OPSO - The OpenGL based Field Acquisition and Telescope Guiding System

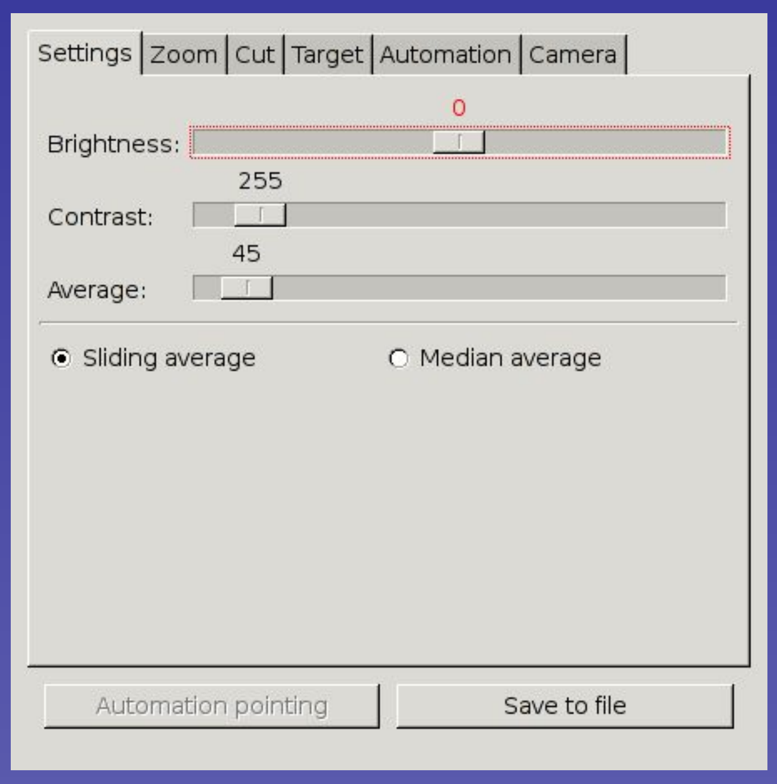
Petr Škoda, Jan Fuchs, Jaroslav Honsa
Astronomical Institute, Academy of Sciences of the Czech Republic
251 65 Ondřejov
skoda@sunstel.asu.cas.cz

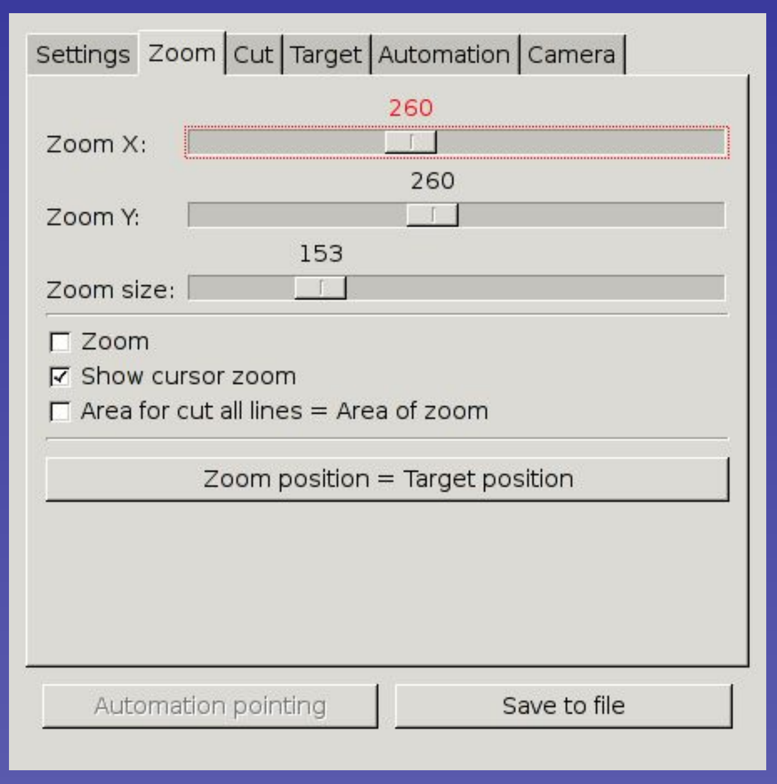
Abstract

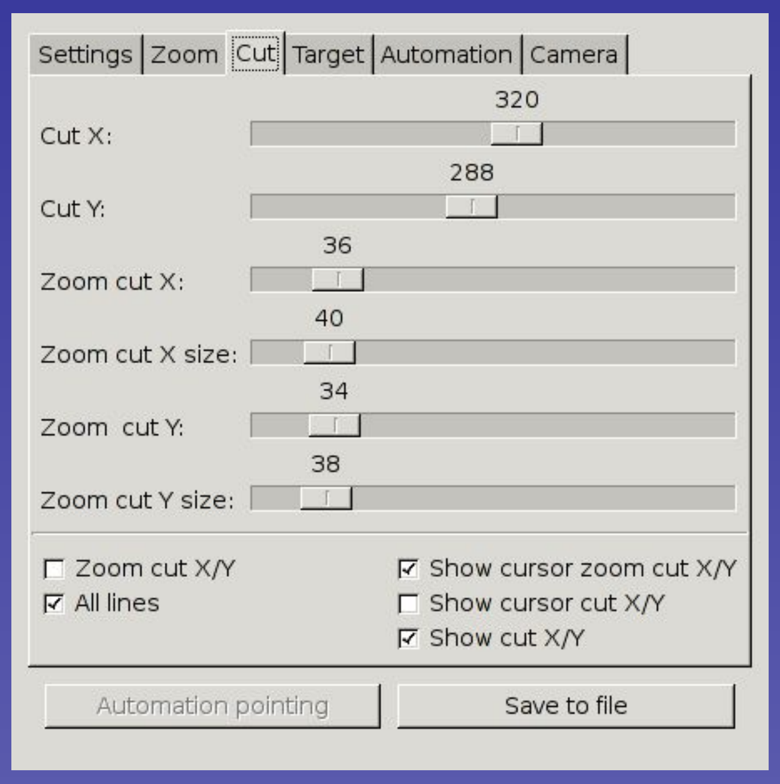
We present OPSO, a modular pointing and auto-guiding system for the coudé spectrograph of the Ondřejov observatory 2m telescope. The current field and slit viewing CCD cameras with image intensifiers are giving only standard TV video output. To allow the acquisition and guiding of very faint targets, we have designed an image enhancing system working in real time on TV frames grabbed by BT878-based video capture card. Its basic capabilities include the sliding averaging of hundreds of frames with bad pixel masking and removal of outliers, display of median of set of frames, quick zooming, contrast and brightness adjustment, plotting of horizontal and vertical cross cuts of seeing disk within given intensity range and many more.
From the programmer's point of view, the system consists of three tasks running in parallel on a Linux PC. One C task controls the video capturing over Video for Linux (`v4l2`) interface and feeds the frames into the large block of shared memory, where the core image processing is done by another C program calling the OpenGL library. The GUI is, however, dynamically built in Python from XML description of widgets prepared in Glade. All tasks are exchanging information by IPC calls using the shared memory segments.

Introduction

The Ondřejov observatory coudé spectrograph uses a image intensified TV camera Proxytronic for viewing the targets at the slit. The direct image obtained from its TV output at the 50 fps rate is, however, very insufficient for viewing the faint objects that are barely visible in the noise.
The OPSO (OpenGL based Pointing System for Ondřejov) enhances the image by digital processing of the images grabbed from TV camera. It allows the averaging of frames, zoom, contrast correction and plot of cross cuts parallel and perpendicular to the slit. In its final stage it should drive the optical elements to compensate for the tip-tilt offset variations and thus providing the telescope with the autoguiding capability.
The system can switch between slit camera and the field viewing camera remembering the position of different targets in the field.


Settings | Zoom | Cut | Target | Automation | Camera
Brightness: 0
Contrast: 255
Average: 45
(●) Sliding average ( ) Median average
Automation pointing | Save to file


Settings | Zoom | Cut | Target | Automation | Camera
Zoom X: 260
Zoom Y: 260
Zoom size: 153
☐ Zoom
☑ Show cursor zoom
☐ Area for cut all lines = Area of zoom
Zoom position = Target position
Automation pointing | Save to file


Settings | Zoom | Cut | Target | Automation | Camera
Cut X: 320
Cut Y: 288
Zoom cut X: 36
Zoom cut X size: 40
Zoom cut Y: 34
Zoom cut Y size: 38
☐ Zoom cut X/Y
☑ All lines
☑ Show cursor zoom cut X/Y
☐ Show cursor cut X/Y
☑ Show cut X/Y
Automation pointing | Save to file

The OPSO design description

The system consists of three tasks running in parallel and communicating through shared memory with IPC calls. The main speed was achieved by using OpenGL exploiting the graphic accelerator of video card.

The grabber task

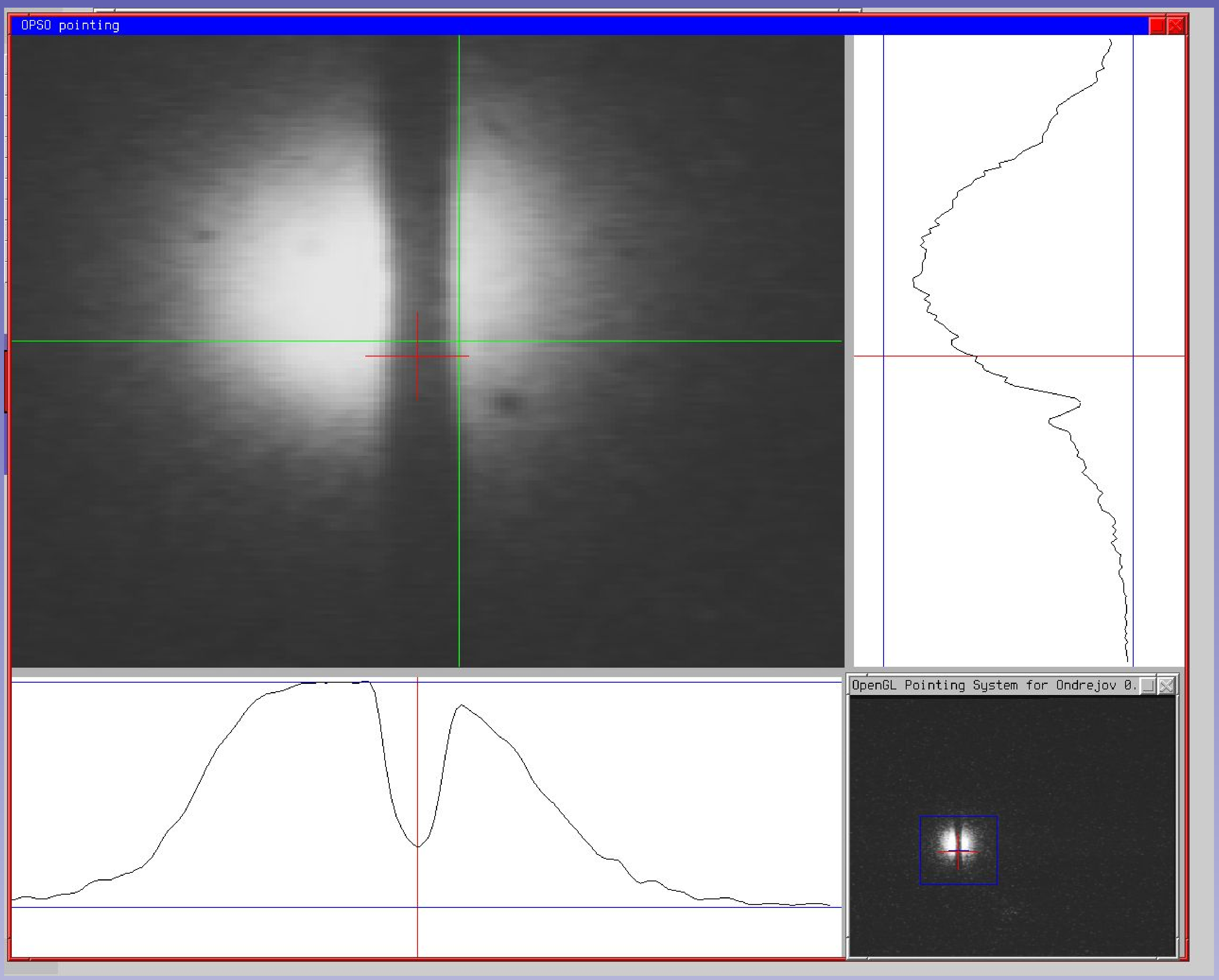
The one process (in C) drives the frame grabber card (using Linux `v4l2` API) feeding the large shared memory buffer and displays every frame using OpenGL (the small window in the lower left corner). The required target position (red cross, can be adjusted differently for various objects) and the zoomed area selected for further processing (blue rectangle) may be set here as well.

Main processing task

The main loop of this process (again in C, calling OpenGL) takes care of the whole digital image processing – it averages (either median of sliding) selected number of images, calculates cross cuts in one selected column or row – green cross (or averages of rows or columns), enhances image visibility by changing contrast or brightness and shows the horizontal and vertical cross cuts in the given intensity range (blue lines). There is a option for masking bad pixels as well.

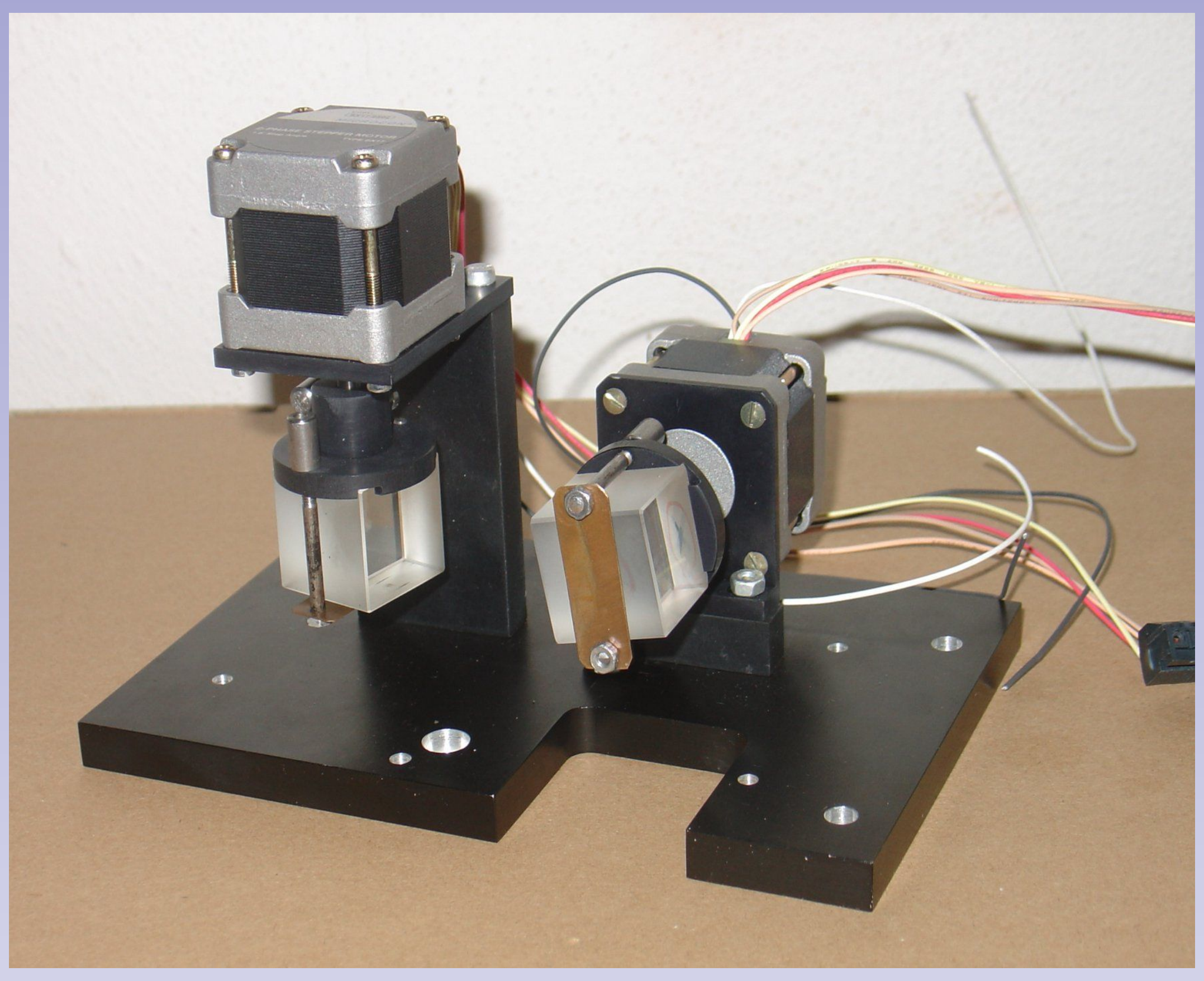
The GUI

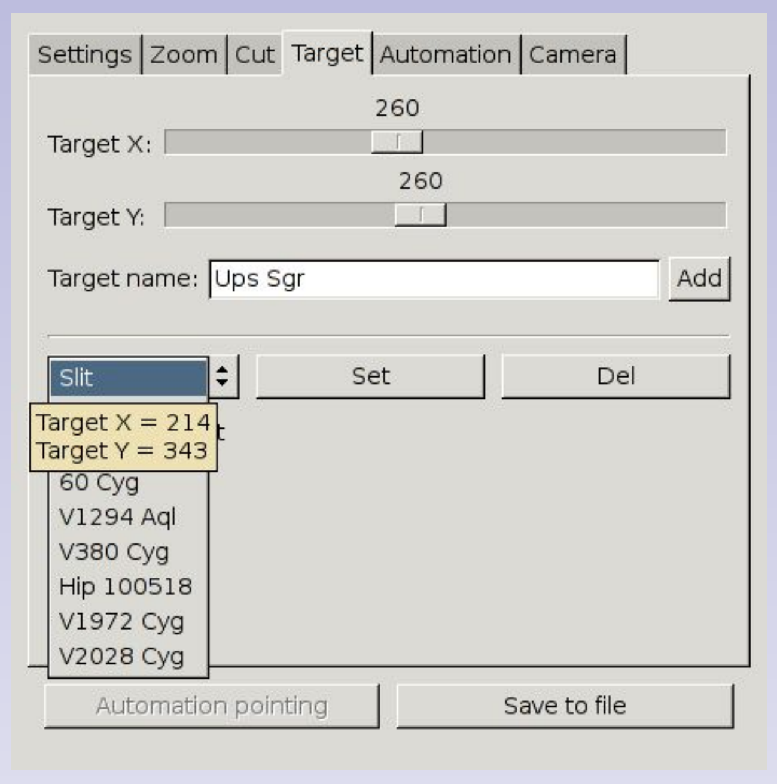
The GUI is running as the Python task. The design of widgets was done in Glade2 and saved into XML file . The Python module `gtk.glade` generates the widget objects dynamically. The change of object parameters (e.g. the number of averaged frames, image contrast or position of the cross) is transferred to the shared memory by the module `pyipc`.

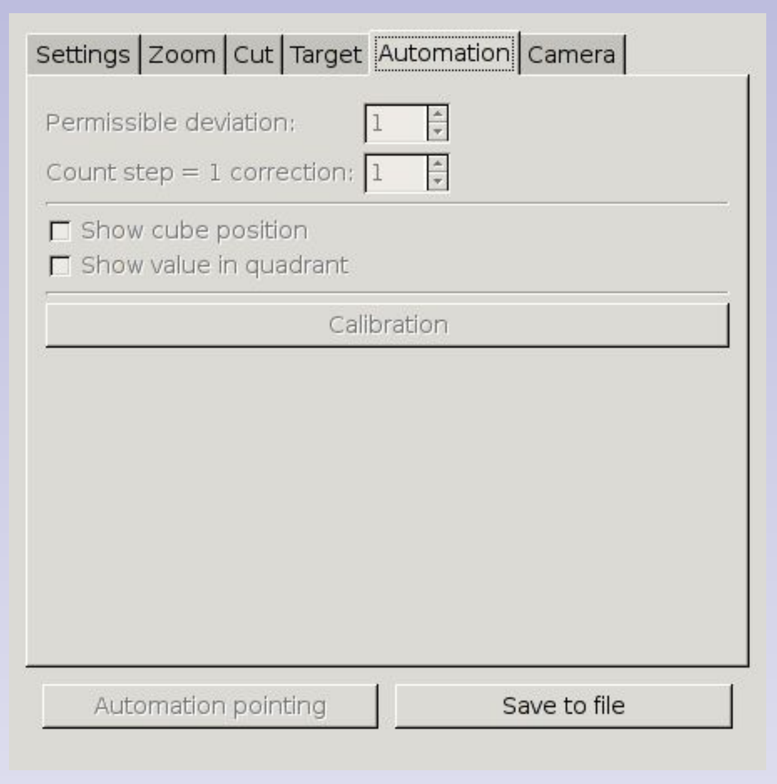


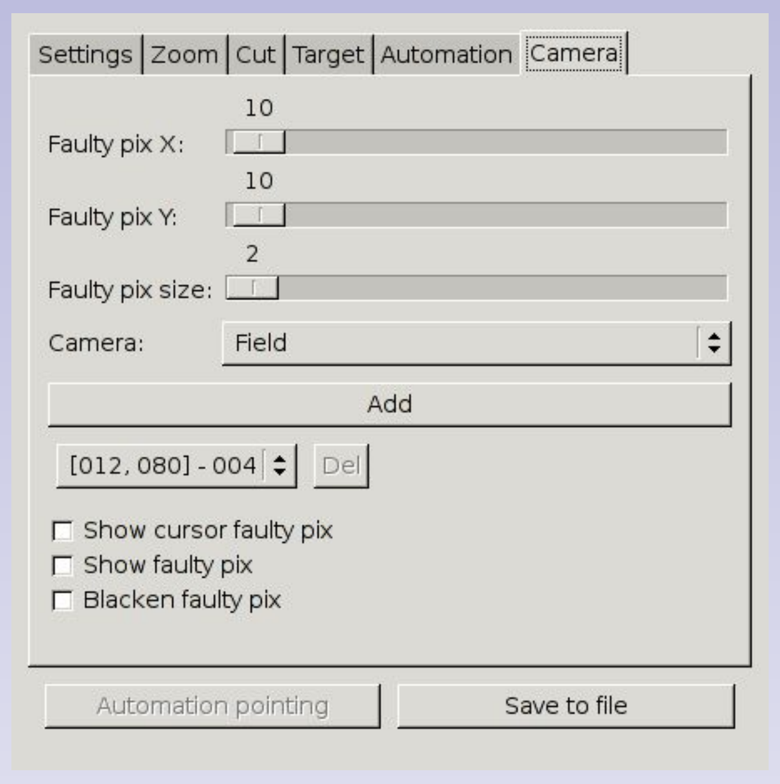
The autoguiding module

Although the system has improved the visibility of faint objects on the slit considerably, the primary goal is the use of digitally processed frames for estimating the center of the stellar image that is hidden behind the slit.
The various algorithms are being considered, the easiest one is the quadrant weighting. It calculates sum of pixel values in every quadrant bordered by target cross. The difference of these values then derives the compensation signal used to drive two perpendicular stepper motors with glass plan parallel plates sweeping the target image in tip-tilt manner. (see photo).
This system is not yet fully working in the present, however.




Settings | Zoom | Cut | Target | Automation | Camera
Target X: 260
Target Y: 260
Target name: Ups Sgr | Add
Slit | Set | Del
Target X = 214
Target Y = 343
60 Cyg
V1294 Aql
V380 Cyg
Hip 100518
V1972 Cyg
V2028 Cyg
Automation pointing | Save to file


Settings | Zoom | Cut | Target | Automation | Camera
Permissible deviation: 1
Count step = 1 correction: 1
☐ Show cube position
☐ Show value in quadrant
Calibration
Automation pointing | Save to file


Settings | Zoom | Cut | Target | Automation | Camera
Faulty pix X: 10
Faulty pix Y: 10
Faulty pix size: 2
Camera: Field
Add
[012, 080] - 004 | Del
☐ Show cursor faulty pix
☐ Show faulty pix
☐ Blacken faulty pix
Automation pointing | Save to file

The memory and speed issues

OPSO runs on a moderate PC (512MB RAM, P4 at 2.4GHz, BT878 based grabber, Matrox 550 accelerated video card) on Linux Debian Sarge (kernel 2.6.8).
The system needs a lot of memory allocated for storing the grabbed images. Their size is 640x576 pixels, the buffer is for 750 frames (15 second averaging)
This requires about 264 MB of shared memory allocated for image buffer. The speed is considerably reduced when using medianing of frames, but for sliding averages it can show cuts and calculate center for every frame (at 50 fps rate).

Conclusions

We have built a powerful telescope guiding system with real-time image processing running on a moderate Linux PC. The most critical part of the image transformation, intensity and contrast changes, zoom etc . is accomplished in OpenGL and thus the power of the graphical accelerator chip on video card is used for most time-critical tasks. The system is very robust due to separation of the GUI handling and the real engine allowing the communication through shared memory and IPC calls.

Presented at ADASS XV, ESAC, San Lorenzo de El Escorial, Spain, October 2-5, 2005