A Complete Robotic Telescope System with CCD Camera Control

R.L. Mutel and E.C. Downey
Dept. Physics and Astronomy, University of Iowa, Iowa City IA 52242

Abstract

We have developed a robotic telescope system primarily for use in a new undergraduate laboratory. The system consists of a 486 PC running Unix/Motif which controls an equatorial mount, focus, dome, filter wheel, and CCD camera. The entire system can be controlled using an X-terminal anywhere on the Internet. In addition, the telescope can be scheduled so that a series of observations can be run sequentialy completely unattended. The software is highly modular and can be easily adopted to a variety of hardware platforms.

1 Introduction

The Department of Physics and Astronomy at the University of Iowa has recently developed a new CCD-based introductory astronomy laboratory. The goal of the lab is to allow students to obtain and analyze *their own images.* In order to allow large numbers of students typically in daytime labs to obtain these images, we have developed a robotic telescope/CCD camera system which can be run using input schedule files. The students submit observing requests using a simple format and receive images and log files within a few days. Interface to the stepper motors and CCD camera is via ISA bus controller cards. The system controls all systems, including mount, dome, and camera. The software runs under a standard Unix/Motif platform, so that it can be operated by any X display on the Internet.

2 Hardware

2.1 Mount, Optics

The mount is a custom designed equatorial with size 34 stepper motors (125 microsteps per step) on each axis. The nomimal slew rate is 5 degrees per second with an RMS blind pointing accuracy ± 1 arcmin. The optical assembly consists of an Astrophysics 18cm f/9 refractor with motorized focus and 10 position filter wheel. All four motor axes are controlled by a OMS PC-34 intelligent motor controller card plugged into an ISA buson a Gateway 486-33DX PC. The dome and shutter are also computer controlled using digital I/O control cards. The mount is equipped with 16 bit (20 arcsecond angle resolution) optical shaft encoders.

2.2 Camera

The camera is a SpectraSource HPC-1 equipped with a TK-215 CCD chip (1024x1024 format,12 micron square pixels) thermoelectrically cooled to -30° C. The camera interfaces to the PC via an ISA bus card. The camera has a read noise $\sim$17 electrons per pixel and a dark current of $\sim$0.6 electrons per pixel per second.

2.3 Computer

The control computer is a Gateway 486-33DX equipped with 32MB memory, and a total of 2.2 GB disk space (540 MB IDE disks; 1.7GB SCSI disk). The PC O/S is System-5 Unix (Novell Unixware) with a Motif GUI. A network of student 486 PC's running DOS/Windows are connected to the control PC by NFS mounted disks using PC-NFS 5.0. This allows transparent exchange of schedules, logs, and images between the control PC and all client PCs.

3 Software

The controlling software is written in C using standard X windows X11R5 and Motif 1.1.4 library calls. The software is highly extensive, i.e. hardware depencies are isolated to low level processes which communicate with user programs via stream pipes. The design also includes one global shared memory segment which is used to maintain a summary of current system status. Most programs read a configuration file at startup which contains hardware specific inofrmation (e.g. radians per bit for each encoder, software hour angle and elevations limits, etc.) as well as user preferences (e.g. default directories for logs and images).

3.1 Hardware-specific Drivers

- **Stepper Motor Control.** The motor control driver sends primitive move and slew commands to an intelligent 4-axis stepper motor controler (Oregon Microsystems PC-34). The card takes care of acceleration and deceleration during move commands, as well as auto-stops for limit switch closures.

- **Digital I/O Interface.** The DIO program reads and writes digital data to two digital I/O cards: a 96 line TTL-level I/O card (CyberResearch DIO-96) which receives optical shaft encoder bits from the mount and dome, and a 24-line I/O (DIO-24) which controls the dome motors.

- **Camera Control.** The camera control as present communicates with a SpectrSoruce HPC-1 CCD camera using *ioctl()* calls. Available functions include setting exposure times, sub-imaging, bin-

atf_sw.overview.ps

ning, and shutter control. Since the HPC-1 is not interrupt-driven, explicit Unix *read()* calls must be made to retrieve each pixel of information.

3.2 Daemons

- **Telescope Daemon** The telescope daemon controls the mount, dome, filter, and focus stepper motors, as well reading the encoders. In a quiescent state, the daemon listens to several stream pipes for generic commands, which are then sent to the hardware driver programs.

- **Camera Daemon** The camera daemon listens to the camera stream pipe for requests to acquire an image and send the requests to the camera driver. The interface is via several *ioctl()* commands.

3.3 X Clients

- **Telescope Scheduler** The telescope scheduler serves both as a manual telescope control and as the main automated control interface. In manual mode, the telescope may be slewed and optionally track an arbitrary position, either in celestial or azimuth-elevation coordinates. The dome may be commanded either to track the telescope or manually sent to a given azimuth. In scheduled mode, the program reads a schedule file containing commands for source selection and camera operation. The syntax consits of a series of *keyword=keyvalue* statements in ASCII format. Source coordinates can be specified explicitly or can be implied by specifying a catalog file which contains the source name(s). Solar systems objects (including nearly 5,000 asteroids and comets) are calculated automatically using orbital elements in a default catalog.

  The program keeps two logs: a global log of all operations of the program, and a schedule-specific log which contains only information relevant to the observing program in progress. Schedules can be concatenated by the telescope operator at the start of an evening's session, so that the program will sequentially observe all queued observing schedules.

  Images are in standard FITS format. Two copies of each image are written to disk: an uncalibrated image is written in compressed format (using STSci's *hcompress* and a calibrated image (dark subtracted, flat fielded) is written to the user area. The dark frame is synthesized by appropriate scaling of an archive thermal frame. The flat field is obtained from the latest archive flat for the appropriate filter found in the archive area. The use may optionally request compression of the user images with a user-specified compression factor.

- **Camera Control/Display** The camera control program can be set to automatically display images as they written by the telescope control program. It can also display images manually, with a few image analysis tools such as a magnifying glass, pixel coordinate reads, sub-image statistics, and adjustment of the intensity histogram. It can also apply calibration (dark/flat) and decompress images.

- **Joystick Control.** The joystick tool is a simple pop-up which allows small telescope moves, either via mouse-activated arrow keys (NSEW) or using sliders.

- **Status Monitor.** The status monitor is a completely passive 'engineering' display of the telescope status. It reads the global memory segment and displays encoder coordinates, mount corrections, refraction corrections, dome position, etc in a continually updated display. This program is used mainly for debugging and maintenance.

4 Future Plans

In the next one to two years we hope to use this software to control a remote telescope in a dry southwestern site over the Internet.

We are grateful to Peter Sauerbrei, Robert Winsor, and Steve Hauser for many hours of assistance in developing this system.

Table of Contents

atf-sw.ps
Version 1.1 December 14, 1994

1. Introduction

This paper will describe the software now in use at the Department of Physics and Astronomy at the University of Iowa for remote and semi-automated operation of a small observatory. This information is intended to guide a computer system administrator who must be aware of the software processes, files and drivers existing on the system, and the programmer who might wish to make modifications to the system.

1.1 Equipment List

The facility includes the following equipment:

- Astrophysics 7" refracting telescope
- filter wheel with stepper motor control
- focus position actuator with stepper motor control
- fork mount with shaft encoders and stepper motors for HA and Declination axes
- dome azimuth encoder and bi-directional AC motor drive
- bi-directional AC motor drive for raising and lowering the dome shutter
- Spectra Source Instruments HPC-1 1024x1024 16bit CCD camera, with electronic cooling
- 24- and 96-port Cyber Research CIO-DIO multichannel discrete controllers
- Oregon Micro Systems PC34 4-axis intelligent stepper motor controller
- ethernet with connectivity to internet and several department IBM PC-class computers via PCNFS
- 66MHZ 486DX computer with 32MB RAM, 2GB SCSI disk
- SVR4.2 UNIX from Unixware (their version 1.1.1 as of this writing)
- X Windows X11R5, Motif 1.1.4 GUI

1.2 Design Goals

Follows is a list of the operational goals for the supporting software.

- The system shall provide a facility whereby undergraduates and local high school students can submit batch observing requests for their individual projects.
- Submission of requests and access to final results shall be possible from UNIX or DOS systems.
- The batch request format shall allow objects to be specified by name from a specified catalog, exposure details, auxiliary demographic information, and provide a simple syntax for repeating and reusing portions of the information.[1]
- The system shall be able to eventually be accessible via batch job requests from anywhere on the Internet.

[1]A full description of the batch file format may be found in the student's guide to using the remote observatory

- The direct-control functions shall all be X Windows clients so that they can be displayed on any system running X Windows on the internet. This provides for completely remote-controlled operation of the observatory.
- The system shall be sufficiently automated that the majority of all observing requests can be executed with little or no supervision from a telescope operator.
- All raw data collected by the system shall be preserved for archival purposes.
- Fully reduced images shall be generated automatically using previously generated calibration fields.
- The creation of the calibration images shall be made as easy as possible, albeit by an experienced operator..
- Manual control of the telescope, camera, dome and auxiliary equipment shall be available for testing and unusual research activities.
- A global log of all system activity shall be maintained automatically.
- An individual log of all activity associated with each observing request shall be created automatically and made available to the submitter along with the reduced images.
- The design shall be extensible, i.e., hardware dependencies shall be isolated to the maximum extent possible to accommodate later changes and growth.
- In particular, the design shall easily accommodate several varieties of CCD cameras, though not simultaneously.

The design to be described below meets these objectives.

2. Software Description

The software consists of several UNIX device drivers, X Windows/Motif client applications, hardware interface daemon processes and a small collection of ancillary processing programs. The X clients are very hardware-independent and communicate generic commands to various daemon processes which utilize the hardware-dependent device drivers to carry out the commands. A special login account, *ccdoper*, provides a convenient way for users to start up all required programs. All software is written in ANSI C to SVR4 operating system interface specifications.

2.1 Development Environment

The robot observatory is developed using SVR4.2 UNIX from Unixware. All code is written to conform to ANSI C and only system calls generic to SVR4 UNIX platforms are used, thus insuring portability is at a maximum to other UNIX systems should it ever become necessary. The special hardware interface boards would be the only area requiring significant re-engineering though the overall design should permit this work to be well partitioned.

The Graphical User Interface is written directly in X11R5 supplemented by the Motif 1.1.4 toolkit, further insuring very portable development runtime display capability. A "GUI builder" was not used.

Follows is a tabular summary of the top-most levels of the directory subtree containing all the development software. At the time of this writing, this subtree resides on *gastro3* and is anchored at *~ecdowney/telescope.*

Directory	Contents or Role
drivers	*hpc, dio* and *pc34* hardware interfaces
telmenus	*telsched, camera, shm,* and *joystick* X clients
daemons	*telescoped* and *hpcd* background processes
tools	*fitshdr, calimage, photom* programs, among others
libastro	code to implement functions for *astrolib.h, fits.h, fitscorr.h* and *telstatshm.h*

2.2 Streams Pipes

The software architecture provides a high degree of separation between the user interface, image processing programs and the underlying hardware. This is accomplished via several bi-directional *streams pipes* (or *fifos*) which transport generic commands to and from the hardware-specific components. The idea is that the GUI processes use only these streams for generic commands and daemon processes interpret and execute the commands using the underlying hardware via the various device drivers.

Another advantage to using streams pipes for the interface between hardware-dependent and independent processing is that they can be read and written using standard UNIX text tools such as *echo* and *cat.* This proves very effective during development. Tests of the daemon processes can be performed without requiring the use of the relatively complex GUI clients as drivers and to verify that the GUI clients are emitting proper commands in arbitrary test scenarios.

A further advantage to streams pipes is how well they cooperate with X Windows clients. The X11R5 programming interface includes a function to listen for activity from arbitrary file descriptors, such as from streams pipes, and activate a callback function when data arrives. This makes it very efficient to write X clients that listen to the required X server in addition to the observatory daemon processes.

These streams are summarized as follows:

Stream File Attach Point	Sample Message Format	Description
/tmp/Dome	AZ: *%g*	move dome to given azimuth
/tmp/TelCTRL	Stop	emergency telescope stop
/tmp/RADecEOD	RA: *%g* Dec: *%g*	slew to given RA/Dec and track
/tmp/HADecEOD	HA: *%g* Dec: *%g*	slew to given HA/Dec and stop
/tmp/Joystick	dHA: *%g* dDec: *%g*	small changes to HA/Dec
/tmp/Focus	Focus: *%d*	move to given displacement
/tmp/Shutter	Open	open or close the dome shutter
/tmp/Filter	Filter: *%c*	set given filter location
/tmp/CameraFile	*filename*	file to display with camera program
/tmp/Camera	Expose *x+yxwxy bxxby secs* ...	take an exposure

In addition to these commands, all streams support a string of *Reset.* This will inform the daemons to restore all underlying hardware to a known, quiescent state and reread the various configuration files.

Most streams define a return message of the form: *n message*, where *n* is 0 for successful completion and positive for various errors; *message* is a descriptive ASCII error message.

See the description of each daemon for more information about the commands over each stream.

2.3 *telstatshm:* Global Shared Memory

The design includes one shared memory segment. It is used to maintain a summary of current system status. This is referred to as the *telstatshm* segment because the C struct that defines its memory layout template is known as `telstatshm`. See *telstatshm.h* in the *libastro* directory for the definition of each field in this structure.

Several processes write information to this globally writable segment but, by agreement, only one process may ever write any one field. This information serves as the "engineering and status" data of the system. The X client *shm* and the UNIX processes *telshow* and *engshow* display the contents of this shared memory segment, as described elsewhere.

2.4 *ccdoper* login

The telescope operator logs into the UNIX system using the *ccdoper* account which runs the *csh* script *startTelescopeControl.* This starts all daemons and X client processes, some in an iconized state to help reduce initial screen activity. This login functions within the general Unixware desktop model, and so all tools available from this facility are also available as desired, just as with any UNIX user account. In particular, the correct way to end a telescope operation session is to Exit the desktop.

The *ccdoper* directory can be loaded with fresh copies of all software from the development area using the *csh* script *ccdoperUpdate.* This script is located in the top-level source directory. The script saves all existing executables and supporting files in the directory *old* before loading new copies. The installation script must be run as user *ccdoper* from its home directory.

The default directory arrangement for the *ccdoper* login is described in the following table:

Directory	Contents or Role
archive/config	configuration files; read once on startup or "Reinitialize" command
archive/Xconfig	X resource files for all clients
archive/images	compressed, calibrated CCD images
archive/logs	activity logs from each major software component
archive/calib	bias, thermal and flat CCD calibration reference images
archive/catalogs	astronomical object catalogs to which schedule files (*.sch) may refer
user/images	uncompressed, calibrated CCD images (as per *archive/calib*)
user/logs	activity logs on a per-schedule file basis
user/schedin	observation requests in the form of *.sch files
bin	all executable files used during ccdoper operation
old	copies of all previous files when reloaded via ccdoperUpdate script

The *user/schedin* directory is mounted read-write for easy access by students using department PCs running PCNFS. The *user/images* and *user/logs* directories are available read-only. This is to prevent accidentally erasure of data since all students have access to these directories. Note this implies a degree of administrative attention is necessary to clear out these directories occasionally.

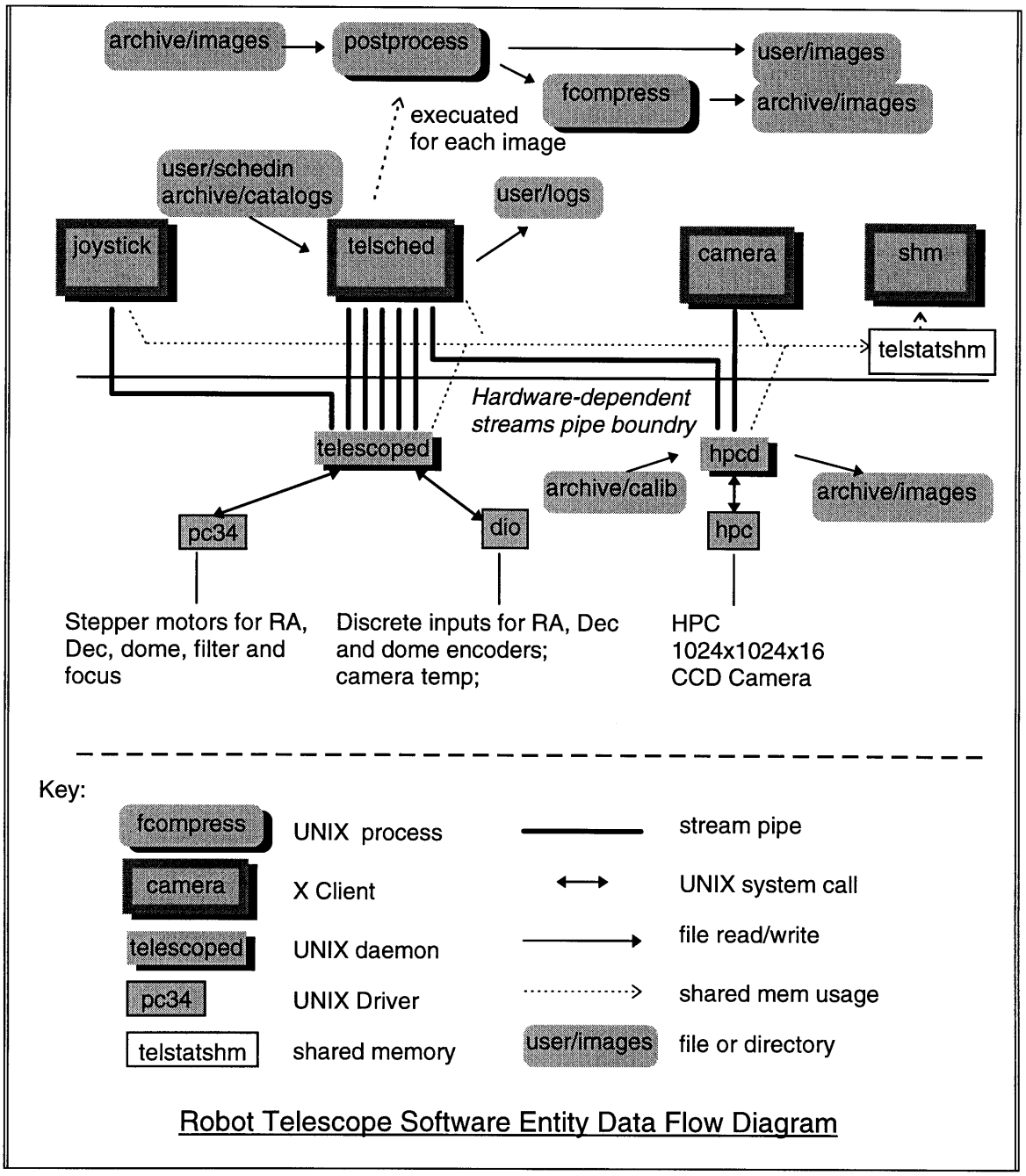
Observing requests are submitted by creating a schedule request in the form of a *.sch file in the *user/schedin* directory. Results appear later in the *user/images* directory with supporting details in the *user/logs* directory on a per-schedule basis. Compression is controlled by the *postprocess* shell script, described elsewhere.

2.5 *ccdoper* Runtime Software Entities

This section begins with a diagram showing the overall data flow relationships among all major software entities used during the general *ccdoper* operating environment. Following the diagram is a description of each entity, including a discussion of the control flow within and between the entities.

All relative path names in the diagrams and descriptions that follow refer to the base of the directory subtree containing the robot telescope development directories. All executables reside in *~ccdoper/bin* unless otherwise specified.

2.5.1 Entity Relationship and Data Flow Diagram



Robot Telescope Software Entity Data Flow Diagram

2.5.2 UNIX Drivers

The following sections describe the UNIX drivers. These serve as the underlying hardware-to-software interfaces.

2.5.2.1 *pc34*: Stepper motor driver

This driver is in charge of the PC34 intelligent stepper motor controller interface from Oregon Micro Systems, Inc..

The pc34 driver provides a buffered ASCII interface to the pc34 controller. Since all motors are accessed through one stream of ASCII messages, the driver enforces that only one process may open it at a time. The message format of the pc34 is line oriented and the driver capitalizes on the presence of newline characters to provide buffering boundaries and to allow servicing the controller from within the interrupt handler.

Access to the driver is via the special file */dev/telescope/pc34.* The source code to the pc34 is in the directory *drivers/pc34*. The bus I/O address is specified in the file *System*; see the comments therein for complete details. To build the *pc34* driver, change to the *drivers/pc34* directory and type *make build*. To install the driver into the UNIX operating system type *make install.* You will need *root* privilege to install the driver.

The directory *drivers/pc34/tools* contains a simple test process, *pc34,* that can be used to directly control the pc34 interface from the keyboard. To build this process, type *make* in this directory. To run it, type its name; it has no arguments. Once started successfully, all commands and replies will be as advertised in the hardware documentation for the pc34 controller. Note that since the driver enforces single process access, this tool may not be ran while any other process is active that has the pc34 driver open (such as *telescoped*, described elsewhere).

Also in the *tools* directory is a program named *pc34stat*. This program just reads and displays the current values in the *done, stat* and *ctrl* registers on the PC34 interface card.

2.5.2.2 *dio:* DIO24/96 Discrete controller

This driver is in charge of the CIO-DIO cards from Cyber Research, Inc. . These cards come in three models that provide 24, 48 or 96 separate lines of input and/or output. Each card contains one Intel 8255 for each set of 24 ports. See the documentation for the DIO cards and the 8255 for more details on the exact operation.

The source code to the pc34 is in the directory *drivers/dio*. The number of cards installed in the system, their model and I/O address configurations must be set up correctly in several files. The *README* contains complete instructions for specifying the driver configuration. The configuration as of this writing is one 24-port and one 96-port card. Access to the drivers for these boards is via the special files */dev/telescope/dio24* and *dio96*, respectively. If the board configuration changes, update the description files as per the *README* and rebuild and reboot the system using *make build* as *root.*

The driver is capable of testing each board when the system is booted to insure it matches the software configuration file. However, in order to do this meaningfully the boards are temporarily placed into an output mode. Since the driver has no knowledge of the usage assigned to the various ports, it can not avoid doing potentially undesired commands as a side effect of its tests. Therefore, these tests are currently turned off but may be turned on at a later time as desired by recompiling with the *TESTCHIPS* conditional flag defined and reinstalling the driver.

The directory *drivers/dio/tools* contains a simple test program, *dio,* that can be used to directly read and write the data ports of any one dio board directly from the keyboard. To build this process, type *make* in this directory. To run it, type its name with the special filename of the desired board as the first argument. For example, to access the dio24 board, type:

```
dio /dev/telescope/dio24
```

Once started successfully the program enters an infinite loop of displaying the current settings of each port and allowing the operator to enter an alternate value. New bit values are specified on one line as a complete set of new bit values in hex. Only those bits that are to be changed need be entered, i.e., leading blanks serve only as place holders and the line may end early. See the documentation for the dio boards and the 8255 chips they use for the correlation between these bits and the hardware ports.

Another program in the same directory, *diomode*, allows one to directly read or write all 4 registers of any one 8255 on any DIO card. This is most useful to change the input/output modes of a card. Both input and output are in hex. The command arguments the name of the special device; r or w (for read or write); chip number; and, if writing, four hex numbers to specify the desired values for each of the four registers on the given chip.

For example, follows is the usage for displaying the registers of the first 8255 on the DIO-25:

```
diomode /dev/telescope/dio24 r 0
```

For another example, follows is the usage for setting the registers on the second 8255 on the DIO-96 so that all ports are input except for the upper half of the C port:

```
diomode /dev/telescope/dio96 w 1 0 0 0 93
```

Both of these programs may be run at any time in any combination with themselves or other processes.

2.5.2.3 *hpc:* HPC CCD Camera driver

This driver accepts fairly generic commands to describe a CCD camera and image acquisition board. It controls the HPC camera interface hardware to accomplish taking one exposure and sending the resulting pixels back to the user-level process. The driver is accessed via the special file */dev/telescope/hpc.* The source code for the *hpc* driver is in *drivers/hpc.* The I/O address for the board is specified in the file *System*; see the comments therein for the exact format. If the address changes, rebuild the driver with the command *make build* and install it into the UNIX kernel with the command *make install.* Installing a new driver requires *root* privilege.

Exposure parameters are set up using an *ioctl* using the structure defined in *hpc.h.* Image data can then be obtained using a standard UNIX *read* system call. Both blocking and non-blocking reads are implemented. The driver also implements the *chpoll()* DDI entry point to support the user-level UNIX *poll()* system call interface for POLLIN and POLLRDNORM activity. The *hpcd* camera daemon uses the *poll()* approach and so has received the most testing.

Unfortunately, the HPC camera interface supports neither DMA nor interrupts for transferring image data. Furthermore, several explicit I/O reads and writes must be done to retrieve *each pixel* and must be done so as rapidly as possible in order to retrieve the best possible image. This requires the driver to stay in the kernel while the entire image is being read to perform manual polling loops from the camera. Staying within a driver for extended periods directly conflicts with UNIX driver model constraints. Three driver parameters can be tuned to trade off the time spent in an uninterrupted polling state and the overall time to retrieve an image. These are NROWBUF, MAXROWBUFS and UIODELAY. See the comments in *hpc.c* for an explanation of the effects of each of these parameters. Rebuild and install the driver as described above if changes are made to these constants.

The directory *drivers/hpc/tools* contains a stand-alone image capture program, *hpctest.* This program takes one exposure and writes it as a *FITS* file to *stdout.* The command arguments are a description of the image offset and size in the form "X+YxWxH"; a description of the desired binning in the form "BXxBY"; and the exposure time, in seconds. The following example takes a 10 second exposure of a 1024x1024 image starting at [0,0] with 2x2 binning and places the resulting image a file *file.fts.*:

```
hpctest 0+0x1024x1024 2x2 10 > file.fts
```

2.5.3 Daemon processes

The role of several processes in this architecture function as interfaces between the GUI client programs and the hardware drivers. These processes live indefinitely and are call *daemon* processes in the usual UNIX vernacular. The various daemons listen to the generic hardware-independent commands being issued from the GUI clients on the stream pipes and issue the necessary hardware-dependent commands to the drivers. They also serve as focal points for maintaining overall state information and can perform repetitive autonomous monitoring functions.

It is important to differentiate the particular daemon implementations described in the following sections from the general isolation that is provided by the streams pipe strategy. The syntax of the messages flowing over the streams pipes do not presume any particular implementation or process organization of the listening and responding entities. The X clients using the streams pipes do not even know if there is one process per streams pipe, one process listening to them all or any other combination. In testimony, the number of deamons *has* changed more than once during the past several months unbeknownst to the GUI clients.

Follows is a description of each daemon process and the streams pipes and message formats it supports..

2.5.3.1 *telescoped:* telescope control daemon

The telescope daemon, *telescoped*, is primarily in charge of pointing the telescope. It also runs the focus motor and filter wheel motors, operates the dome and shutter motors, and supports fine pointing adjustments via the joystick emulation. It has this much responsibility because the stepper motor driver can only be opened by one process at a time and most of these actions are affected via stepper motors.

The source code to *telescoped* is in *daemons/telescoped*. It listens to several streams pipes for generic commands. It uses the pc34 driver to control stepper motors. It uses the dio drivers to read encoders and perform other hardware specific duties to carry out the commands. When first executed, and whenever a *Reset* command is received from the */tmp/TelCTRL* stream pipe *telescoped* reads several configuration parameters from a file named *telescoped.cfg*, assumed to be in a directory *archive/config.*

The main loop of *telescoped* is in an infinite loop listening to several stream pipes. Based on the file descriptor of the pipe with an incoming message, it is dispatched to a separate function for interpretation and execution. Several variables maintain state while commands are in progress.

The pointing error is broken down into three components: refraction, mount, and joystick. The "ideal" position is that of a naive observer. It is the position at which he or she would ostensibly wish to observe. The "commanded" position is that which is sent to the actual telescope stepper motors. The commanded position is the sum of the ideal plus the three error components. The refraction correction is a simple computation based on the elevation. The mount correction is based on a model computed from many test images of objects at known coordinates. The joystick correction is supplied by an stream pipe and discussed elsewhere.

2.5.3.1.1 configuration files

This daemon reads three configuration files when it is first executed and whenever it receives the command "reset" from */tmp/TelCTRL*. The names of these files are *telescoped.cfg, focus.cfg,* and *domed.cfg.* These should each be located in the directory *archive/config* relative to the directory from which *telescoped* was executed. These files change fairly often and are well commented so the reader is referred to read them directly for the latest configurable parameters.

2.5.3.1.2 */tmp/HADecEOD*

Telescoped listens to */tmp/*HADecEOD for a specific HA and Dec (assumed to be relative to the epoch-of-date) to which to slew and stop. The format of the command is:

HA:*%g* Dec:*%g* each angle is in radians

When a properly formatted command is received, *telescoped* reads the current HA and Dec encoders and computes and initiates the appropriate stepper motor motion command to achieve the desired position. The encoders are read a few times per second to monitor progress. If the total angular error between the read position and the target position ever starts to increase a new set of motion commands are computed and initiated. This can happen due to several influences on the telescope mount especially for large slewing motions. Once the current position is acquired the motors are commanded to halt, a reply is sent to the */tmp/HADecEOD* stream pipe and *telescoped* resumes its quiescent state.

2.5.3.1.3 */tmp/RADecEOD*

The stream pipe */tmp/RADecEOD* is similar to */tmp/HADecEOD* except that it used to send a position to acquire and track. The format is:

RA:*%g* Dec:*%g* each angle is in radians

The response to a command from is rather similar to that if the HADec stream. The major difference is that once the specified position is acquired the HA motor is commanded to maintain a sidereal tracking rate. The acquisition slewing algorithm is as above. At this time no error checks are performed on the telescope position once tracking has begun, i.e., it is strictly open-loop. The code does contain sufficient state information and structure to support a closed-loop tracking scheme implementation if desired at a later time.

2.5.3.1.4 */tmp/Filter*

Commands to set the filter wheel are received via */tmp/Filter* stream pipe. This command format consists of a single character that uniquely names each filter position. Details of the filter implementation, including the mapping of filter name characters to motor commands, is n the file *filter.c*. No direct feedback is available from the hardware to detect when the desired filter is in fact in position, but the code attempts to compute a time delay based on motor speed and distance to moved that has proven adequate in practice. For maximum accuracy and to eliminate cumulative errors, the filter wheel stepper motor is always driven back to the home position between each change of filter. Furthermore, the filter wheel home position microswitch works best when the wheel rotates in one direction. For this reason the implementation attempts to always rotate the wheel in the same direction.

2.5.3.1.5 */tmp/Focus*

Commands to set the focus are received via the */tmp/Focus* stream pipe. This stream is in an early development stage. See the code in *focus.c* for the current state of affairs. At the time of this writing, the command consists simply of the number of relative stepper motor steps that are sent directly to the appropriate motor. No feedback is available for command completion so a fixed time delay is implemented to predict command completion that, in general, is too short. The longer term plans call for establishing a mapping between filter selection and focus and have the focus follow the filter automatically.

2.5.3.1.6 */tmp/Dome*

The */tmp/Dome* stream pipe can be used to command the dome to a specific azimuth or to tell *telescoped* to automatically rotate the dome to maintain the slit always in front of the telescope.

The dome is controlled by a simple bi-directional AC motor. The motor may be commanded to rotate in either direction using lines on the DIO discrete card. The motor is always activated to drive the dome in the shortest direction to the new azimuth *without going through north.* The latter is to avoid forever wrapping cables that may be connected to dome equipment.

The dome may be commanded to a specific azimuth by sending a command of the form:

AZ:*%g* azimuth angle, radians east of north

When a new azimuth is commanded the target azimuth is stored and a dome state variable is set to indicate the dome is rotating. While the dome is in this state the main loop of *telescoped* reads the dome azimuth encoder and compares it with the target position once each second. The motor is commanded to remain rotating in the desired direction until they agree to within a certain error. If the dome overshoots, the motor might be commanded in the reverse direction as required. The dome position error may be specified in the configuration file. When the error is less than the limit a reply is issued to the */tmp/Dome* stream pipe.

The dome may also be commanded to simply follow the telescope around. This is done by sending the command "Auto" on the */tmp/Dome* stream pipe. This condition is saved in a state variable. When in auto mode, *telescoped* reads the value of the dome azimuth encoder approximately once per second. It also reads the pointing direction of the telescope and computes the required dome azimuth for an unobstructed view. This computation is necessary because the telescope optical axis is not through the center of the dome sphere. The computation is performed by the function *domeParallax()* in *domectrl.c* and is based on fitting measured data to a model. As long as the dome error is larger than the configurable error limit the dome motor is rotated in a direction that should reduce the error. As with manual commands, care is taken to never rotate the dome through due north. This automatic mode can be turned off by sending an explicit azimuth, as above, or by sending the command "Manual" via the */tmp/Dome* stream pipe.

2.5.3.1.7 */tmp/Shutter*

The dome shutter may be controlled by sending commands to */tmp/Shutter.* The commands are "Open" and "Close" to open and close the shutter respectively. There is no feedback from the shutter so a simple timer is used to estimate when the shutter motion is complete and send a reply to the sender in the streams pipe.

The bi-directional AC motor which controls the dome shutter is controlled via two lines on the DIO card, one for each direction of motion. Power is supplied to this motor by a set of wiper contacts on the inside of the dome. These only make contact when the dome is at a certain azimuth. *Telescoped* automatically positions the dome at the required azimuth when commanded to move the dome shutter. If the dome is in Auto mode, it will return to its proper azimuth when the shutter has completed its operation. These actions are all coordinated within *telescoped* by using internal state variables; the process which issues the shutter commands does not need to know anything about the implied dome azimuth changes required to operate the shutter mechanism.

2.5.3.1.8 */tmp/Joystick*

The pointing direction may be modified slightly at any time by sending an offset in HA and Dec via the */tmp/Joystick* stream pipe. The format of the command is as follows:

dHA:*%g* dDec:*%g* desired change in HA and Dec, rads, += current values

When a correctly formatted command arrives the offsets are saved and all error computations use the new values. If the telescope is currently idle or is tracking open-loop *telescoped* will basically force a new pointing cycle to respond to the new "error".

2.5.3.1.9 */tmp/TelCTRL*

Any data whatsoever, regardless of content, arriving on this stream pipe causes the telescope to be stopped immediately. If the message is "reset" it also causes the configuration file, *telescoped.cfg,* to be read again.

2.5.3.2 *hpcd:* HPC CCD Camera daemon

This UNIX daemon process listens to */tmp/Camera* for requests to acquire an image from the HPC CCD Camera. This daemon is used by *telsched* to acquire images from schedule files and by the manual *camera* X client. The daemon communicates to the *hpc* driver via the special file */dev/telescope/hpc.* The interface to the driver is via several *ioctl* commands. See the description of the *hpc* driver elsewhere. Refer to the HPC documentation for full details of the camera operation.

The idea is that to support a different camera only this daemon would change, or more likely, a different daemon would be written for each camera and one started depending on which camera was to be supported. The results of each exposure is a complete FITS file to which bias, thermal and flat corrections have been applied.

2.5.3.2.1 configuration file

This daemon reads a configuration file each times it is executed and also whenever it receives the command "reset" on */tmp/Camera.* The name of this file is *hpcd.cfg* and it should resize in the directory *archive/config* relative to the directory from which the daemon process was executed. This file changes on occasion and it well commented so the reader is referred to read it directly for the latest set of configurable camera parameters.

2.5.3.2.2 */tmp/Camera*

The format of the command from */tmp/Camera* includes many parameters on several lines, as follows:

```
Expose X+YxWxH BXxBY duration shutter filename
       filter  RA  Dec  HA Altitude Azimuth
       source
       catalog
       comment
       title
       observer
       calibdir
```

The first two arguments specify the image offset, size and binning. The duration is a real number and is in seconds. The shutter value may be 1 or 0, the latter meaning to keep the shutter closed as when making a dark calibration frame. The filename specifies where the resulting fully-corrected FITS image should be stored. The remaining fields except the last are supplied only so they may be included in the header of the resulting FITS image. The calibdir argument specifies the directory in which the bias, thermal and flat correction images should be found.

The streams pipe also supports the commands *Reset* and *GetConfig.* The latter responds with matrix size and bits-per-pixel in the format *WxHxN.*

The source for the *hpcd* daemon is in *daemons/hpcd.* The daemon can be rebuilt by typing *make.*

2.5.4 X Clients

All of the graphical user interface in this system is implemented using the *Motif* widget set from the Open Software Foundation, running on the *X Windows* protocol from MIT. Unless otherwise stated, the sources for all of the X clients are within the *telmenus* subtree.

Follows is a description of each of the user interface programs. In general, see the student documentation that is being prepared for more information on the operation of these programs.

2.5.4.1 *telsched:* Main Telescope Control User Interface

This X client serves as both the manual telescope control and as the main automated control interface. The source code is in *telmenus/telsched.* On initial execution, *telsched* reads the file *telsched.cfg* for various initialization constants. This file is expected to be in the *archive/config* directory.

In manual mode, the telescope may be slewed to arbitrary HA/Dec or Alt/Az positions or be set to track an arbitrary RA/Dec. The dome may be positioned at an arbitrary azimuth. The filter and focus may be adjusted as desired. The shutter may be opened and closed. Special convenience controls also allow for direct positioning of the telescope to *service* and *stow* positions.

Telsched can also read in one or more batch request files, i.e., the **.sch* files. It breaks these files out into individual observations in a scrolled window region. If the schedule file did not specify a starting time for the observation then the current sky position of each observation is continuously displayed, as well as the specific characteristics of the exposure. Each field may be manually edited and overwritten. If the starting time as specified or has been established by virtue of a successful sort operation, then the information displayed is correct for the time the exposure will begin.

Once one or more schedule files have been read, a single button press can instruct *telsched* to do everything required for the first observation in the list that is currently feasible, i.e., up. One such observation may be made, or a completely automated operation can be commanded that will move through the list one at a time and perform each observation until the list is exhausted.

Telsched performs all operations by sending commands through the appropriate streams pipes to the control daemons described elsewhere. Image compression is performed by invoking the *postprocess* shell script after the camera indicates an image has been acquired. A composite log of all activities is written to *archive/logs/TelSched* and logs for each schedule file are written to *user/logs.*

2.5.4.1.1 configuration file

This client reads a configuration file each time it is executed and also whenever it is told to perform a "Reset HW" command from its control menu. This command also tell telsched to send the appropriate command to each daemon so they read their configuration files as well.

The name of the configuration file is *telsched.cfg* and it should reside in the directory *archive/config* relative to the directory from which the client was executed. This file changes on occasion and it is well commented so the reader is referred to read it directly for the latest set of configurable camera parameters.

2.5.4.1.2 *telsched* sorting function

After one or more schedule files have been read, *telsched* can sort the resulting set of individual observations into an order that attempts to makes maximum utilization of a night of observation. A description of the sorting algorithm follows.

Recall that each observation is required to have specified a sky position to observe (either by reference to a catalog object or by giving explicit RA and Dec values) and an exposure duration. It may also optionally specify a starting time for the observation.

The 24 hour period to be sorted is represented as a collection of time slots. These slots may be arbitrarily small and are currently configured at one minute each. Each slot is initialized as unused. The idea is to work through the set of observation requests and assign them to one or more slots, depending on their duration.

The first step in filling the slots is to assign observations which specify a particular starting time immediately to their slots in a first-come-first-serve order. Collisions are averted by sliding the start time of an observation away from its desired starting time in the direction which causes it the least error. No checks are made for unreasonable starting times; it is felt that the system should not overrule any explicit request.

The follows discusses how the observations are assigned for which no particular starting time as specified. In all the algorithms, observations are never assigned to time slots when the sky position is not up or which would violate local equipment constraints. Once an observation has been successfully scheduled it is not eligible for reconsideration by a subsequent selection step.

The first step in assigning floating observations is to search for all observations which are deemed "evening" observations. These are at sky positions which transit before dusk but which set after dusk. The idea is to observe these objects first, beginning at dusk because they are getting ready to set, starting with the one which will set first. The algorithm sorts the unassigned observations according to increasing set time and merges them into the slot list beginning at dusk. Conflicts are resolved by letting the observation slip to a later time.

The next step is basically the opposite. All observations are collected which are deemed "morning" observations, that is, those which rise before dawn but which transit after dawn. The idea here is to observe these objects as late as possible in the night in order to let them rise as high as possible. The algorithm sorts the heretofore unassigned observations according to decreasing rise time and merges them into the slot list beginning at dawn. Conflicts are resolved by letting the observation slip to an earlier time.

Note that in assigning an observation as evening or morning, it is often the case that a sky position behaves according to both criteria. In these cases, the observations are assigned to the category which gives them the most flexibility, that is, the one in which they remain the longest between dusk and dawn.

The next step is to find all heretofore unassigned observations which transit some time during the night, that is, between dusk and dawn. These observations are scheduled at their time of transit. Conflicts are resolved by choosing the closest available time slot in either direction of the transit time.

Finally, observations of sky positions which are circumpolar are assigned. Those which transit at night are assigned as above. Those which transit nearer dawn than dusk are assigned the latest slot available before dawn in an attempt to observe them as high in the sky as possible. Similarly, all remaining observations must transit nearer dusk than dawn and are assigned the first available slot after dusk.

Observations which are *still* not assigned are either at sky positions which are only visible during the daytime or the night was completely subscribed. These objects remained unassigned and *telsched* displays them with their observing eligibility disabled.

Note that this strategy implements, in effect, a priority mechanism which favors observations that specify an explicit starting time, followed in order by evening, morning, night and circumpolar observations. Future work might factor these into a "charge" assessed for each class of observation.

2.5.4.2 *camera:* Manual CCD Camera User Interface

This X client is both a passive FITS file viewer as well as a means to directly operate the CCD camera. This client is also designed to allowing making CCD calibration files for bias, thermal and flat corrections. It can not be used to run the camera while automatic image acquisition is in progress via the operation of *telsched.* Several basic image manipulation functions are provided, include a magnifying glass, an area of interest, and computation of several basic pixel statistics.

This client listens for file names to display on */tmp/CameraFilename.* This allows it to automatically display an image and is used for just that purpose by the *postprocess* script. This feature may be disabled via a menu selection.

The source code for this client is in *telmenus/camera.*

2.5.4.3 *shm:* Passive Engineering Data Display

This X client attaches to the *telstatshm* shared memory segment and displays the value of some of the fields once each second. This client has no user input capabilities. Its function is to provide the "engineering" data display of the system.

A brief description of each field may be found in the comments in the header file that defines the structure, *telstatshm.h* in the *libastro directory.* The fields are shown with proper formatting for easy reading. For example, the current hour angle reading from the encoders is displayed in HH:MM.D format, not raw encoder values.

The source code for this client is in *telmenus/shm.*

2.5.4.4 *joystick*

This X client displays four arrows and two slider controls. It generates small values of HA and Dec offsets onto the streams pipe */tmp/Joystick.* The intent is for these offsets to modify the pointing direction of the telescope as though a joystick were making small manual adjustments to the *ideal* pointing position.

At the present time these offset commands are read by the *telescoped* daemon which it uses to update terms in its correction from ideal to commanded position. It thus has the effect of moving the telescope as desired, but it updates the *commanded* telescope position rather than the *ideal* direction as it should. This causes the logged positions to ignore joystick corrections.

The source code for this client is in *telmenus/joystick.*

2.5.5 Auxiliary processes

Some of the work performed in parallel to the actual control of the observatory hardware is done by separate UNIX processes. These extra programs are described below.

All programs generate a usage summary if invoked with incorrect arguments or *-help.*

2.5.5.1 *fcompress* and *fdecompress*

These are programs which compress and decompress FITS files using the "H-transform" method developed by Rick White, white@stsci.edu, of STScI. These are actually *csh* shell scripts which invoke the *hcomp* and *hdecomp* compression and decompression programs on FITS format files, respectively. This code is in *tools/hcompress* and is used by permission by the author.

Basically, the H-transform is a compression scheme which tries to discard and hence avoid compressing the background sky noise contained in astronomical images. It also has the unusual property that images decompress the most "useful" information first. This can be used as the basis for an effective telecommunications protocol since the user can watch the image build up gradually and stop it when it looks "good enough".

These scripts and the corresponding C programs reside in */use/bin/local* for general access outside the context of the *ccdoper* operating environment.

Fcompress is used by *telsched's postprocess* script to compress image data into the archive/images directory. It's usage is as follows:

```
fcompress -s scale files ...
```

The given files are compressed *in place* using the supplied (and required) scale factor. The FITS header is modified, using the *fitshdr* program, to include an integer field HCOMSCAL, whose value is the scale factor, and the string field HCOMSTAT which is set to HCOMPRSD.

Fdecompress accepts the names of compressed FITS files and decompresses them, also *in place,* and changes the string field HCOMSTAT to UNHCOMP in the header to record that the file has been decompressed.

2.5.5.2 *fitshdr*

This program allows one to print, add, delete or replace any and all header fields in one or more FITS files. The usage is as follows:

```
fitshdr [-d name] [-i name value] [-l name value] [-r name value] [-s
name value] [-c name value] [-p] [files ...]
```

Name refers to the name of a FITS header entry; *value* refers to a replacement value. These must appear as one argument to the program so be sure to enclosed them with quotation marks if they included embedded whitespace or other characters interpreted by your shell. -d deletes the field with the given name; -i adds (or replaces) an integer field; -l a logical field; -r a real field; -s a string field; -c a comment field; and -p just prints the header to stderr. If files are given on the command line the changes are made in place; if no files are given then stdin is read and the converted file is written to stdout without effecting the file on stdin.

2.5.5.3 *postprocess*

This is a *csh* shell script that is executed by *telsched* after each image has been acquired. Its purpose is to copy the image just created by *hpcd* in *archive/images* to *user/images*, possibly applying compression to either file. It will always invoke *fcompress* to compress the former file using a scale factor defined by the COMPRESS parameter in *telsched.cfg.* It will also compress the latter by the same scale factor if the COMPRESS keyword appeared with a value of "Y" in the originating *.sch file.

When this work is complete, this script sends the name of the new user image to the streams pipe */tmp/CameraFilename.* This allows the *camera* X client to automatically display each new image as it is produced.

This script expects the following four command line arguments:

archive image file name

user image directory

scale factor by which to compress the user image, or 0 for no compression
scale factor by which to compress the archive image, or 0 for no compression

2.6 Other Programs

Follows is a description of other useful tools created in the course of this project that are not used directly by the *ccdoper* operating environment. Due to their general nature, the executables reside in */usr/bin/local* unless otherwise specified. The source resides in *tools* unless otherwise specified.

2.6.1 *calimage*

This program is little more than a convenience wrapper around the *correctFITS()* function in *libastro.* It was historically used to post-process the raw images generated by *hpcd* but now that program does its own image corrections and this program is not used by actual *ccdoper* operation. Its usage is as follows:

```
calimage [-c dir] [-b bias] [-t thermal] [-f flat]
```

The program reads an uncorrected FITS file on *stdin*, applies appropriate bias, thermal and flat corrections, and writes the resulting calibrated FITS image to its *stdout.* The arguments allow one to specify an alternate directory containing calibration files, and specific alternate calibration files. The default calibration directory built into the program is */home/ccdoper/archive/calib.*

2.6.2 *chksch*

This program was originally intended to be run under DOS so that students could get a quick check of their *.sch file before submitting them. The *chksch* directory contains slightly-modified copies of the same code used by *telsched* when it reads and parses *.sch files, as well as sufficient other code so it will build using Microsoft C.

The usage is as follows:

```
chksch [-c catdir] file1.sch ...
```

The sources called out in *.sch files refer to objects in catalog files and *chksch* knows to search these files for correct entries. The -c option allows one to specify an alternate directory containing catalogs. If no -c option is given the program searches the directory specified by the catdir[] array in the chksch.c file. This is set to "p:\" as of this writing. Now that this program is being executed on a UNIX machine from DOS via the *rsh* command it is less important to maintain the full stand-alone nature of this directory.

2.6.3 *engshow* and *telshow*

These programs format and display the information in the *telstatshm* shared memory segment to their *stdout* for use on a plain ASCII terminal. The former shows information more germane to engineering status, such as the mount corrections. The latter shows information more germane to finding out what the system is doing from an automatic scheduling point of view.

2.6.4 *fits2gif*

This program accepts the names of one or more .fits files in FITS format as arguments and creates corresponding files in GIF89a format with the same name but with the extension .gif. This program was obtained from Adam Berstein, *adam@bloodhound.jpl.nasa.gov.* Note that until this program is modified to meet our local naming convention of using the extension .fts for FITS files, the .fits extension required by this program will not be convenient.

2.6.5 *fitscrop*

This program will crop an arbitrary rectangular region from a FITS image. It's usage is as follows:

```
fitscrop src dst x y w h
```

If the src and dst files are the same, then the cropping occurs *in place.*

2.6.6 *photom*

This program performs sparse-field photometry from CCD images. It is driven by a configuration file, described in more detail below. The config file lists several .fts image files and x,y locations of stars on each image. The first image is a reference image and its last star is a reference star. Subsequent images may respecify the location of no more than the number of stars in the first image. Subsequent images that specify fewer stars will inherit the location of the unspecified stars from the first image, allowing for image alignment of said image with respect to the first image.

Photom produces one line of output per image. The first column is the geocentric JD of the image; the second is the value to be added to JD to get heliocentric time; the third is the air mass. Subsequent columns are magnitude differences and error estimates for each star pair. The brightness of each star in a given image, except the last, is reported relative to the last star for that image. The brightness reported for the last star of each image is with respect to the last star on the first image. Estimates of the error of each brightness calculation are also given.

Follows is a description of the command line options photom recognizes:

-c

Find a bounding box that contains just the given star coordinates and replace each image file with a version cropped to this size (plus a border of 20 pixels). The box is adjusted for the aligned location of the stars in each image so the location of the box will differ for each image but they will all have the same size box. Photom does this IN PLACE forever destroying the original image files. To record the cropping action that photom has performed it adds two INTEGER fields to the FITS header of each such cropped image: CROPX and CROPY. The effective bounding box size is just the size of the image, of course.

-p

If the images to be processed contain both the CROPX and CROPY fields then they are subtracted from any and all values found in the configuration file for the corresponding image. Star coordinates that are left blank in the configuration file are computed dynamically based on image alignment and so are not modified by the CROP fields. This option is intended to allow running the same config file again on the cropped images (i.e., the ones produced with the -c option, above).

This command has no effect if the images do not contain the CROPX and CROPY header fields.

-v

Generate extra output.

Follows is a description of the photom configuration file format:

all lines that begin with # are ignored.

first line:
rb: max radius to search for brightest pixel
rmax: max radius from brightest pixel to star edge
chg: star ends when pixel sums in expanding disks don't increase by at least this proportion

successive lines:
file x1,y1 x2,y2 x3,y3 ...

where:
<file> can be either a simple file name or a range.
ranges must be of the form Xaa-bb.fts, where:
X is anything;
aa-bb is a starting and ending hex range, such as 10-a0.

<xN,yN> are star coordinates. x and y must be separated with comma and x,y pairs must be separated by blanks or tabs. Up to 8 pairs are supported. No line may contain more coordinates than the first line. Any lines with fewer coordinates than the first will reuse the unspecified coordinates from the previous line.

Sample:

```
# sample photom config file
5 10 .1
/home2/tmp/fx5/fc520704.fts 333,255 418,170 122,253
/home2/tmp/fx5/fc520712.fts 334,256
/home2/tmp/fx5/fc520720.fts
/home2/tmp/fx5/fc520728.fts
```

2.6.7 *saoimage*

This program was picked up from the internet and built for general display of images, particularly those in FITS format. The source is in *~ecdowney/src/saoimage.*

2.6.8 *tstalign*

The *tstalign* program takes two *FITS* file name arguments and determines how the first one can be shifted in X and Y to best match the second. It is used to experiment with the algorithm used by the *align2fits()* function in *libastro.* To try a different algorithm, modify the file *align2fits.c* in the *libastro* directory, remake the *libastro.a* library, then relink the *tstalign* program.

2.6.9 *xv*

This popular X image display client has been modified to accept files in FITS format. The code and the patch for FITS can be found in *~ecdowney/src/xv.*

3. Future Work

The system described above is fully operational but many additional features can be added. Follows is a general list of ideas currently being considered, in no particular order.

- Requests could be accepted from the Internet at large, and an automated mail response system could be set up that emails retrieval instructions back to the originator with a time limit, after which the data is discarded them from the local disk resources.
- Coordinates and exposure conditions of all images could be cataloged and an index made available for general queries. Some users might find that a suitable image set exists to meet the needs of their project and thus avoid taking the same data again.
- Additional cameras should be brought on-line. This allows the use of newer cameras than the HPC. Furthermore, this software is available to anyone interested and yet the HPC CCD camera is perhaps not the most popular camera.
- The automatic setting of focus based on filter selection mentioned elsewhere should be implemented.
- A mode could be devised that automatically focuses. This would basically involve taking an image repeatedly and fitting a curve to the point-spread function of a star image to find its minimum.
- Images can currently only be calibrated if a set of bias, thermal and flat reference files are available that match the size of the image. Work could be done to resize suitable reference files in an exact fit is not available. Similarly, images can not currently be calibrated if they are subsets of the entire CCD frame.
- The *joystick* corrections should update the ideal pointing direction, and not just be used as an additional pointing error. This will insure that the pointing direction saved in the logs, image headers and displayed in the engineering display takes into account the joystick commands.
- Work on an image transmission scheme that capitalizes on the H-transform's unique ability to build up image quality with time. A reception client could be built that listens for such data on, say, a socket and displays it in real-time, allowing the user to get a very fast quick look at the image and to stop it when he/she thinks it is "good enough".
- *Camera* could be enhanced to perform astrometry by tapping into available stellar database such as the GSC.
- The same databases could serve as a feedback loop for automatic pointing corrections.

- The *telsched* X client should be split into two processes. One process, which would remain an X client, would serve as a schedule builder. It would read *.sch files, perform sorts, and build a bursted list of individual observations. The second process, no longer an X client, would read in these bursted observations and perform the methodical function of controlling the available hardware and software facilities to accomplish each one. This would allow the X client to come and go rather than having to remain on-line constantly and relieve it if its real-time control duties. It would also allow for the bursted observation lists to be generated at different times and perhaps by different programs. A separate X client could be built as a front-end to the new controller process if direct control of the telescope was needed for any reason. This X client would not have any knowledge of *.sch files.