USING CEMES CCD CAMERA WITH AUDELA

- **General and specific camera functions**

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

CEMES

Table of contents

1. Introduction 3

1.1. Contents 3
1.2. Supported Operating Systems 3
1.3. System Requirements 3
1.4. Installation 3

2. Audela 4

2.1. Software for astronomic proposals 4
2.2. Why Audela 4
2.3. Audela achievements 4
2.4. Audela software structure 5

3. Building a camera driver for Audela 6

3.1. Procedure diagrams 6

4. Camera's hardware 10

4.1. Camera's characteristics 10
4.2. Camera's performance 11
4.3. Acquiring modes 14
4.4. Side effects 14

5. Audela commands 15

5.1. General functions 15
5.2. Specific functions 16
5.3. Usage examples 19
5.4. Calibrating the camera 20
5.5. Images captured and the limiting magnitude 22

6. Driver code 25

6.1. camera.c 27
6.2. camtcl.c (functions) 42
6.3. camera.h 49
6.4. camtcl.h 51

1. Introduction

1.1 Contents

This document provides a description of all the Audela Cemes CCD camera functions as well as their way of use. It also contains some characteristics of the camera itself.

1.2 Supported operating systems

Although there's also a linux version of Audela, at the moment the CEMES CCD camera driver for Audela is only suitable for all the Windows versions since Windows 95.

1.3 System requirements

The minimum requirements for Audela to run properly are:

- Windows 95/98/ME/2000/XP
- PC Pentium 75Mhz
- Free disk space: 50 Mb
- RAM memory: 16 Mb RAM
- 16 bits 800x600 Colour screen

1.4 Installation

In order to successfully install the CEMES camera in Audela one should install the CEMES files in the corresponding folders:

- The folder *libcemes*, which has the CEMES camera's driving files, must be placed in folder ...*audela**src**libcam*\;
- The folder *cemes*, which has the CEMES camera's external dependencies, must be placed in folder ...*audela**src**external*\. This includes files with extension .h and .lib. Every time these dependency files are changed, they must be placed in the corresponding folder and then one must run the *install.bat* application.

2. Audela

About the CEMES CCD camera's software, it can be said that it was written according to some specifications that should be accomplished to help the good interaction between the software and the program that will use this software to drive the camera, the Audela 1.40. This program, the Audela, it's an interface software that allows image acquiring, visualize processed images and command robotic telescopes. In the meantime, only the first two capabilities of this program will be important, acquire and visualization of an image.

2.1 Software for astronomical proposals

Like in all electronic digital devices, the information gathered in CCD has to be processed by means of software. Some software has been developed with the purpose to be a useful tool to the Astronomy and Astrophysics. Such software permits the user to control the exposure time of the CCD, the temperature of the cooling system, set the size of the image by choosing its binning, change contrast or brightness settings, apply filters and many other functions that might be specific to each camera. Some software available nowadays permits to control a few different types of cameras, having the specific controls to each camera, avoiding the need of using one program for each different camera.

2.2 Why Audela

- Audela program is completely configurable by the user, even its interface.
- Audela is totally polyvalent; it drives the camera and a telescope even remotely.
- It may be used by a beginner/amateur astronomer (interface buttons) or by an expert/professional astronomer (command line console). In our case interface buttons will not be used, only the console.
- Audela is already translated to 5 languages other than the French, they are the English, Spanish, Italian, German and soon the Danish.
- GNU open source code that can be improved by students or researchers.

2.3 Audela achievements

- With Audela, Robin Chassagne discovered 15 supernovae and Alain Klotz discovered 2.
- Audela is being used in the Pic du Midi observatory in the French Pyrenees for the visualization of images obtained with the Bernard Lyot, 2 meter telescope.
- Audela is used in the cloud monitoring of the Paranal observatory.
- Audela is used to schedule observations and process images at the TAROT observatory.

2.4 Audela's software structure

Although there are great benefits of using Audela, the programmer that wants to improve the program (like developing a new library as libcemes) should respect some rules not to be disappointed with the final work. There are 3 categories of rules that should be accomplished:

- Rules about its general structure and the files read in the script
- Rules about the way to use the general variables like
- Rules about the way the translations should be made

It can be said that more than create a great library for astronomical applications, Audela creators have divided voluntary the program in extensions in order to gain in terms of modularity. To drive a specific CCD camera, the Audela program needs the corresponding dll file from each camera. To compile such a dll file there must be a file with the characteristics of the camera (camera.c in this case), the functions that shall be used to control the camera and the corresponding command list to be used in Audela's console.

In order to drive a specific CCD camera, the Audela program needs the corresponding dll file from each camera. To compile such a dll file there must be a file with the characteristics of the camera (camera.c) and the functions that shall be used to control the camera (camera.c and camtcl.c).

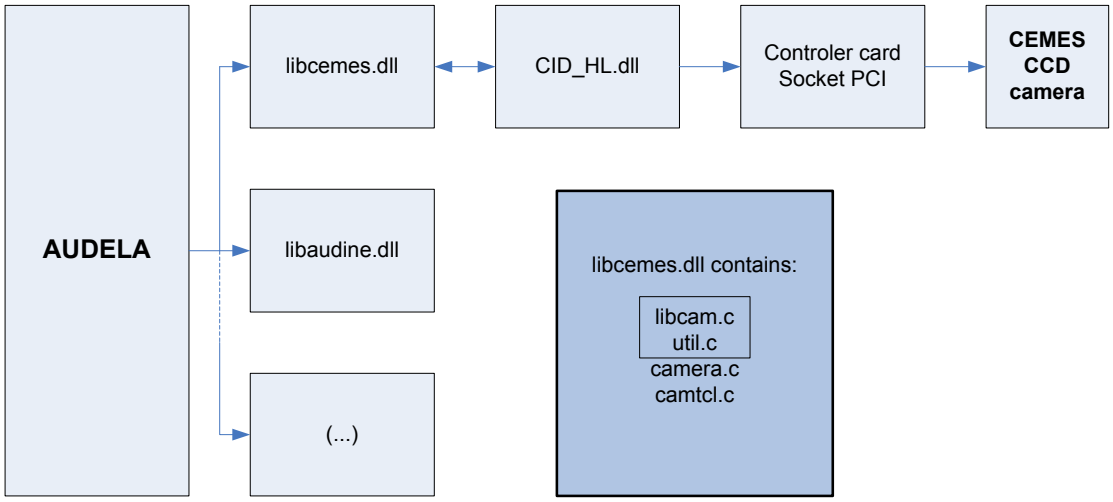

Figure 1: Overview on software structure

As shown in figure 1, Audela has a library of .dll files, one for each specific camera. Each camera has itself a group of .c files: libcam.c, util.c, camera.c and camtcl.c. While libcam.c and util.c are common to every camera, **camera.c** and **camtcl.c** have the specific functions for each camera. The functions in libcemes.dll interact with a lower level library, CID_HL.dll, which was already developed by the constructors of the camera. Software communicates with the camera by a PCI socket.

Note: In camera.c there is the standard code to drive the camera and the acquiring procedure. The specific functions to each camera concerning the balance of the image or other procedures that only exist in a particular camera are in the camtcl.c file.

CAMERA.C

The most important file to drive a camera is camera.c. This file contains the functions for driving the camera. These functions are the ones that should be modified from a camera to another, although not changing the header of the functions, which should remain from a camera to another. Even in case of not needing to use none of the functions defined on camera.c, one should clean its body but leave the header of the function as it is. In the beginning of camera.c it was defined the main characteristics of the camera in structure camini, then it was written the driving and acquiring functions of the camera. This functions can also be called in camtcl.c. In the following pages there are some diagrams to show the main procedures of the camera.c functions.

CAMTCL.C

Camtcl.c is the file that has the interface functions between the Tcl interpreter and the C language, specifics to each camera. The functions within this file can be directly called in the Audela program, not with their header name (from camtcl.c file) but as they are defined in structure SPECIFIC_CMDLIST in camtcl.h. By this, one can understand that this file is used anytime a professional astronomer uses the console (command line window) of the Audela program, where he can directly call functions related with the acquiring of an image, the amplifier, the shutter and many other important capabilities of the camera. It's in this file that new functions should be created. To add a new function, its code must be written with a header like int cmdCamFunction(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]), the name of the function "Cam" will be replaced by the camera's name and "Function" by the name of the function. Furthermore, the header must be declared in camtcl.h and the respective Audela's command line name added in SPECIFIC_CMDLIST structure. The functions created for this camera as "setwin" (sets a smaller window display) or "balance" (balances brightness level of the 4 zones of the ccd) are defined in this file. In camtcl.c one can also fill a global structure (cam) with all the parameters of the camera (see function "parameter"), which will be used by functions in camera.c.

3. Building a camera driver for Audela

When building a new camera driver for Audela, one should start to build the camera.c functions. This means that one should write the code of the 2 major functions, the acquiring functions cam_start_exp and cam_read_ccd, the code of the function to stop acquiring function, cam_close and firstly the function to start up the power supply and the camera itself, cam_init. Having this code written, the user is already capable of taking a picture, despite having no other specific functions, which should be wtitten in the camtcl.c file (cooling or balancing functions as an example).

3.1 Procedure diagrams

Below, it will be showed the procedure diagram of the functions cam_close, cam_init, and the two major (acquiring) functions of the driver, cam_start_exp and cam_read_ccd.

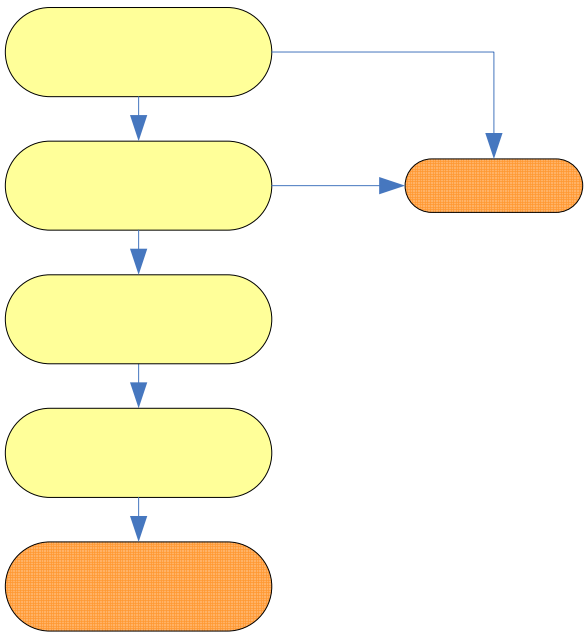

Diagram 1: cam_close function procedure

cam_close

cam_init

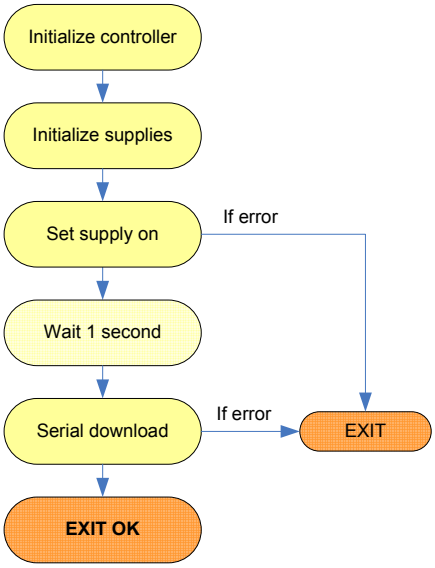

Diagram 2: cam_init function procedure

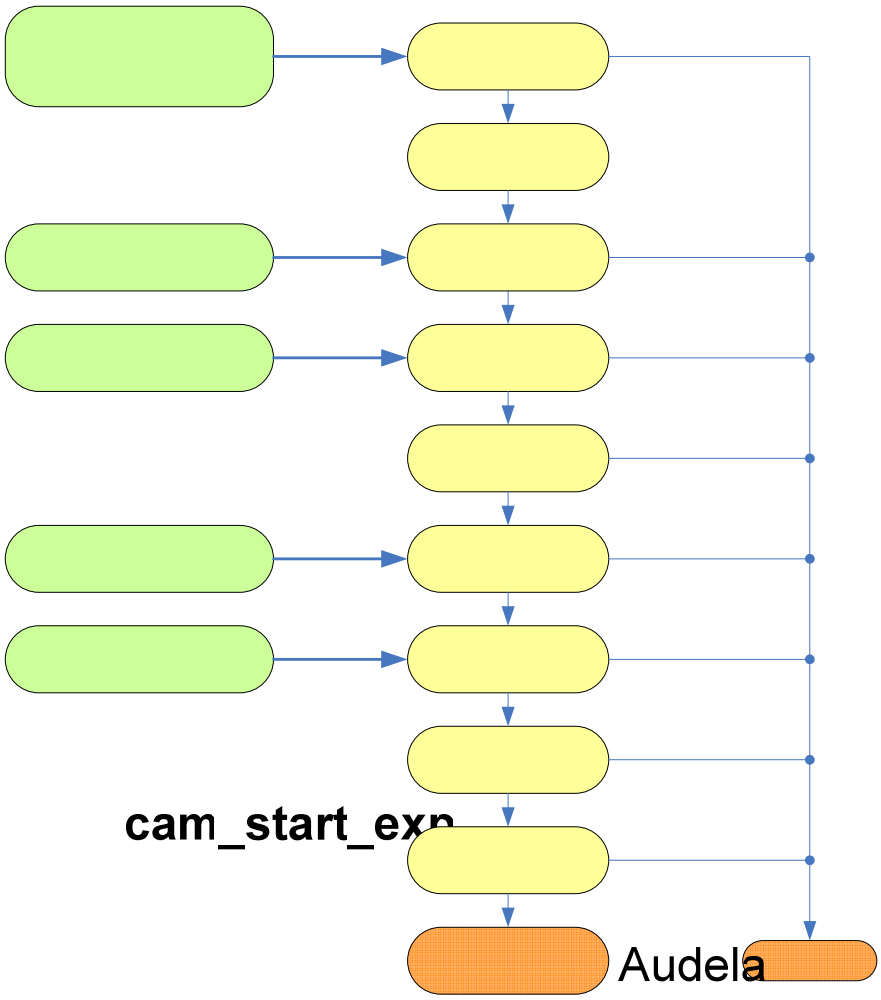

Diagram 3: cam_start_exp function procedure

Binning: 1x1, 2x2, 4x4, 8x8
HS: On/Off
Speed: 1, 2, 3, 4
Debug: On/Off

cam_read_ccd

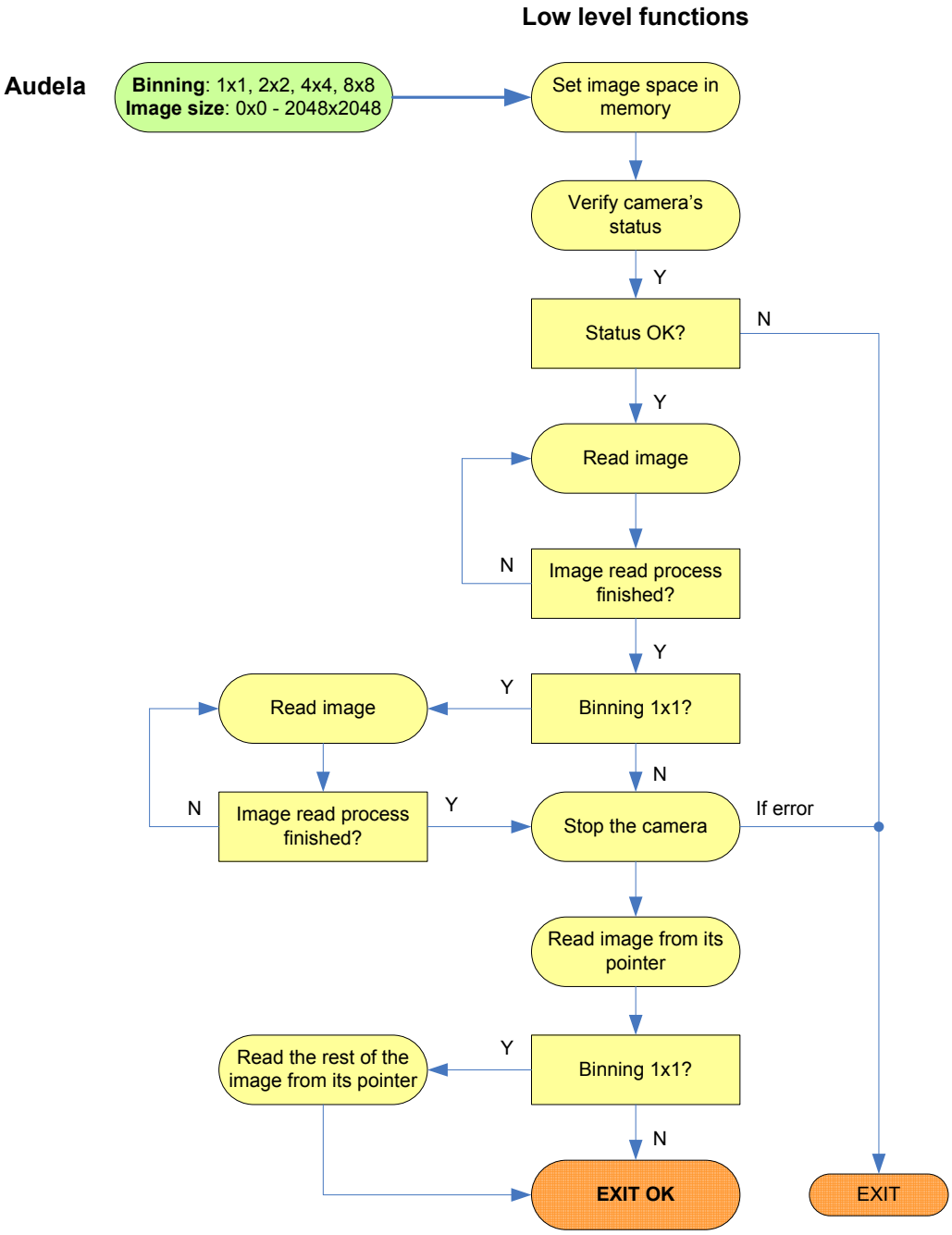

Diagram 4: cam_read_ccd function procedure

4. Camera's Hardware

4.1 Camera's characteristics

The CEMES CCD camera is a high sensibility numerical CCD camera for use in electronic microscopy and astronomy. The CCD is supplied by a power supply which is controlled by the PC via a RS232 connection optically coupled. The CCD is cooled by a cooling liquid provided by a cooling supply and by a 3 stage peltier that allows the camera to achieve temperatures as low as -55ºC. The Interface board communicates with the camera by optical fibre.

The CCD is a 2048x2048 full frame Charge Couple Device designed for a wide range of applications due to both its operating mode flexibility and its high dynamic range combined with its high resolution. The nominal photosensitive area is made of 2048x2048 useful pixels splited vertically in 4 zones of 520 lines each.

The matrix has also:

- 7 columns of dark reference, formed by photo-elements that are not exposed to photons, showing thou the thermal behavior of the detector and are used as reference level;
- 5 lines of isolating pixels, formed by identical photo-elements to those used in the matrix, used to ensure the 2048 active columns;
- The TH7889M is read at a frequency 888.89, KHz at High Resolution (HR – haute résolution) mode;
- The exposure times can be adjusted between 10µs and 167s;???
- Binning: 2x2, 4x4, 8x8;
- Working at non MPP in HV mode and MPP with HR mode;
- Band-pass limited at 14.1 MHz at HR (BP_{HR}) and at 37.9 MHz at HV (BP_{HV});
- The window is centered;
- Converting factor, without binning (F_{NB}): 7µV/e-;
- Converting factor, with binning (F_B): 4.5µV/e-;
- Maximum read charge without binning is equal to 115 ke- and 178 ke- with binning;
- Pixel size: 14µmx14µm with 100% aperture;
- Image zone: 28,67mmx28,67mm (2048x2048 pixels)
- Data rates up to 4 x 20MHz (compatibility with 15 frames/second);
- Very low dark current (MPP mode);
- Optimized resolution and responsivity in the 400-1100nm spectrum;
- Compatible with fiber optic face plate coupling;
- Operating temperature range: -55 ºC to +85 ºC;
- The read duration of a vertical transfer is 6µs.

4.2 Camera's performance

Processes that were took in account in order to achieve the best response:

- 3 stage peltier design with an high current supply
- Water cooling (ethanol solution at -10ºC) of the first stage of the peltiers with a cooling central to reduce the operating temperature of the camera to -55ºC, a temperature 15ºC below the minimal operating temperature recommended to this CCD to avoid malfunctioning.
- Reading frequency of the CCD reduced to 112 KHz, which is 8 times slower than 900KHz, the original operating frequency.

Although the camera was designed by spending the above efforts to achieve the maximal performance in long exposure acquiring, there are always thermal noise in this kind of cameras, and it always increases when augmenting the exposure time, which is in fact, the really important proposal of the camera. Nevertheless, by comparing this camera to other similar cameras, it can be said after the testing and the noise measures, that this camera has a good long exposure response, and a practical noise as low as ??? electron/volt.

Expected noise formulae:

$$BthrnbCCD := \sqrt{\frac{Bllhr^2}{Néchan} + Btrnb^2 + \frac{Bélechrnb^2}{Néchan} + \frac{Bmis^2}{Néchan} + \frac{Bps^2}{Néchan} + Bthhr^2}$$

- **BthrnbCCD:** Total noise of the CCD camera
- **Néchan:** Number of samples
- **BLhr:** Reading noise of the CCD in High Resolution mode (HR)
- **Btrnb:** Transfer noise of the CCD in HR mode
- **Bélechrnb:** Noise due to the amplification and conversion stages
- **Bmis:** Other noises
- **Bps:** Noise due to the power supply
- **Bthhr:** Thermal noise in HR mode

Expected noise:

Number of samples	N=4	N=8	N=16	N=32	N=64
Total Noise HR (TEMPERATURE = - 40ºC)	16,15	14,87	14,18	13,83	13,65
Total Noise HR (TEMPERATURE = - 55ºC)	11,12	9,16	8,00	7,35	7,00

Table 1: Expected noise for a 300sec exposure

Experimental noise:

Number of samples	N=4	N=8	N=16	N=32	N=64
Total Noise HR (TEMPERATURE = - 40ºC)					
Total Noise HR (TEMPERATURE = - 55ºC)					

Table 2: Experimental noise for a 300sec exposure

Graphics on expected noise:

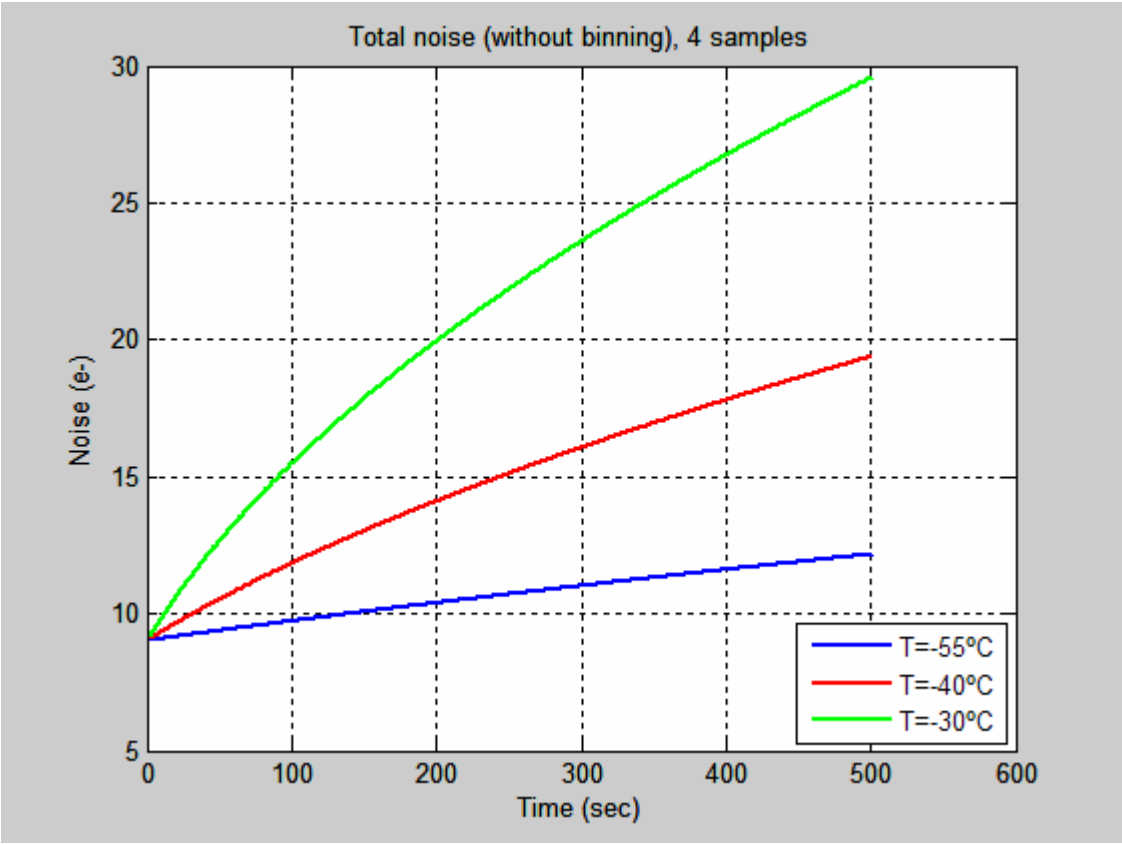

Figure 4: CCD's total noise without binning, samples=4

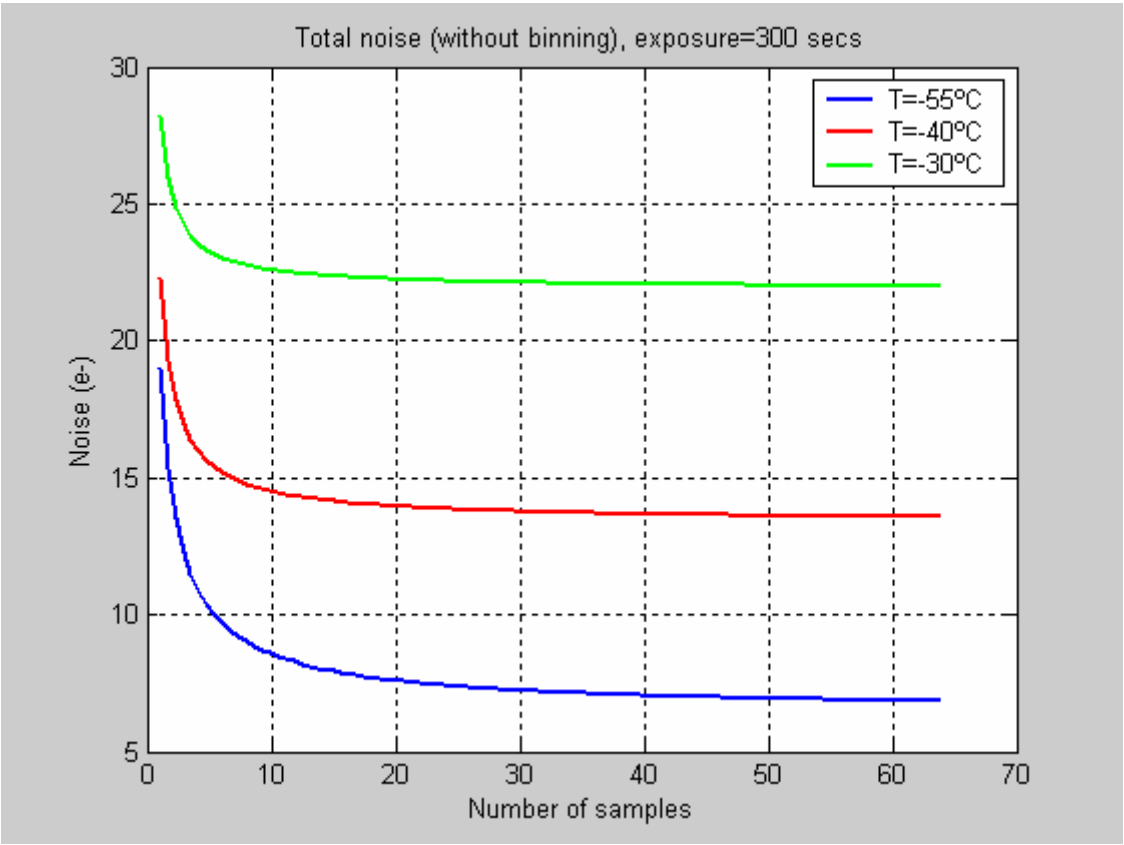

Figure 5: CCD's total noise without binning, exposure=300 secs

Graphics on experimental noise:

4.3 Acquiring modes

The **High Resolution mode** (HR) is a “slow scan” mode that allows the user to choose between 4 different speed/resolution ways of imaging:

HR1: High resolution with 1x1 binning (2048x2048)
HR2: High resolution with 2x2 binning (1024x1024)
HR4: High resolution with 4x4 binning (512x512)
HR8: High resolution with 8x8 binning (256x256)

The exposure time should not be inferior to 1ms regardless the acquisition mode. The maximum charge corresponds to the electron number that saturates the measure. 132000 electrons will be converted to the level of 65536 in the ADC (analog/digital converter), which results in about 2 electrons/level without binning and 3.2 in binning mode.

4.4 Side effects

Due to electronic imperfections the user should not expect to have a similar response from the 4 quadrants. In order to adjust the grey level of the image in running time (instead of executing a posterior image processing, which can also be done with great results) it is possible to use a dll function to adjust the 4 quadrants. This process should be taken each time the image doesn’t present similar grey levels in the 4 quadrants.

5. Audela commands

The commands supported by the Audela are grouped into the following sections discussed individually below:

- **General functions** (to use with every camera)
- **Specific functions** (to use only with the CEMES camera)

5.1 General functions

The general Audela functions are able to be used with every camera, and as being a responsibility of the Audela software constructor, they should not be descripted in this manual. Even so, they will be enumerated here and one can study them in the Audela's help or in the Audela's web page:

http://software.audela.free.fr/

General Audela functions:

drivername
product
nbcells
celldim
maxdyn
rgb
port
gain
bin
buf
acq
tel
shutter
temperature
interrupt
close
mirrorv
lasterror
headerproc
name
ccd
nbpix
pixdim
fillfactor
info
timer
readnoise
exptime
window
stop
radecfromtel
cooler
foclen
overscan
mirrorh
capabilities
debug

5.2 Specific functions

parameter

Command ::cam1 parameter argv[2] argv[3] argv[4] argv[5] argv[6]

Description camera's acquiring parameter setting

Parameters arg[2]: 0=High Speed OFF/ 1=High Speed ON;
arg[3]: 0=Speed 1 / 1=Speed 2 / 2=Speed 3;
arg[4]: number of pixels in X axis (0-2048);
arg[5]: number of pixels in Y axis (0-2048);
arg[6]: 0=Debug OFF / 1=Debug ON.

See also

---

peltON

Command ::cam1 peltON

Description Turns on the peltier

Parameters None

See also

---

peltOFF

Command ::cam1 peltOFF

Description Turns off the peltier

Parameters None

See also

---

settemp

Command ::cam1 settemp argv[2]

Description Sets the CCD's cold finger temperature

Parameters argv[3]: 0=-30ºC
1=-35ºC
2=-40ºC
3=-45ºC
4=-50ºC

See also

---

gettemp

Command ::cam1 gettemp argv[2]

Description Gets the peltier, thermal exchanger and cold finger temperatures

Parameters argv[3]: 0=Peltier
1=Thermal exchanger
2=Cold finger

Return The temperature of the demanded device

See also

---

ampliobtu

Command ::cam1 ampliobtu argv[2] argv[3] argv[4] argv[5] argv[6]

Description Sets amplifier and shutter acquiring modes.

Parameters argv[2]: Amplifier 0=Auto/1=Manual;
argv[3]: Shutter 0=Auto/1=Manual;
argv[4]: Amplifier 0=Off/1=On;
argv[5]: Shutter 0=Off/1=On;
argv[6]: Shutter mode 0=Astro/1=Detector/2=Imaging.

See also

---

stat_dina

Command ::cam1 stat_dina argv[2]

Description Selects either the static acquiring mode either the dynamic acquiring mode.

Parameters argv[2]: 0=static/1=dynamic.

See also balance

balance

Command ::cam1 balance argv[2]

Description Permits balancing the image on either the static acquiring mode either the dynamic acquiring mode. It can be used under binning mode or non binning mode.

Parameters argv[2]: 0=static / 1=dynamic / 2=static with binning / 3-dynamic with binning.

See also stat_dina

5.3 Usage examples

In the following example, after creating the camera (::cam::create cemes) when the user wants to know how to call the functions available he should write “::cam1” . When he do so, and as one can see in the picture below, the console prints in the screen all the functions of the camera chosen, which was the cemes camera in this particular case. In the example, the function called is “parameter”. To know how to call the function the user should not write any input arguments but simply “::cam1” and then the “function name”. After this, the user already knows how to correctly call the function.

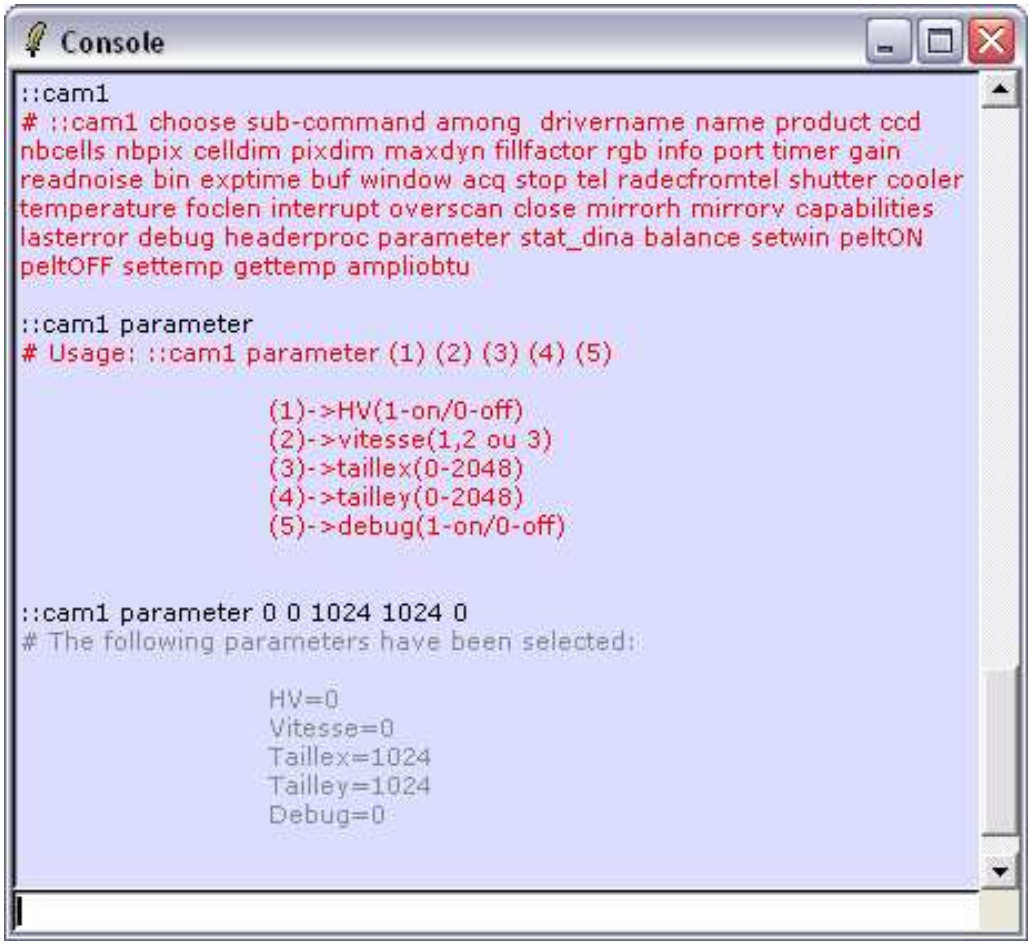

Figure 6: Audela console printscreen

NOTE: Although there are some functions with input arguments, there are other functions that are called without them. This is important to refer because one may write “::cam1 peltON” in order to know how to call the function and if he do so, the peltier is turned on suddenly, perhaps degrading the final image.

5.4 Calibrating the camera

Whenever the user pretends to reach the camera's maximal performance, he should calibrate the camera's 4 independent zones by balancing the grey level of each of them before starting the acquire process. This procedure consists on static and dynamic base voltages compensation. More specifically, it can be said that the goal of this procedure is to reach a 50 ADU mean in the 4 zones of the CCD in static mode and 150 ADU in dynamic mode.

In order to achieve an optimal balancing during this process one should:

1) Set the static acquiring mode by calling the function **stat_dina** followed by the in value "0" (static mode).

2) Once this function is called, the camera is ready to capture a static mode image, which is exactly what the user should do before the next step

3) If the 4 zones of the image are indeed quite different one from each other, the user should call the function **balance** followed by the in value "0" (static mode).

4) The last two steps should be taken repetitively until the grey level difference between the 4 zones of the image is negligible.

5) To repeat exactly the same last 4 steps, except that the **stat_dina** and **balance** functions should be called with the in value "1" (dynamic mode) instead of the "0" (static mode).

Example: The following picture shows two static mode images. In the left side there's a capture before the balancing process, and in the right a capture after 5 balancing processes.

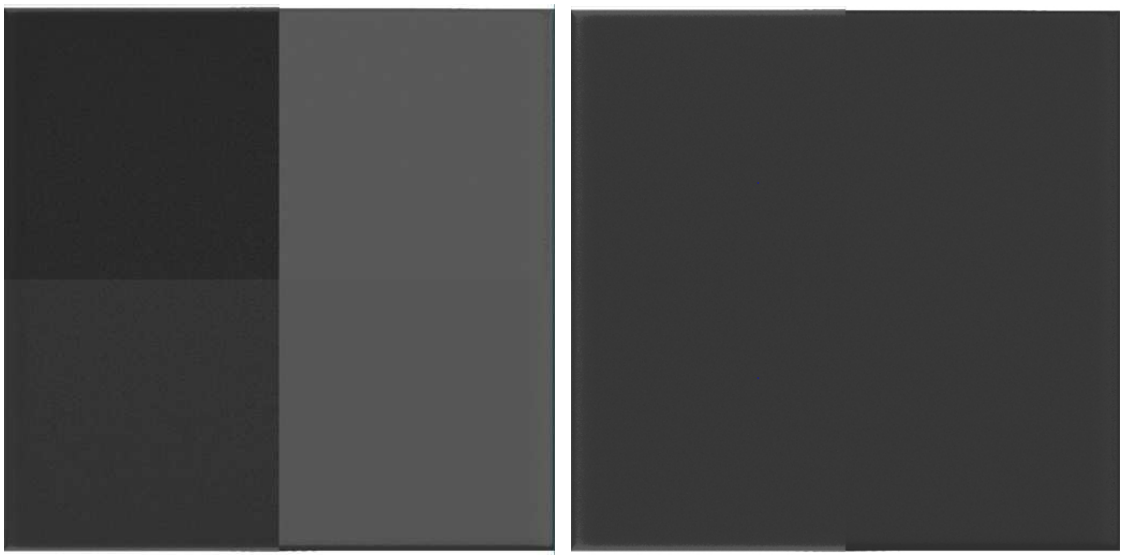

Figure 7: A static mode capture before and after the balancing process

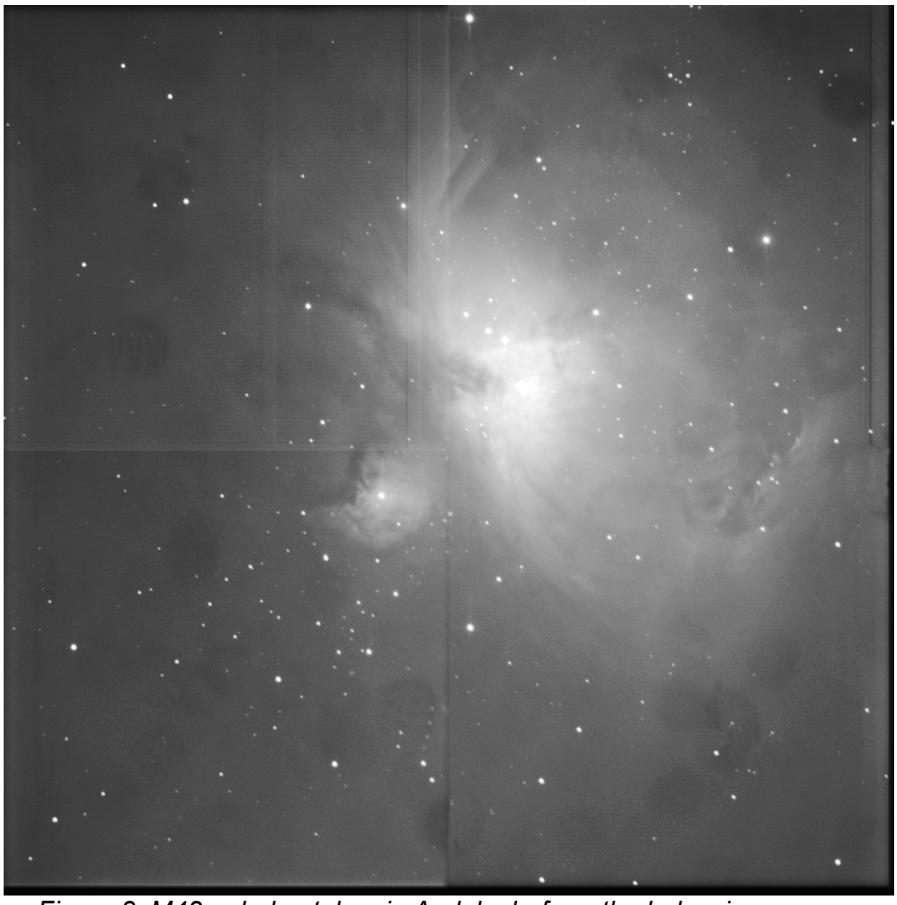

Figure 8: M42 nebulae taken in Audela before the balancing process

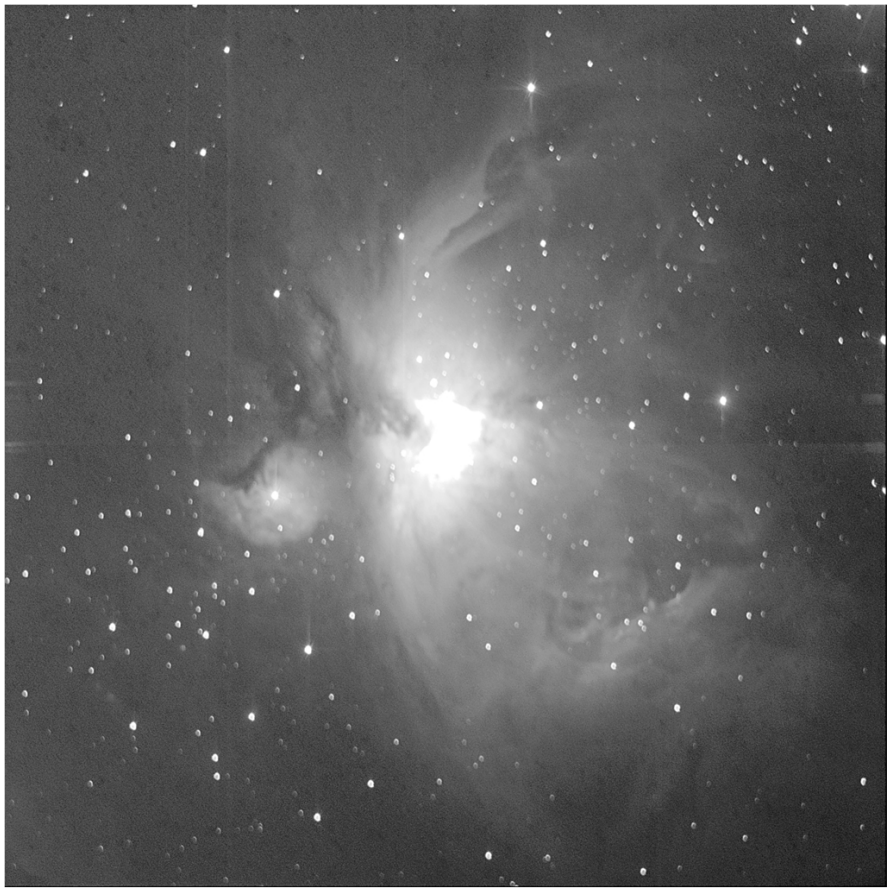

Figure 9: M42 nebulae taken in Audela after the balancing process

5.5 Images captured and the limiting magnitude

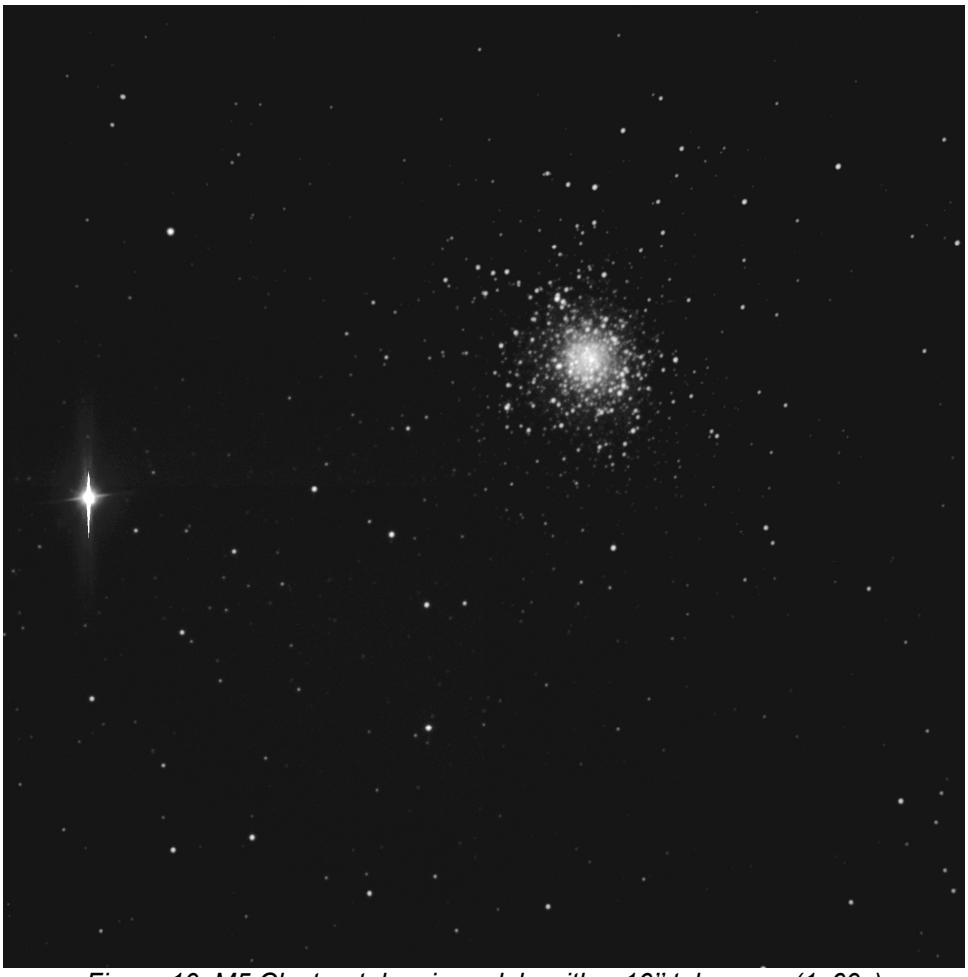

Figure 10: M5 Cluster taken in audela with a 16" telescope (1x60s)

Figure 11: M5 Cluster (amplification over the previous image)

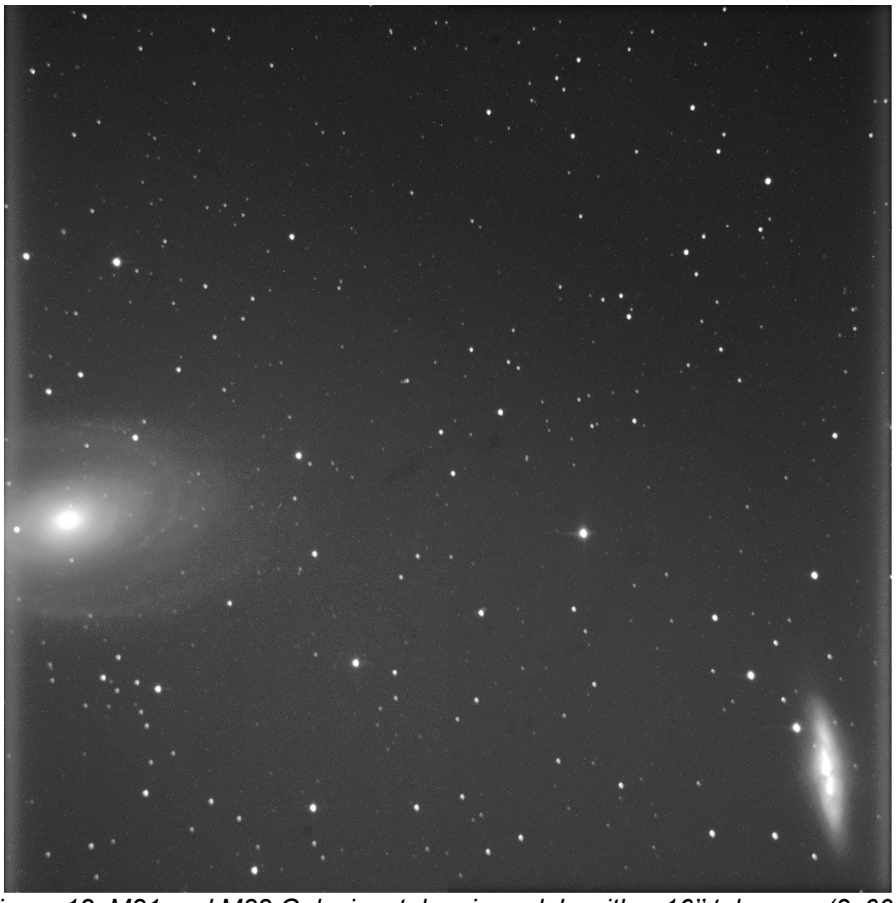

Figure 12: M81 and M82 Galaxies taken in audela with a 16'' telescope(3x60s)

Figure 13: M51 Galaxy taken in audela with a 16'' telescope (7x60s)

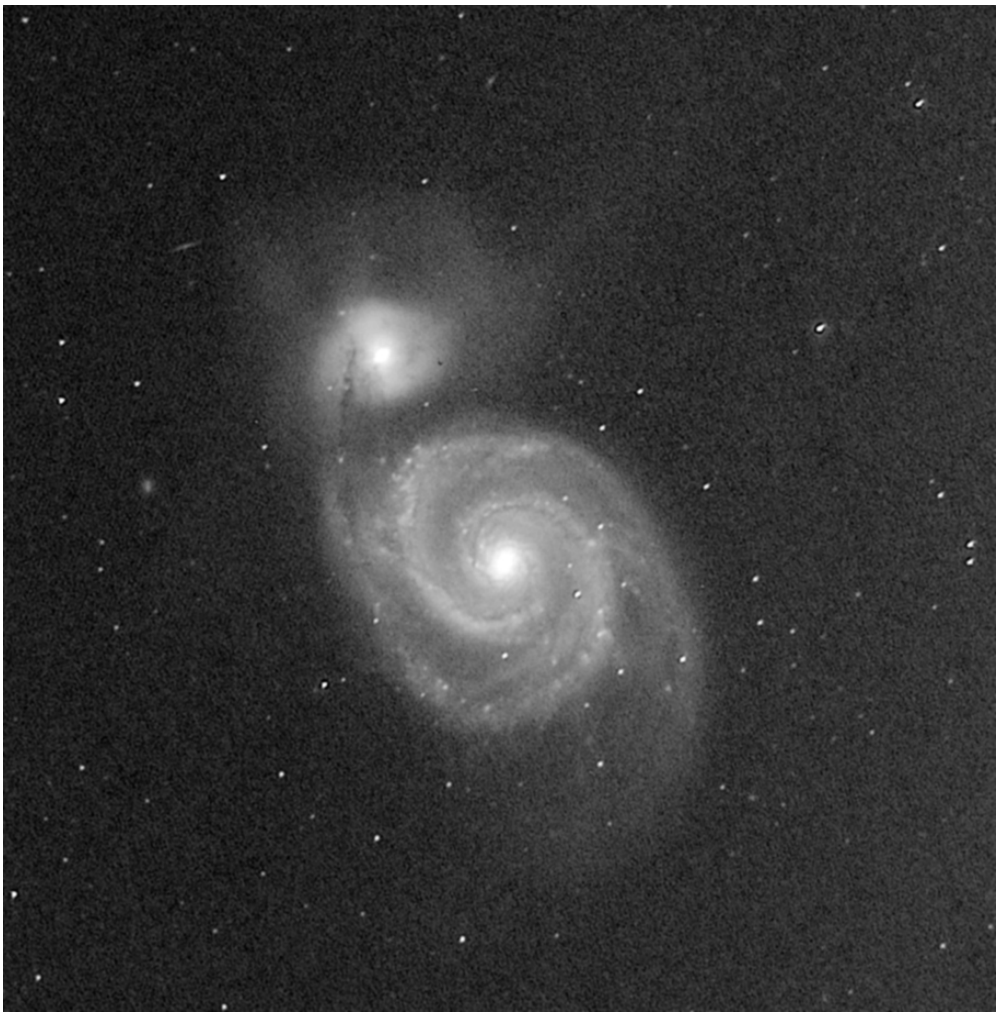

Figure 14: M51 Galaxy (amplification over the previous image)

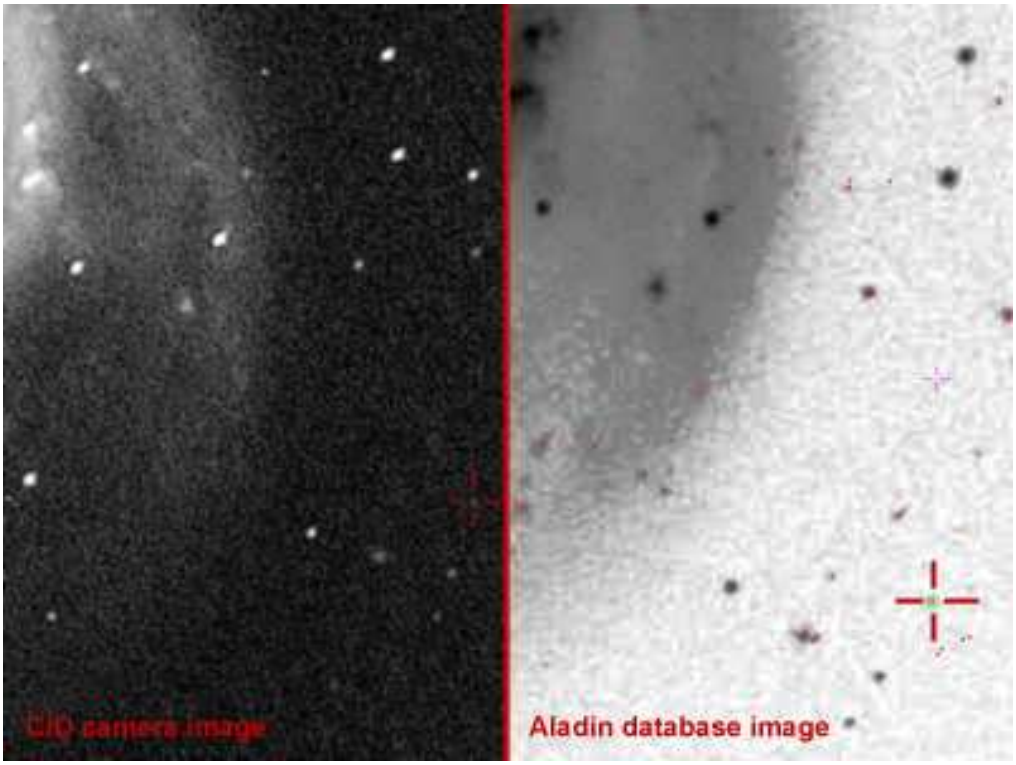

Figure 15: M51 Galaxy (A 20.44 magnitude star was found among the field)

6. Driver code

6.1 camera.c code

```
#define READOPTIC
#include "sysexp.h"
#if defined(OS_WIN)
#include <windows.h>
#endif
#if defined(OS_LIN)
#include <unistd.h>
#endif

#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdio.h>
#include <math.h>
#include <libcam/util.h>
#include "camera.h"
unsigned int mean1, mean2, mean3, mean4, mean5, mean6, mean7, mean8;

/*
 *  Definition of different cameras supported by this driver
 *  (see declaration in libstruc.h)
 */

#ifdef __cplusplus
extern "C" {
#endif

struct camini CAM_INI[] = {
   {"Cemes",                    /* camera name */
    "cemes",       /* camera product */
    "THX7899M",                         /* ccd name */
    2048, 2048,                 /* maxx maxy */
    0, 0,                  /* overscans x??? */
    0, 0,                  /* overscans y??? */
    14e-6, 14e-6,                  /* photosite dim (m) */
    65535.,                     /* observed saturation??? */
    1.,                    /* filling factor */
    2.,                    /* gain (e/adu)??? */
    11.,                   /* readnoise (e)??? */
    2, 2,                  /* default bin x,y */
    1.,                    /* default exptime */
    1,                          /* default state of shutter (1=synchro) */
    1,                          /* default num buf for the image */
    1,                          /* default num tel for the coordinates taken */
    0,                          /* default port index (0=lpt1) */
    1,                          /* default cooler index (1=on) */
    -15.,                  /* default value for temperature checked */
    1,                          /* default color mask if exists (1=cfa) */
    0,                          /* default overscan taken in acquisition (0=no) */
    1.                          /* default focal lenght of front optic system */
    },
   CAM_INI_NULL
};
```

```
static int cam_init(struct camprop *cam, int argc, char **argv);
static int cam_close(struct camprop * cam);
static void cam_start_exp(struct camprop *cam, char *amplionoff);
static void cam_stop_exp(struct camprop *cam);
static void cam_read_ccd(struct camprop *cam, unsigned short *p);
static void cam_shutter_on(struct camprop *cam);
static void cam_shutter_off(struct camprop *cam);
static void cam_ampli_on(struct camprop *cam);
static void cam_ampli_off(struct camprop *cam);
static void cam_measure_temperature(struct camprop *cam);
static void cam_cooler_on(struct camprop *cam);
static void cam_cooler_off(struct camprop *cam);
static void cam_cooler_check(struct camprop *cam);
static void cam_set_binning(int binx, int biny, struct camprop *cam);
static void cam_update_window(struct camprop *cam);

struct cam_drv_t CAM_DRV = {
   cam_init,
          cam_close,
   cam_set_binning,
   cam_update_window,
   cam_start_exp,
   cam_stop_exp,
   cam_read_ccd,
   cam_shutter_on,
   cam_shutter_off,
   cam_ampli_on,
   cam_ampli_off,
   cam_measure_temperature,
   cam_cooler_on,
   cam_cooler_off,
   cam_cooler_check
};

#ifdef __cplusplus
}
#endif

#ifdef __cplusplus
CAlims *calim;
Ccontroleur *control;
#endif

/* ========================================================== */
/* ========================================================== */
/* ===     Macro fonctions de pilotage de la camera      === */
/* ========================================================== */
/* ========================================================== */
/* Ces fonctions relativement communes a chaque camera.      */
/* et sont appelees par libcam.c                             */
/* Il faut donc, au moins laisser ces fonctions vides.       */
/* ========================================================== */

int cam_init(struct camprop *cam, int argc, char **argv)
/* ------------------------------------------------------- */
/* --- cam_init permet d'initialiser les variables de la --- */
/* --- structure 'camprop'                              --- */
/* --- specifiques a cette camera.                      --- */
/* ------------------------------------------------------- */
/* ------------------------------------------------------- */
```

```
{
        unsigned int error;

        mean1=0;
        mean2=0;
        mean3=0;
        mean4=0;
        mean5=0;
        mean6=0;
        mean7=0;
        mean8=0;
        cam->status=0;

        //CamInit();
        ControleurInit();
        AlimsInit();

        //Mise en marche
        error=SetBasseTension(1);
        if (error==1) {
           sprintf(cam->msg,"SetBasseTension returns %d",error);
           cam->status=1;
        return 1;
        }
        Sleep(1000);
        error=SerialDownload();
        if (error==1) {
           sprintf(cam->msg,"SerialDownload returns %d",error);
           cam->status=2;
        return 2;
        }

        control->Initialise(1);

        return 0;
}

int cam_close(struct camprop * cam)
{
        unsigned int error;

        error=Stop(0);
        if (error==1) {
           sprintf(cam->msg,"Stop returns %d",error);
           cam->status=4;
        return 3;
        }

        /*error=SetBasseTension(0);
        if (error==1) {
           sprintf(cam->msg,"SetBasseTension returns %d",error);
           cam->status=4;
        return 2;
        }*/

        AlimsStop();
        ControleurStop();

   return 0;
}
```

```
void cam_start_exp(struct camprop *cam, char *amplionoff)
{
	//BINNING (pour HR mode)
	//0: binning=1x1
	//1: binning=2x2
	//2: binning=4x4
	//3: binning=8x8

	//VITESSE:
	//1: HV3 -> binning=2x2
	//2: HV2 -> binning=4x4
	//3: HV1 -> binning=8x8
	//Default-> binning=2x2

	double expos;		//temps de pose
	unsigned int error,erreur,binning;
	unsigned int obtu=0;
	unsigned int automan=0;

	cam->status=0;

	if (cam->binx==1)
		binning=0;
	if (cam->binx==2)
		binning=1;
	if (cam->binx==4)
		binning=2;
	if (cam->binx==8)
		binning=3;

	error=SetModeBinning(cam->HV, binning, cam->vitesse, cam->debug);
	if (error==1) {
	   sprintf(cam->msg,"SetModeBinning returns %d",error);
	   cam->status=5;
	return ;
	}

	cam_update_window(cam);

	error=SetModeTension(cam->HV, binning);
	if (error==1) {
		sprintf(cam->msg,"SetModeTension returns %d",error);
		cam->status=6;
	return ;
	}

	erreur=SetTempsExposition(cam->exptime, &error);
	if (error==1 || erreur==1) {
	   sprintf(cam->msg,"SetTempsExposition returns %d",error);
	   cam->status=7;
	return ;
	}

	erreur=GetTempsExposition(&expos, &error);		//Pour verifier le
SetTempsExposition
	if (error==1 || erreur==1) {
	   sprintf(cam->msg,"GetTempsExposition returns %d",error);
	   cam->status=8;
	return ;
	}
```

```
	error=SetArea((16/cam->binx),(16/cam->binx),10000,10000);	//Regler la taille de
l'image
	//error=SetArea(((2080-cam->nb_photox)/cam->binx)/2,((2080-cam->nb_photoy)/cam-
>binx)/2,10000,10000);
	if (error==1) {
	   sprintf(cam->msg,"SetArea returns %d",error);
	   cam->status=9;
	return ;
	}

	if (cam->shutterindex==0) {
		automan=1;		//obturateur manuel
		obtu=0;			//obturateur ferme
	}
	if (cam->shutterindex==1) {
		automan=0;		//obturateur auto
	}
	if (cam->shutterindex==2) {
		automan=1;		//obturateur manuel
		obtu=1;			//obturateur ouvert
	}
	error=SetAmplisObtu(cam->ampliautoman,automan,cam->amplionoff,obtu,cam-
>obtumode);  //Regler les parameters d'amplificateur et d'obturateur

	if (error==1) {
	   sprintf(cam->msg,"SetAmplisObtu returns %d",error);
	   cam->status=10;
	return ;
	}
	control->SetDebugLevel(0);

	error=Stop(1);
	if (error==1) {
	   sprintf(cam->msg,"Stop returns %d",error);
	   cam->status=11;
	return ;
	}

	error=Start();
	if (error==1) {
	   sprintf(cam->msg,"Start returns %d",error);
	   cam->status=12;
	return ;
	}
}

void cam_stop_exp(struct camprop *cam)
{
	unsigned int error;

	error=Abort(0);
	if (error!=0) {
	   sprintf(cam->msg,"Abort returns %d",error);
	   cam->status=13;
	}
}
```

```
void cam_read_ccd(struct camprop *cam, unsigned short *p)
{
	bool full = false;
	unsigned short *sdata=NULL;
	unsigned short *sdata1=NULL;
	unsigned int k, tailletotal;			//tailletotal c'est le numero de pixels de l'image

	unsigned int slice=0;
	bool fin_image=0;
	int nb_image;
	int nbimages=0;
	unsigned long size = 0;
	unsigned int taille=((cam->nb_photox)/(cam->binx))*((cam->nb_photoy)/(cam-
>biny))*sizeof(unsigned short);  //cam->nb_photox=2048
	sdata=(unsigned short *)malloc(taille);
	sdata1=(unsigned short *)malloc(taille);
	unsigned short val;
	//unsigned int somamedia,moyennetotal,moyenne1voie,ecarttype;
	unsigned int i,j,error;
	unsigned int somadesvio=0;
	unsigned long sum=0;
	unsigned int equilibrer=1;

	if (cam->status!=0) {
		return;
	}

	//Dans ce boucle on attend la fin de la lecture de l'image
	while(size == 0){
		size = control->GetNextImage(sdata,0,&fin_image, &nb_image);  //((unsigned
char *)sdata,0,&fin_image, &nb_image);
	}
	size=0;

	//Dans ce boucle on attend la fin de la lecture d'une image de 2048x2048
	if (cam->binx==1)
	{
		while((size == 0) && (!fin_image)){
			size = control->GetNextImage(sdata1,0,&fin_image, &nb_image);
	//((unsigned char *)sdata1,0,&fin_image, &nb_image);
		}
	}

	error=Stop(1);
	if (error==1) {
	   sprintf(cam->msg,"Stop returns %d",error);
	   cam->status=14;
	return ;
	}

	size=0;

	tailletotal=(cam->nb_photox/cam->binx)*(cam->nb_photoy/cam->biny);

	//Dans ce boucle on rempli le pointeur *p avec l'image capturée
	for (k=0;k<tailletotal;k++)
	{
		val=sdata[k];  //na primeira volta sdata nao tem nada
		p[k] = val;
	}
```

```
if (cam->binx==1)
{
	//Dans ce boucle on rempli le pointeur *p avec l'image 2048x2048 capturée

	for (k=tailletotal/4;k<(tailletotal*0.75);k++)
	{
		val=sdata1[k];
		p[k] = val;
	}

	//REGLAGE DES 4 VOIES NON BINNING
	//FIRST SQUARE
	for (i=256;i<768;i++)
	{
		for (j=256;j<768;j++)
		{
			sum=p[2048*i+j]+sum;
		}
	}
	mean1=sum/(tailletotal/16);

	//SECOND SQUARE
	sum = 0;
	for (i=256;i<768;i++)
	{
		for (j=256;j<768;j++)
		{
			sum=p[2048*i+1024+j]+sum;
		}
	}
	mean2=sum/(tailletotal/16);

	//THIRD SQUARE
	sum = 0;
	for (i=256;i<768;i++)
	{
		for (j=256;j<768;j++)
		{
			sum=p[2048*i+(tailletotal/2)+j]+sum;
		}
	}
	mean3=sum/(tailletotal/16);

	//FOURTH SQUARE
	sum = 0;
	for (i=256;i<768;i++)
	{
		for (j=256;j<768;j++)
		{
			sum=p[2048*i+(tailletotal/2)+1024+j]+sum;
		}
	}
	mean4=sum/(tailletotal/16);		//NOW WE'VE GOT THE MEANS IN
NON BINNING MODE!
```

```
/*
		//Calculer la moyenne des 4 voies
		somamedia=0;
		for (i=0;i<tailletotal;i++)
		{
			somamedia=somamedia+p[i];
		}

		moyennetotal=somamedia/tailletotal;

		//Calculer la moyenne de la premiere voie
		somamedia=0;
		for (i=0;i<1024;i++)
		{
			for (j=0;j<1024;j++)
			{
				somamedia=somamedia+p[i*2048+j];
			}
		}

		moyenne1voie=somamedia/(tailletotal/4);

		//Calculer l'ecart type
		for (i=0;i<tailletotal;i++)
		{
			somadesvio=somadesvio+((p[i]-moyennetotal)*(p[i]-moyennetotal));
		}
		ecarttype=somadesvio/tailletotal;
		//ecarttype=sqrt(ecarttype);
*/
	}
	else
	{
		//REGLAGE DES 4 VOIES BINNING
		//FIRST SQUARE
		for (i=(256/cam->binx);i<(768/cam->binx);i++)
		{
			for (j=(256/cam->binx);j<(768/cam->binx);j++)
			{
				sum=p[2048/(cam->binx)*i+j]+sum;
			}
		}
		mean5=sum/(tailletotal/(16/cam->binx));

		//SECOND SQUARE
		sum = 0;
		for (i=(256/cam->binx);i<(768/cam->binx);i++)
		{
			for (j=(256/cam->binx);j<(768/cam->binx);j++)
			{
				sum=p[2048/(cam->binx)*i+(1024/(cam->binx))+j]+sum;
			}
		}
		mean6=sum/(tailletotal/(16/cam->binx));
```

```
                //THIRD SQUARE
                sum = 0;
                for (i=(256/cam->binx);i<(768/cam->binx);i++)
                {
                        for (j=(256/cam->binx);j<(768/cam->binx);j++)
                        {
                                sum=p[2048/(cam->binx)*i+((tailletotal/2)/cam->binx)+j]+sum;
                        }
                }
                mean7=sum/(tailletotal/(16/cam->binx));

                //FOURTH SQUARE
                sum = 0;
                for (i=(256/cam->binx);i<(768/cam->binx);i++)
                {
                        for (j=(256/cam->binx);j<(768/cam->binx);j++)
                        {
                                sum=p[(2048/cam->binx)*i+((tailletotal/2)/cam-
>binx)+(1024/cam->binx)+j]+sum;
                        }
                }
                mean8=sum/(tailletotal/(16/cam->binx));         //NOW WE'VE GOT THE
MEANS IN BINNING MODE!
        }
}

void cam_shutter_on(struct camprop *cam)
{
        SetAmplisObtu(1,1,1,1,0);
}

void cam_shutter_off(struct camprop *cam)
{
        SetAmplisObtu(1,1,1,0,0);
}

void cam_ampli_on(struct camprop *cam)
{
        SetAmplisObtu(1,1,1,1,0);
}

void cam_ampli_off(struct camprop *cam)
{
        SetAmplisObtu(1,1,0,1,0);
}

void cam_measure_temperature(struct camprop *cam)
{
        GetTemperature(2,&(cam->temperature));
}

void cam_cooler_on(struct camprop *cam)
{
        SetPeltier(1);
        cam->coolerindex=1;
}
```

```
void cam_cooler_off(struct camprop *cam)
{
	SetPeltier(0);
	cam->coolerindex=0;
}

void cam_cooler_check(struct camprop *cam)
{
	double temp;
	unsigned int consigne;
	temp=cam->check_temperature;
	if (temp>=-30.) { consigne = 7900; cam->check_temperature=-30.; }
	else if (temp<=-50.) { consigne = 25500; cam->check_temperature=-50.; }
	else {
		consigne=(unsigned int)pow(10.,2.6551e-6*temp*temp*temp+4.9964e-
4*temp*temp+1.516e-3*temp+3.565199); //conversion temp->consigne
	}
   SetPeltierConsigne(consigne);
}

void cam_set_binning(int binx, int biny, struct camprop *cam)
{
	//bin doit etre 1, 2, 4 ou 8
	if ((binx!=1) && (binx!=2) && (binx!=4) && (binx!=8) && (binx!=1) && (binx!=2) &&
(biny!=4) && (biny!=8))
	{
		sprintf(cam->msg,"The binning must be 1, 2, 4 or 8");
	return ;
	}
	else
	{
		cam->binx = binx;
		cam->biny = biny;
	}
}

void cam_update_window(struct camprop *cam)
{
   int maxx, maxy;
   maxx = cam->nb_photox;
   maxy = cam->nb_photoy;
   if (cam->x1 > cam->x2)
	libcam_swap(&(cam->x1), &(cam->x2));
   if (cam->x1 < 0)
	cam->x1 = 0;
   if (cam->x2 > maxx - 1)
	cam->x2 = maxx - 1;

   if (cam->y1 > cam->y2)
	libcam_swap(&(cam->y1), &(cam->y2));
   if (cam->y1 < 0)
	cam->y1 = 0;
   if (cam->y2 > maxy - 1)
	cam->y2 = maxy - 1;

   cam->w = (cam->x2 - cam->x1) / cam->binx + 1;
   cam->x2 = cam->x1 + cam->w * cam->binx - 1;
   cam->h = (cam->y2 - cam->y1) / cam->biny + 1;
   cam->y2 = cam->y1 + cam->h * cam->biny - 1;
}
```

```
// ================================================================
// ================================================================
// ===     Fonctions de base pour le pilotage de la camera      ===
// ================================================================
// ================================================================
// Ces fonctions sont tres specifiques a chaque camera.


void ControleurInit()
{
	control = new Ccontroleur();
}

void ControleurStop()
{
	if (control != NULL) {
		delete control;
		control = NULL;
	}
}

void AlimsInit()
{
	calim = new CAlims();
}

void AlimsStop()
{
	if (calim != NULL) {
		delete calim;
		calim = NULL;
	}
}

void equilibrer(unsigned int stat_dina)
{
	//EQUILIBRER EN CAS DE BINNING = 1
	unsigned int dif1,dif2,dif3,dif4,dif5,dif6,dif7,dif8;

	//MAINTENANT IL FAUT EQUILIBRER L'IMAGE FINALE

	//equilibrage statique
	if (stat_dina==0)
	{
		dif1=(mean1-50)*6 + (control->ReadMyDoubleKey2("DEC1", (2000)));
		dif2=(mean2-50)*6 + (control->ReadMyDoubleKey2("DEC2", (2000)));
		dif3=(mean3-50)*6 + (control->ReadMyDoubleKey2("DEC3", (2000)));
		dif4=(mean4-50)*6 + (control->ReadMyDoubleKey2("DEC4", (2000)));

		SetDECALAGE(13,dif1);
		SetDECALAGE(14,dif2);
		SetDECALAGE(15,dif3);
		SetDECALAGE(16,dif4);

		control->SaveRegistry();
	}
```

```
	//equilibrage dinamique  (funciona mas demorou bastante a convergir para 150)
	if (stat_dina==1)
	{
		dif1=(mean1-150)/16*6 + (control->ReadMyDoubleKey2("DECD1", (2000)));
		dif2=(mean2-150)/16*6 + (control->ReadMyDoubleKey2("DECD2", (2000)));
		dif3=(mean3-150)/16*6 + (control->ReadMyDoubleKey2("DECD3", (2000)));
		dif4=(mean4-150)/16*6 + (control->ReadMyDoubleKey2("DECD4", (2000)));

		SetDECALAGE(17,dif1);
		SetDECALAGE(18,dif2);
		SetDECALAGE(19,dif3);
		SetDECALAGE(20,dif4);

		control->SaveRegistry();
	}

	//equilibrage statique avec binning
	if (stat_dina==2)
	{
		dif5=(mean5-50)*6 + (control->ReadMyDoubleKey2("DECB1", (2000)));
		dif6=(mean6-50)*6 + (control->ReadMyDoubleKey2("DECB2", (2000)));
		dif7=(mean7-50)*6 + (control->ReadMyDoubleKey2("DECB3", (2000)));
		dif8=(mean8-50)*6 + (control->ReadMyDoubleKey2("DECB4", (2000)));

		SetDECALAGE(13,dif5);
		SetDECALAGE(14,dif6);
		SetDECALAGE(15,dif7);
		SetDECALAGE(16,dif8);

		control->SaveRegistry();
	}

	//equilibrage dinamique avec binning
	if (stat_dina==3)
	{
		dif5=(mean5-150)/16*6 + (control->ReadMyDoubleKey2("DECDB1", (2000)));
		dif6=(mean6-150)/16*6 + (control->ReadMyDoubleKey2("DECDB2", (2000)));
		dif7=(mean7-150)/16*6 + (control->ReadMyDoubleKey2("DECDB3", (2000)));
		dif8=(mean8-150)/16*6 + (control->ReadMyDoubleKey2("DECDB4", (2000)));

		SetDECALAGE(17,dif5);
		SetDECALAGE(18,dif6);
		SetDECALAGE(19,dif7);
		SetDECALAGE(20,dif8);

		control->SaveRegistry();
	}
}

unsigned int ResetADLINK()
{
	bool ret=control->ResetADLINK();

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
```

```
unsigned int SetConfCalcul(short ConfCalcul)
{
	bool ret=control->SetConfCalcul(ConfCalcul);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int SetDECALAGE(unsigned int n, unsigned int val)
{
	bool ret=control->SetDECALAGE(n, val);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int SetBasseTension(int onoff)
{
	bool btruefalse;
	if (onoff==1) { btruefalse=true; }
	else { btruefalse=false; }
	bool ret=calim->SetBasseTension(btruefalse);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int SetModeTension(int HR, unsigned int binning)
{
	bool truefalse;
	if (HR==1) { truefalse=true; }
	else { truefalse=false; }
	bool ret=calim->SetModeTension(truefalse, binning);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
unsigned int GetTemperature(int n, double *temp)
{
	bool ret=calim->GetTemperature(n, temp);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
unsigned int SetPeltier(int on)
{
	bool truefalse;
	if (on==1) { truefalse=true; }
	else { truefalse=false; }
	bool ret=calim->SetPeltier(truefalse);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
```

```
unsigned int SetPeltierConsigne(int temp)
{
	bool ret=calim->SetPeltierConsigne(temp);

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int SetSupply(int on)
{
	bool truefalse;
	if (on==1) { truefalse=true; }
	else { truefalse=false; }
	bool ret=calim->SetSupply(truefalse);

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int SetModeBinning(int HV, unsigned int binning, unsigned int vitesse,int debug)
{
	bool HVbool;
	if (HV==1) { HVbool=true; }
	else { HVbool=false; }

	bool debugbool;
	if (debug==1) { debugbool=true; }
	else { debugbool=false; }

	bool ret=control->SetModeBinning(HVbool, binning, vitesse, debugbool);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int SetTempsExposition(double pose, unsigned int *erreur)
{
	bool erreurbool=0;
	bool ret=control->SetTempsExposition(pose, &erreurbool);
	if (erreurbool==false) { *erreur=0; }
	else { *erreur=1; }

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
```

```
unsigned int SetArea(unsigned short x0, unsigned short y0, unsigned short xb, unsigned short
yb)
{
	xb=10000;
	yb=10000;
	bool ret=control->SetArea(x0, y0, xb, yb);

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int Start(void)
{
	bool erreurbool=0;
	int erreur=0;
	erreurbool=control->Start();

	if (erreurbool==true) { erreur=0; }
	else { erreur=1; }
	return erreur;
}

unsigned int Stop(int stp)
{
	bool onBOOL;
	int erreur=0;
	if (stp==1) { onBOOL=true; }
	else { onBOOL=false; }
	bool erreurbool=control->Stop(onBOOL);

	if (erreurbool==true) { erreur=0; }
	else { erreur=1; }
	return erreur;
}

unsigned int SerialDownload(void)
{
	unsigned int com=0, iteration=0;
	bool com2=0;
	while((com==0)&&(iteration<10))
	{
		com = control->SetDECALAGE(20, 2000);
		iteration++;
	}

	if (com!=0)
	{
		com2 = control->SerialDownload();
	}

	unsigned int error;
	if (com2==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
```

```
void Initialise(int initls)
{
	bool onBOOL;
	if (initls==1) { onBOOL=true; }
	else { onBOOL=false; }
	control->Initialise(onBOOL);
}

unsigned int GetTempsExposition(double *pose, unsigned int *erreur)
{
		bool erreurbool=0;
		bool ret=control->GetTempsExposition(pose, &erreurbool);
		if (erreurbool==true) { *erreur=0; }
		else { *erreur=1; }

		unsigned int error;
		if (ret==true) { error = 0 ; }
		else { error = 1; }
		return error;
}

unsigned int SetImageSize(unsigned long sx, unsigned long sy)
{
		bool ret=control->SetImageSize(sx,sy);

		unsigned int error;
		if (ret==true) { error = 0 ; }
		else { error = 1; }
		return error;
}

unsigned int GetImageSize(unsigned long *sx, unsigned long *sy)
{
	bool ret=control->GetImageSize(sx,sy);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int Abort(int abrt)
{
	bool onBOOL;
	if (abrt==1) { onBOOL=true; }
	else { onBOOL=false; }
	bool ret=control->Abort(onBOOL);
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int Reset(void)
{
	bool ret=control->Reset();
	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
```

```
unsigned int GetStatusCamera(int est0, int est1)
{
	bool *estBOOL0=0;
	if (est0==1) { *estBOOL0=true; }
	else { *estBOOL0=false; }

	bool *estBOOL1=0;
	if (est1==1) { *estBOOL1=true; }
	else { *estBOOL1=false; }

	bool ret=control->GetStatusCamera(estBOOL0, estBOOL1);

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int GetStatusAmplis(int comG, int comBP, int comL, int ampliam, int ampliof)
{
	bool *b0=0;
	if (comG==1) { *b0=true; }
	else { *b0=false; }

	bool *b1=0;
	if (comBP==1) { *b1=true; }
	else { *b1=false; }

	bool *b2=0;
	if (comL==1) { *b2=true; }
	else { *b0=false; }

	bool *b3=0;
	if (ampliam==1) { *b3=true; }
	else { *b3=false; }

	bool *b4=0;
	if (ampliof==1) { *b4=true; }
	else { *b4=false; }

	bool ret=control->GetStatusAmplis(b0, b1, b2, b3, b4);

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
```

```
unsigned int GetStatusObtu(int onoff, int am, int ouvfer)
{
	bool *ON=0;
	if (onoff==1) { *ON=true; }
	else { *ON=false; }

	bool *AM=0;
	if (am==1) { *AM=true; }
	else { *AM=false; }

	bool *OUV=0;
	if (ouvfer==1) { *OUV=true; }
	else { *OUV=false; }

	bool ret=control->GetStatusObtu(ON, AM, OUV);

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}

unsigned int SetAmplisObtu(int onoff, int onoff2, int onoff3, int onoff4, int onoff5)
{
	int on5int;

	bool onBOOL;
	if (onoff==1) { onBOOL=true; }
	else { onBOOL=false; }

	bool on2BOOL;
	if (onoff2==1) { on2BOOL=true; }
	else { on2BOOL=false; }

	bool on3BOOL;
	if (onoff3==1) { on3BOOL=true; }
	else { on3BOOL=false; }

	bool on4BOOL;
	if (onoff4==1) { on4BOOL=true; }
	else { on4BOOL=false; }

	on5int=onoff5;

	bool ret=control->SetAmplisObtu(onBOOL, on2BOOL, on3BOOL, on4BOOL, on5int);

	unsigned int error;
	if (ret==true) { error = 0 ; }
	else { error = 1; }
	return error;
}
```

```
// ====================================================================
// ====================================================================
// ===     Fonctions etendues pour le pilotage de la camera     ===
// ====================================================================
// ====================================================================
// Ces fonctions sont tres specifiques a chaque camera.
// ====================================================================


void cemes_updatelog(struct camprop *cam, char *filename, char *comment)
{
        /*
   char s[100];
   char fname[256];
   FILE *fil;
   if (cam->updatelogindex == 1) {
        Tcl_Eval(cam->interp, "clock format [clock seconds] -format \"%Y-%m-
%dT%H:%M:%S.00\"");
        strcpy(s, cam->interp->result);
        if (strcmp(filename, "") == 0) {
           strcpy(fname, "updateclock.log");
        } else {
           strcpy(fname, filename);
        }
        fil = fopen(fname, "at");
        if (fil == NULL)
           return;
        fprintf(fil, "%s : %s\n", s, comment);
        fclose(fil);
   }
   return;
        */
}
```

6.2 camtcl.c code

```
#include "sysexp.h"

#if defined(OS_WIN)
#include <windows.h>
#endif
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "camera.h"
#include <libcam/libcam.h>
#include "camtcl.h"
#include <libcam/util.h>

/*
#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif*/

//ACRESCENTAMOS TUDO ISTO ATE ...
/*
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                                                )
{
   switch (ul_reason_for_call)
          {
                case DLL_PROCESS_ATTACH:
                      AlimsInit();
                      ControleurInit();
                      break;
                case DLL_THREAD_ATTACH:
                      break;
                case DLL_THREAD_DETACH:
                      break;
                case DLL_PROCESS_DETACH:
                            AlimsStop();
                            ControleurStop();
                      break;
   }
   return TRUE;
}
*/
//... ATE AQUI

int cmdPeltierMarche(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
          int result=TCL_OK;
          char ligne[256];

          if (SetPeltier(true)==1) { //true=1-> peltier on
                      sprintf(ligne, "Peltier error");
                      Tcl_SetResult(interp, ligne, TCL_VOLATILE);
                      result = TCL_ERROR;
```

```
	}
	else {
			sprintf(ligne, "Peltier ON\n");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_OK;
	}
	return result;
}

int cmdPeltierArret(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	int result=TCL_OK;
	char ligne[256];

	if (SetPeltier(false)==1) { //false=0-> peltier off
			sprintf(ligne, "Peltier error");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
	}
	else {
			sprintf(ligne, "Peltier OFF\n");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_OK;
	}
	return result;
}
int cmdCemesGetTemp(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	int result=TCL_OK;
	char ligne[256];
   struct camprop *cam;
   cam = (struct camprop *) clientData;
	unsigned int n;
	double temperature=0;
		if (argc!=3)
	{
		sprintf(ligne, "Usage: %s %s (1)\n\n\t\t(1)->Temperature to read:  \n\t\t\t\t0-
Peltier\n\t\t\t\t\t1-Thermal Exchanger\n\t\t\t\t\t2-Cold Finger\n", argv[0], argv[1]);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
		result = TCL_ERROR;
	}
	else
	{
		n=atoi(argv[2]);
		if ((n<0) || (n>2))
		{
			sprintf(ligne, "It can only be chosen 0, 1 or 2");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
		}
		else
		{
			GetTemperature(n, &temperature);
			sprintf(ligne, "Temperature (Celsius): %f\n", temperature);
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_OK;
		}
	}
	return result;
}
```

```
int cmdCemesSetTemp(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	int result=TCL_OK;
	char ligne[256];
   struct camprop *cam;
   cam = (struct camprop *) clientData;

	unsigned int Temp;
	unsigned int consigne=0;
	bool ret = false;

	if (argc!=3)
	{
		sprintf(ligne, "Usage: %s %s (1)\n\n\t\t(1)->Temperature to reach:  \n\t\t\t\t0-> -
30\n\t\t\t\t1-> -35\n\t\t\t\t2-> -40\n\t\t\t\t3-> -45\n\t\t\t\t4-> -50\n", argv[0], argv[1]);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
		result = TCL_ERROR;
	}
	else
	{
		Temp=atoi(argv[2]);
		if ((Temp<0) || (Temp>4))
		{
			sprintf(ligne, "It can only be chosen 0, 1, 2, 3 or 4");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
		}
		else
		{
			if (Temp == 0) consigne = 7900;			//-30°C
			if (Temp == 1) consigne = 10250;	//-35°C
			if (Temp == 2) consigne = 13600;	//-40°C
			if (Temp == 3) consigne = 18500;	//-45°C
			if (Temp == 4) consigne = 25500;	//-50°C

			SetPeltierConsigne(consigne);

			if (Temp == 0) sprintf(ligne, "You've chosen -30\n");
			if (Temp == 1) sprintf(ligne, "You've chosen -35\n");
			if (Temp == 2) sprintf(ligne, "You've chosen -40\n");
			if (Temp == 3) sprintf(ligne, "You've chosen -45\n");
			if (Temp == 4) sprintf(ligne, "You've chosen -50\n");

			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_OK;
		}
	}
	return result;
}


int cmdCemesParam(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	char ligne[256];
	int result=TCL_OK;
   struct camprop *cam;
   cam = (struct camprop *) clientData;

	if (argc!=7)
	{
```

```
		sprintf(ligne, "Usage: %s %s (1) (2) (3) (4) (5)\n\n\t\t(1)->HV(1-on/0-off)\n\t\t(2)->vitesse(1,2 ou 3)\n\t\t(3)->taillex(0-2048)\n\t\t(4)->tailley(0-2048)\n\t\t(5)->debug(1-on/0-off)\n", argv[0], argv[1]);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
		result = TCL_ERROR;
		return result;
	}
	else
	{
		if ((atoi(argv[2])!=1) && ((atoi(argv[2])!=0)))
		{
			sprintf(ligne, "HV(1-on/0-off)");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
			return result;
		}

		//HAUTE VITESSE (1-ON/0-OFF)
		if (atoi(argv[2])==1)
		{
			cam->HV=1;

			//VITESSE (1/2/3)
			if ((atoi(argv[3])!=2) && (atoi(argv[3])!=3))
			{
				cam->vitesse=1; //vitesse 1
			}
			else{
				cam->vitesse=atoi(argv[3]); //vitesse 2 ou 3
			}
		}
		else
		{
			cam->HV=0;
		}

		//TAILLEX (0-2048)
		if ((atoi(argv[4])<0) || (atoi(argv[4])>2048))
		{
			sprintf(ligne, "taillex(0-2048)");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
			return result;
		}
		cam->nb_photox=atoi(argv[4]);

		//TAILLEY (0-2048)
		if ((atoi(argv[5])<0) || (atoi(argv[5])>2048))
		{
			sprintf(ligne, "tailley(0-2048)");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
			return result;
		}
		cam->nb_photoy=atoi(argv[5]);
```

```
                //DEBUG (1-ON/0-OFF)
                if ((atoi(argv[6])!=0) && (atoi(argv[6])!=1))
                {
                        sprintf(ligne, "debug(1-on/0-off)");
                        Tcl_SetResult(interp, ligne, TCL_VOLATILE);
                        result = TCL_ERROR;
                        return result;
                }
                cam->debug=atoi(argv[6]);
        }
        sprintf(ligne, "The following parameters have been selected: \n\n\t\tHV=%d\n\t\tVitesse=%d\n\t\tTaillex=%d\n\t\tTailley=%d\n\t\tDebug=%d\n", cam->HV, cam->vitesse, cam->nb_photox, cam->nb_photoy, cam->debug);
        Tcl_SetResult(interp, ligne, TCL_VOLATILE);
        return result;
}

int cmdCemesObtu(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
        char ligne[256];
        int result=TCL_OK;
   struct camprop *cam;
   cam = (struct camprop *) clientData;
        if (argc!=7)
        {
                sprintf(ligne, "Usage: %s %s (1) (2) (3) (4) (5)\n\n\t\t(1)->Ampli(0-AUTO/1-MAN)\n\t\t(2)->Obtu(0-AUTO/1-MAN)\n\t\t(3)->Ampli(0-Off, 1-On)\n\t\t(4)->Obtu(0-Off, 1-On)\n\t\t(5)->Obtu Mode(0-Astro, 1-Detector, 2-Imag)\n", argv[0], argv[1]);
                Tcl_SetResult(interp, ligne, TCL_VOLATILE);
                result = TCL_ERROR;
                return result;
        }
        else
        {
                //AMPLIFICATEUR (0-AUTO/1-MAN)
                if ((atoi(argv[2])!=1) && ((atoi(argv[2])!=0)))
                {
                        sprintf(ligne, "Ampli(0-AUTO/1-MAN)");
                        Tcl_SetResult(interp, ligne, TCL_VOLATILE);
                        result = TCL_ERROR;
                        return result;
                }

                //OBTURATEUR (0-AUTO/1-MAN)
                if ((atoi(argv[3])!=1) && ((atoi(argv[3])!=0)))
                {
                        sprintf(ligne, "Obtu(0-AUTO/1-MAN)");
                        Tcl_SetResult(interp, ligne, TCL_VOLATILE);
                        result = TCL_ERROR;
                        return result;
                }

                //AMPLIFICATEUR (0-OFF/1-ON)
                if ((atoi(argv[4])!=1) && ((atoi(argv[4])!=0)))
                {
                        sprintf(ligne, "Ampli(0-Off/1-On)");
                        Tcl_SetResult(interp, ligne, TCL_VOLATILE);
                        result = TCL_ERROR;
                        return result;
                }
```

```
		//OBTURATEUR (0-OFF/1-ON)
		if ((atoi(argv[5])!=1) && ((atoi(argv[5])!=0)))
		{
			sprintf(ligne, "Obtu(0-Off/1-On)");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
			return result;
		}

		//MODE OBTURATEUR (0-ASTRO/1-DETECTEUR/2-IMAGEUR)
		if ((atoi(argv[6])!=2) && (atoi(argv[6])!=1) && ((atoi(argv[6])!=0)))
		{
			sprintf(ligne, "Obtu Mode(0-Astro, 1-Detector, 2-Imag)");
			Tcl_SetResult(interp, ligne, TCL_VOLATILE);
			result = TCL_ERROR;
			return result;
		}

		//PARAMETRAGE
		cam->ampliautoman=atoi(argv[2]);
		cam->obtuautoman=atoi(argv[3]);
		cam->amplionoff=atoi(argv[4]);
		cam->obtuonoff=atoi(argv[5]);
		cam->obtumode=atoi(argv[6]);
		sprintf(ligne, "The following parameters have been selected: \n\n\t\tAmpli(0-
AUTO/1-MAN)=%d\n\t\tObtu(0-AUTO/1-MAN)=%d\n\t\tAmpli(0-Off/1-On)=%d\n\t\tObtu(0-
Off/1-On)=%d\n\t\tObtu Mode(0-Astro, 1-Detector, 2-Imag)=%d\n", cam->ampliautoman, cam-
>obtuautoman, cam->amplionoff, cam->obtuonoff, cam->obtumode);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
	}
	return result;
}

int cmdStatique_dinamique(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	int result=TCL_OK;
	char ligne[256];

	if (argc!=3)
	{
		sprintf(ligne, "Usage: %s %s 0=static/1=dynamic\n", argv[0], argv[1]);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
		result = TCL_ERROR;
		return result;
	}
	else
	{
		if (atoi(argv[2])==0)
		{
			//statique
			SetConfCalcul(2);
		}
		else
		{
			//dinamique
			SetConfCalcul(5);
		}
	}
	return result;
}
```

```
int cmdBalance(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	int result=TCL_OK;
	char ligne[256];

	if (argc!=3)
	{
		sprintf(ligne, "Usage: %s %s 0=static (non binning)/1=dynamic (non
binning)/2=static (binning)/3=dynamic (binning)\n", argv[0], argv[1]);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
		result = TCL_ERROR;
		return result;
	}
	else
	{
		if (atoi(argv[2])==0)
		{
			//statique
			equilibrer(0);
		}
		if (atoi(argv[2])==1)
		{
			//dinamique
			equilibrer(1);
		}
		if (atoi(argv[2])==2)
		{
			//statique avec binning
			equilibrer(2);
		}
		if (atoi(argv[2])==3)
		{
			//dinamique avec binning
			equilibrer(3);
		}
	}

	return result;
}

int cmdSetwindow(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	int result=TCL_OK;
	/*char ligne[256];

	if (argc!=4)
	{
		sprintf(ligne, "Usage: %s %s (1) (2)\n\n\t\t(1)->X size (0-2048)\n\t\t(2)->Y size
(0-2048)", argv[0], argv[1]);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
		result = TCL_ERROR;
		return result;
	}
	else
	{
		SetArea((16/cam->binx),(16/cam->biny),10000,10000);
	}
*/
	return result;
}
```

```
int cmdRESET(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[])
{
	int result=TCL_OK;
	char ligne[256];

	if (argc!=2)
	{
		sprintf(ligne, "Usage: %s %s 0=static/1=dynamic\n", argv[0], argv[1]);
		Tcl_SetResult(interp, ligne, TCL_VOLATILE);
		result = TCL_ERROR;
		return result;
	}
	else
	{
		ResetADLINK();
	}

	return result;
}
```

6.3 camera.h code

```
#ifndef __CAMERA_H__
#define __CAMERA_H__

#ifdef OS_LIN
#define __KERNEL__
#   include <sys/io.h>
#endif

#include <tcl.h>
#include "libname.h"
#include <libcam/libstruc.h>

#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <stdio.h>

#include <libcam/util.h>

#ifdef __cplusplus
extern "C" {
#endif

/*
 * Donnees propres a chaque camera.
 */

/* --- structure qui accueille les parametres---*/
struct camprop {
   /* --- parametres standards, ne pas changer --- */
   COMMON_CAMSTRUCT;
   /* Ajoutez ici les variables necessaires a votre camera (mode d'obturateur, etc). */

         //Variables a Leandro et Luis pour CEMES
         boolean HV;
         unsigned int vitesse;
         boolean debug;
         unsigned int sizeX;
         unsigned int sizeY;
         boolean ampliautoman;
         boolean obtuautoman;
         boolean amplionoff;
         boolean obtuonoff;
         unsigned int obtumode;
         int status;

   /* --- flag pour updatelog --- */
   int updatelogindex;
};

#ifdef __cplusplus

#include "alims.h"
#include "controleur.h"

#endif
```

```
void AlimsInit(void);
void AlimsStop(void);
void ControleurInit(void);
void ControleurStop(void);

unsigned int SetBasseTension(int onoff);
unsigned int GetTemperature(int n, double *temp);
unsigned int SetSupply(int on);
unsigned int SetModeTension(int HR, unsigned int binning);
unsigned int SetModeBinning(int HV, unsigned int binning, unsigned int vitesse, int debug);
unsigned int SetTempsExposition(double pose, unsigned int *erreur);
unsigned int GetTempsExposition(double *pose, unsigned int *erreur);
unsigned int SetArea(unsigned short x0, unsigned short y0, unsigned short xb, unsigned short
yb);
unsigned int SetImageSize(unsigned long sx, unsigned long sy);
unsigned int GetImageSize(unsigned long *sx, unsigned long *sy);
unsigned int Start(void);
unsigned int Stop(int stp);
unsigned int Abort(int abrt);
unsigned int SerialDownload(void);
unsigned int Reset(void);
void Initialise(int initls);
unsigned int GetStatusCamera(int est0, int est1);
unsigned int SetPeltier(int on);
unsigned int SetPeltierConsigne(int temp);
unsigned int GetStatusAmplis(int comG, int comBP, int comL, int ampliam, int ampliof);
unsigned int GetStatusObtu(int onoff, int am, int ouvfer);
unsigned int SetAmplisObtu(int onoff, int onoff2, int onoff3, int onoff4, int onoff5);
unsigned int SetDECALAGE(unsigned int n, unsigned int val);
unsigned int SetConfCalcul(short ConfCalcul);
void equilibrer(unsigned int stat_dina);

unsigned int ResetADLINK();


void cemes_updatelog(struct camprop *cam, char *filename, char *comment);

#ifdef __cplusplus
}
#endif

#endif
```

6.4 camtcl.h code

```
#ifndef __CAMTCL_H__
#define __CAMTCL_H__

#define SPECIFIC_CMDLIST \
   {"parameter",        (Tcl_CmdProc *)cmdCemesParam}, \
   {"stat_dina",     (Tcl_CmdProc *)cmdStatique_dinamique}, \
   {"balance",     (Tcl_CmdProc *)cmdBalance}, \
   {"setwin",     (Tcl_CmdProc *)cmdSetwindow}, \
   {"peltON",     (Tcl_CmdProc *)cmdPeltierMarche}, \
   {"peltOFF",     (Tcl_CmdProc *)cmdPeltierArret}, \
   {"settemp",     (Tcl_CmdProc *)cmdCemesSetTemp}, \
   {"gettemp",     (Tcl_CmdProc *)cmdCemesGetTemp}, \
   {"reset",     (Tcl_CmdProc *)cmdRESET}, \
   {"ampliobtu",     (Tcl_CmdProc *)cmdCemesObtu},

/* === Specific commands for that camera === */
#ifdef __cplusplus
extern "C" {
#endif

int cmdPeltierMarche(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdPeltierArret(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdCemesParam(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdCemesObtu(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdCemesGetTemp(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdStatique_dinamique(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdBalance(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdSetwindow(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdCemesSetTemp(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);
int cmdRESET(ClientData clientData, Tcl_Interp * interp, int argc, char *argv[]);

#ifdef __cplusplus
}
#endif

#endif
```