik, L. Mankiewicz, K. Poźniak, R. Domaniuk, P. Sitek, M. Sokołowski, R. Sulej, J. Użycki, G. Wrochna, *New low noise CCD cameras for „Pi of the Sky" project*, Proceedings of SPIE Vol.6347, 2006, http://spiedigitallibrary.aip.org/dbt/dbt.jsp?KEY=PSISDG&Volume=6347&Issue=1&bproc=volrange&scode=6300+-+6399#MAJOR3

[9] Strona główna protokołu *NUDP*, http://grb.fuw.edu.pl/pi/user/juzycki

[10] Strona główna biblioteki *Sockets*, http://www.alhem.net/Sockets

[11] Strona główna systemu *ROOT*, http://root.cern.ch

[12] Strona główna biblioteki *libconfig*, http://www.hyperrealm.com/libconfig/libconfig.html

[13] Strona główna programu *Flex*, http://flex.sourceforge.net

[14] Strona główna programu *Bison*, http://www.gnu.org/software/bison

[15] Strona główna programu *nudpsim*, http://grb.fuw.edu.pl/pi/user/juzycki

[16] Strona główna formatu *Flexible Image Transport System (FITS)*, http://fits.gsfc.nasa.gov

[17] Strona główna programu *ds9*, http://hea-www.harvard.edu/RD/ds9

[18] Strona główna programu *CCDToolkit*,
http://mazepi.fuw.edu.pl/~mmolak/pl/Ccd/index

[19] Strona główna projektu *The All Sky Automated Survey (ASAS)*,
http://www.astrouw.edu.pl/asas

[20] Strona główna dystrybucji linuksa *PLD*, http://pld-linux.org