Components and Operation of the ROTSE-III Telescope System

Eli Rykoff
University of Michigan

Don Smith
University of Michigan

May 20, 2003

Contents

List of Figures

List of Tables

Chapter 1

ROTSE-III Data Acquisition System

1.1 Introduction

The ROTSE-III data acquisition system ("daq" system) runs on Linux RedHat 6.2. It is comprised of a set of daemons that communicate via shared memory. The central "rotse daemon" (`rotsed`) handles the communication between the various daemons. The peripheral daemons each interface with a different piece of the hardware.

The control system has two modes, automatic and manual. The default mode is auto mode, where all the commands are issued by the `astrod` scheduler daemon (see Section 1.2.8). In manual mode, commands come from the Rotse User Shell (`rush`) via the `userd` daemon (see Section 1.2.9). `rush` is a light, telnet compatible shell based on `rc` that provides flexible access even from limited internet connections. In addition, the `rmonitor` program is a nice way to monitor status in realtime, described in Section 2.6.

1.2 The Daemons

1.2.1 The ROTSE Daemon: `rotsed`

The ROTSE Daemon (`rotsed`) is the central nervous system of the ROTSE daq system, as shown in Figure 1.1. `rotsed` controls system startup and shutdown and handles interdaemon communication. It also handles emergency responses to bad weather and outputs system status for the web based status monitor (See Section 8.4.2).

The ROTSE IPCS shared memory structure is divided into several structures, one for each daemon. Each daemon has a status structure used to report status to `rotsed`, and a command structure, to receive commands from `rotsed`. On each loop, `rotsed` reads the status of each daemon to ensure that they are all running smoothly. If a daemon has not updated its timestamp recently, `rotsed` shuts down the system. If a status structure from one daemon contains information for another daemon, `rotsed` copies the relevant information to the command structure of the destination daemon.

When the weather turns bad, `rotsed` is responsible for issuing an immediate command to close the clamshell. It must be noted that the weather monitor daemon, `weathd`, must be turned on in order for this to work.

While shared memory commands are used to communicate between two daemons at a time, Linux system signals are used to communicate with all the daemons at once. There are several signals used by `rotsed` and the daq system:

- `SIG_TERM`: The termination signal (15)[1] is used to tell a ROTSE daemon to shutdown cleanly. Usual system shutdown is accomplished by issuing a SIG_TERM to `rotsed`, which passes on the signal to the other daemons.

[1] This is the Linux standard signal number associated with a `TERM` signal.

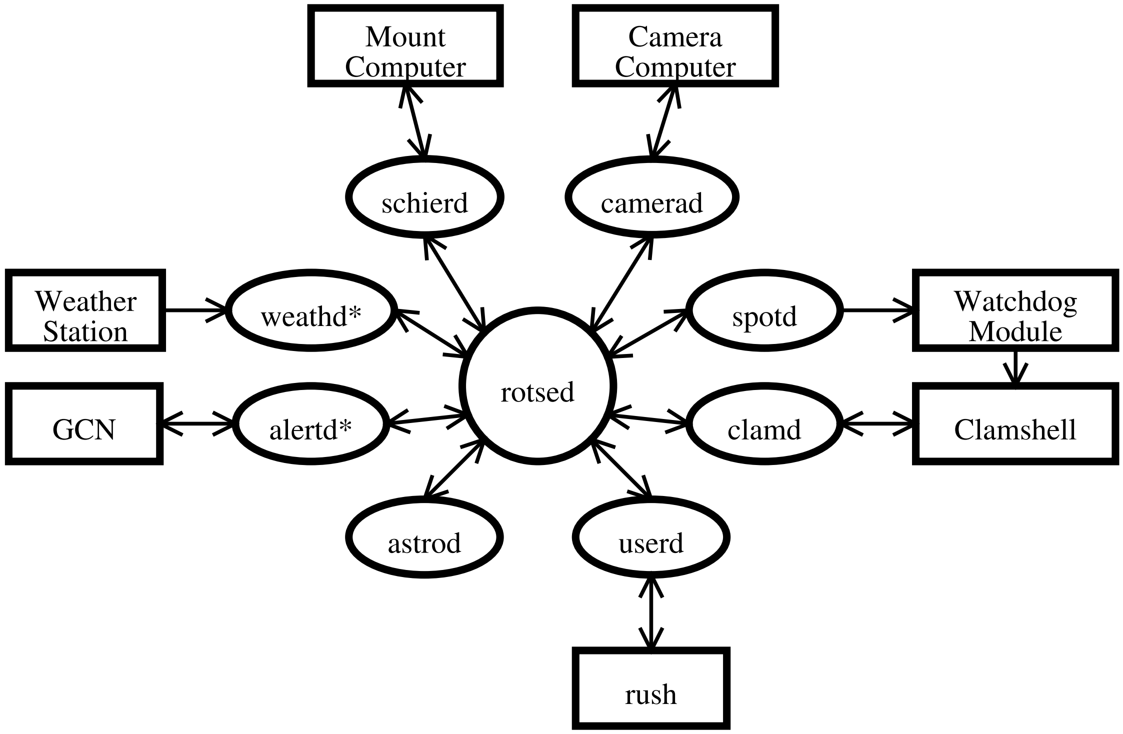

Figure 1.1: The ROTSE-III Data Aquisition system is made up of a number of interconnected daemons which communicate through the central `rotsed`. Each daemon interfaces with a different logically distinct aspect of the telescope system. The daemons with asterisks are also able to send Linux signals to interrupt the camera, mount, and scheduler.

- `SIG_ROTSE`: The ROTSE signal (equivalent to interrupt (2)) is used to tell all the ROTSE daemons to stop what they are doing and wait for further commands. The camera will abort an exposure, the mount will stop slewing, and the scheduler will interrupt its current programming for this important announcement. This signal is used when a burst alert is received by `alertd`, or when the weather turns bad in `weathd`.
- `SIG_HUP`: The hangup signal (1) is used to tell the `astrod` scheduler to re-read its configuration file and reset the schedule, as described in Section 4.1. The other daemons ignore this signal.
- `SIG_KILL`: When all else fails, a KILL signal (9) can be used to terminate a daemon with extreme prejudice. A daemon killed in this way might not exit cleanly.

When the weather turns bad, `weathd` signals a weather alarm in the shared memory status, and sends out a `SIG_ROTSE` to interrupt the other daemons. When `rotsed` receives a `SIG_ROTSE` in conjunction with the weather alarm, it directs `clamd` to close the clamshell.

1.2.2 The Clamshell Daemon: `clamd`

The Clamshell Daemon (`clamd`) interfaces with the i/o box controller clamshell lines. It can issue commands to open or close the clamshell, and can query the clamshell limit switches to determine if the clamshell is open, closed or in transition. If a limit switch opens or closes unexpectedly, clamd will try to issue a close command and will shut down the system.

One should note that operating the clamshell with the manual external switch is incompatible with the normal operation of the daq system. If the system (specifically `spotd`, described in Section 1.2.3) is not running, then the watchdog module will not allow the clamshell to be opened. If `spotd` is running, any unexpected change in the clamshell status caused by opening limit switches will log a fatal error, and the system will shut down. The external clamshell switch only is useful in early set-up stages of the telescope.

1.2.3 The Spot Daemon: `spotd`

The Spot Daemon (`spotd`) pings the watchdog module in the i/o box every second. If the watchdog module fails to receive a ping after five seconds, the clamshell is automatically closed via a hardware switch. Hence, in the unfortunate event of an unexpected shutdown of the daq system (eg, due to software failure or power failure), the clamshell will close itself.

1.2.4 The Camera Daemon: `camerad` and `camserverd`

The Camera Daemon (`camerad`) handles communication with the CCD camera computer. Connection is provided by a TCP/IP socket connection over a private LAN network configured on the `eth1` secondary ethernet card. The `camerad` acts as the socket client, and the Camera Server Daemon (`camserverd`) acts as the server on the camera computer itself. If the socket connection fails and connection cannot quickly be reestablished, `camerad` assumes the camera computer has crashed, and the daq system will be shut down.

The camera will accept the following commands:

- `PICT_EXPOSE`: Take an image with the shutter open.
- `PICT_DARK`: Take an image with the shutter closed.
- `PICT_STATUS`: Return the status (expose, dark, readout, temperature, etc.)
- `PICT_ABORT`: Abort the current exposure. [Will this abort during readout?]

`camserverd` is organized on a ping/pong buffer format. The multi-threaded code can write one image to disk from one buffer ("ping"), while it reads out another image from the camera to the other buffer ("pong"). The image is saved in FITS format, with a custom header describing every relevant statistic from the daq system, including date, time, exposure and weather information, as described in Section C.1. Additionally, the mean and rms deviation of the bias region and the central subframe are calculated. If the mean value of the central subframe is too large, eg, when all the pixels are saturated, then only the image header is written to disk. The bias values are a good indication of the readout noise in the camera. `camserverd` also creates a symbolic link to the image which is read in by the automated analysis pipeline program, as described in Chapter 8.

1.2.5 The Schier Mount Daemon: `schierd`

The Schier Mount Daemon (`schierd`) handles communication with the mount controller computer. Connection is provided by a serial line at 9600 baud, with CRC checksums to ensure every command is understood. The mount controller itself takes simple commands to find its home position, change axis speed, and change target position. All commands are in absolute encoder steps, and it is the responsibility of `schierd` to make the necessary transformations to point at the sky and track at the correct speed and direction. `schierd` also uses the outside temperature and the target elevation to apply a simple focus model described in Chapter 6 and move the focus motor. The connection to the focus motor controller is also via a serial line, at 32000 baud.

`schierd` is controlled by a stack of commands. For example, upon receipt of a "move" command, three commands are put on the stack in the following order: track, move, adjust focus. In this way, the focus is adjusted and the telescope slews (this actually happens in parallel for a faster response time), and then the mount begins to track on the sky. Only when the mount starts tracking does `schierd` signal that the move is complete. `schierd` is also able to park the mount, find the home position, and move to "standby" position towards the zenith, which optimizes the response time to a random location on the sky.

1.2.6 The Weather Daemon: `weathd`

The Weather Daemon (`weathd`) handles communication with the Davis Weather Station and the Vaisala Precipitation Detector. This multithreaded daemon has both client and server modes, so that one weather station and/or precipitation detector can be used for a suite of collateral telescopes.

Aside from returning weather statistics such as temperature which are saved in the image header, the most important task for `weathd` is to determine when the weather gets "bad." Any detection of precipitation is deemed bad. The operator can configure other definitions of bad weather conditions, triggering on windspeed, dewpoint, humidity or temperature. Upon detection of bad weather, `weathd` immediate sends out a `SIG_ROTSE` signal to the entire daq system, and issues a weather alarm for `rotsed` in its status structure. This causes the current exposure to abort, the mount to stop slewing or tracking, and the clamshell to close. `weathd` will wait for at least one hour to check if the weather has cleared.

1.2.7 The Alert Daemon: `alertd`

The Alert Daemon (`alertd`) handles the connection with the Gamma-ray Burst Coordinate Network (GCN) to receive burst triggers. `alertd` acts as a TCP/IP socket server on a port number registered with Scott Bartlemy at Goddard. The port is configurable and site-specific. On start-up, a permanent socket connection with the GCN is established. Each minute the GCN sends a special `IM_ALIVE` packet. If several minutes pass without receipt of an `IM_ALIVE` packet, the socket connection is shut down and tries to reestablish itself. In a separate thread, `alertd` runs a simulated GCN server. This server listens for a temporary socket connection on a configurable port, and can be used to test the response of the daq system to GRB triggers. A program called `simalert` can be used to mimic GCN packets and establish a socket connection with the daq computer on this second port.

Upon receipt of a GCN packet (via the real GCN or the simulated GCN), `alertd` parses the packet to determine the GRB serial number, time, and position, as well as monitoring flags. If the new packet has the same type and serial number as the previous packet, it is assumed to be a glitch and is logged and ignored. If the packet passes this cut, the alert type is then checked against a configurable hash table to determine the response priority. Simulated alerts are automatically given lower priority than real GCN alerts. If no alert is currently running, or if the new alert has a higher priority than the current running alert, a `SIG_ROTSE` is issued and the GRB position is sent to the scheduler for immediate response. It is up to the scheduler to determine whether the burst is currently visible, and if not, to turn off the alert mode in the daq system.

1.2.8 The Astronomical Scheduler Daemon: `astrod`

The Astronomical Scheduler Daemon (`astrod`) schedules observations, system startup and system shutdown. It consists of a modified queue scheduler that decides in real-time which field is the best field to image next. Gamma-ray Burst alerts from `alertd` are automatically put in the front of the queue for immediate processing. In addition, GRB alerts are logged and further observations of the target coordinates ("follow-up" observations) are scheduled to be performed at logarithmically increasing time intervals. See Chapter 4 for information on how to set up the scheduler.

There are four primary types of schedule items that can be configured, listed here in increasing priority:

- "Sky Patrols:" The entire sky is gridded to a pattern, whose spacing depends on the telescope field of view. A configurable region of this grid can be cropped based on right ascension, declination, and galactic latitude. These regularly spaced fields form the "sky patrol," that is the default mode of operation for wide-field surveys. For more details, see Section 4.3.3.
- Targeted Observations: Specific fields can be named as targets in the configuration file. These are considered slightly more important than sky patrol fields.
- Late Burst Follow-up Observations: After a burst alert has been received, the location is logged in a file described in Section 4.5. At late times, the burst location is repeatedly imaged at increasing time intervals.
- Prompt Burst Observations: Upon receipt of a burst alert, if the weather is fine, the sun is down, and the burst is above the horizon, the scheduler will immediately schedule a prompt burst observation sequence.

The parameters for the schedule items are defined in `astrod.conf`, described in Chapter 4. Each schedule item has a corresponding field or set of fields, and a set of configuration options. These include vetoes based on field elevation, lunar illumination, solar elevation, and camera temperature. In addition, if a field is to be imaged more than once, a cadence can be specified.

Each schedule item also has a corresponding imaging sequence defined in the configuration file. For sky patrols, this is typically set to two consecutive long (60 s) exposures. For prompt burst responses, this is currently set to 10 short exposures, 10 medium exposures, and 50 long exposures. New imaging sequences can be defined by the operator, as described in Section 4.4.1.

The observing schedule is built in real-time. When the previous imaging sequence has finished, the scheduler scans through the list of schedule items. First, each field referenced in all the schedule items is checked for quick vetoes. If a field passes the veto cut then the scheduler calculates a score based on current airmass, whether or not it is a specific target, and whether or not the cadence demands a return to the field. The relative weights given to each of these criteria is configurable as described in Section 4.3.5. The scheduler then has the telescope slew to the highest scoring field which is then imaged with the corresponding imaging sequence. The typical calculation time to determine the best field from a sky patrol list is ~ 0.2 seconds. Prompt burst alerts are automatically put at the front of the queue with higher priority, yielding faster processing times. If no schedule item passes the veto cuts then the telescope returns to "standby" mode, pointed at the zenith.

When `astrod` receives a `SIG_ROTSE` from `alertd`, the current schedule is interrupted and the alert information is copied from shared memory. The burst location is then logged for follow-up, and a burst alert schedule item is put in the front of the schedule queue. If the trigger location is above the horizon and the system is running in good weather, the mount is told immediately to slew to the trigger location, and system begins the preconfigured imaging sequence. If any of these conditions are not met, the schedule item remains at the front of the queue until it can be imaged, or its lifetime expires. If the burst field is not viewable, the scheduler will continue with its normal operations.

Now I will describe an example of a typical observing sequence, when a sky patrol and various targets have been configured. Under normal operation, the telescope will point near the maximum elevation of the sky patrol fields, and fields will drift into this area. As specific target fields become visible, they are imaged, and then the telescope returns to the regular patrol. When a field has a cadence specified, the scheduler tries to keep to the cadence to the best of its ability.

The `astrod` scheduler is also responsible for telling the clamshell to open at sunset, and close at sunrise. The configuration file is re-read at the end of each night's observing after the clamshell is closed.

When the system is put in manual mode, `astrod` quietly loops until it regains scheduling control in auto mode.

1.2.9 The User Daemon (`userd`) and the ROTSE User Shell (`rush`)

The User Daemon (`userd`) is the daemon that takes over issuing commands when the telescope system is put into "manual" mode. It is a skeleton daemon whose sole purpose is to interface between the ROTSE User Shell (`rush`) and the daq system. `rush` is a modified bourne shell (based on Byron Rakitzis' `rc` shell) with specific ROTSE commands built in to operate the hardware. While in "auto": mode, `rush` is also useful for monitoring the status of the system in real-time. Specifics on how to use `rush` are in Section 2.5.

Chapter 2

How To Run the ROTSE-III System

2.1 Introduction

In this chapter I describe how to start and run a fully configured ROTSE-III Telescope system. Setting up the system should be done by one of the fully certified ROTSE technicians.

Information on the daq system can be found in the Chapter 1. Working with configuration files is found in Chapter 3. Scheduling information is in Chapter 4. Automated pipeline information is in Chapter 8. And so it goes.

2.2 System Files

The system files contain everything needed to run the data acquisition system. This does not include auxillary binaries, idl files, etc. These system files are put in the following directories:

```
/rotse/run/bin/          Directory with Binary Files
/rotse/run/etc/          Directory with Configuration Files
/rotse/run/cfg/          Directory with rotsed lockfile
/rotse/run/log/          Directory with daemon logfiles
/var/log/rotse.log       ROTSE logfile
```

Image data is saved in the following directories:

```
/rotse/data/3a1/         Directory with normal image files
/rotse/data/3a2/         Directory with alert image files
/rotse/data/pipeline/    Automated Pipeline Parent Directory
```

For more details on file naming conventions, see Appendix A.

2.3 Starting the System

2.3.1 Monitoring Logfiles with `tail -f`

The UNIX command `tail` is indispensible for monitoring logfiles in realtime. `tail` on its own prints the last 20 lines of a text file. Combined with the `-f` “follow” option, it will print each new line from the logfile as it is written. For example, to monitor the system status logfile:

```
$ tail -f /var/log/rotse.log
...
Jan 14 15:43:24 rotsei rotsed:  Beginning ROTSE shutdown at Mon Jan 14 15:43:24 2002
```

2.3.2 The DAQ System: `rotsed`

Starting up the system is easy. First log in as "observer." Make sure all the configuration files are set correctly, and that all the daemons you want to run are turned on in the `rotsed.conf` configuration file (see Section 3.2). You should open two terminals, one to watch system status, and the second to start and operate the telescope. A third terminal to monitor `astrod.log` is sometimes useful.

To monitor the system error logs, in the first terminal use `tail -f` as described in Section 2.3.1.

To start the system, in the second terminal type:

```
$ rotsed
$
```

In the logging terminal you should see the various daemons start up. If there is an error in one of the configuration files, the error is logged and the system will not start up properly.

Upon startup, the system immediately goes into automatic mode, where the `astrod` scheduler has control of the system startup, imaging, and shutdown. To issue manual commands to the telescope see Section 2.5 on the ROTSE User Shell.

2.3.3 Online Analysis: The PacMans

Before you start the pacman programs, check if they are already running:

```
$ ps auxww | grep sexpac
...
$ ps auxww | grep idl
```

If either of these calls returns a process ID, do NOT run the following commands. To start the Pacman programs, open a terminal and type:

```
$ cd /rotse/data/pipeline/
$ startsexpac
$ startidlpac
$ exit
```

The output of `sexpacman` is logged to a file `sexpac.log`. The output of `idlpacman` is logged to a file called `idlpac.log`. The outputs can be monitored with the `tail-f` command as described in Section 2.3.1.

2.4 Shutting Down the ROTSE System

To shut down the ROTSE system, you send a `kill -TERM` signal to `rotsed`. There is a PERL script called `killrotse` that performs this function.

For recalcitrant daemons that will not shut down, use `killrotse -dammit` to send a `kill -KILL` signal to any remaining ROTSE processes.

2.5 The Rotse User Shell: `rush`

2.5.1 Starting `rush`

Starting `rush` is easy, as long as your .rcrc file is correctly configured. This better be the case...[1]

```
$ rush
```

[1] This is one of many responsibilities of your fully certified ROTSE technician.

```
rush> rinit
```

The `rinit` command checks if `rotsed` is running, and if so, connects to the shared memory segment. You must run `rinit` to get control of the telescope. If `rotsed` is not running you will get the following error:

```
rush> rinit
Using observatory file:  /rotse/run/etc/observatory.conf
error:  could not open /rotse/run/cfg/rotsed.cfg,
No such file or directory
      No rotse control functions will be available
rush> exit
```

On the other hand, if you get a message like the following, you're good to go.

```
Need the proper response when we have a working system
```

2.5.2 Shell Information

The ROTSE User Shell (`rush`) is a fully functional shell environment, and can run executables, shell scripts, perform file manipulation, etc. In addition, there are many ROTSE-specific built-in functions that are described in this chapter. Built-in functions can be used from any directory, and from any shell script.

For each command that operates hardware, after the command is issued control is returned to the shell. Execution of the command may take some time. Monitoring the status with `rstat` reveals when the command has finished. If you issue a command with the `-w` wait option, the shell will block until the command has finished.

2.5.3 `rhelp`: The Help Command

The `rhelp` command lists all the builtin ROTSE commands available. These are summarized in Table 2.1.

2.5.4 `rstat`: The Status Command

The `rstat` command returns status information about the various running daemons. A sample output is listed here, and is fairly self-explanatory.

```
rush> rstat
   rush:
       Loc Time: 2002 04 23   15 03 45.69
       UTC:      2002 04 23   21 03 45.69
       LMST: +04 05 39.784   MJD: 52387.877612   EPOCH 2002.3
       SUN: up  ra +02 05 12.36  dec +12 41 35.4
            ha +02 00 27.42  el 54.35  az 237.12
       MOON: in twilight  ra +11 27 06.46  dec +08 50 29.5
            ha -07 21 26.68 el -10.87 az 70.61 frac 0.855
   userd:
       ireq = 1,  oreq = 0
   rotsed:
       state = ALERT
       mode = MODE_MANUAL
   clamd:
       timestamp = Tue Apr 23 15:03:44 2002
       mode = CLAM_CLOSE
   mountd:
       timestamp = Tue Apr 23 15:03:41 2002
       mode = 0
```

rush Command	Description
rhelp	Summary of all commands
rstat	sun, moon, target & system status
rmode [none|manual|auto]	set system control mode
rdome [-w] [open|close]	open—close the dome
rsync [-w]	move mount to sync position
rstand [-w]	move mount to standby position
rpark [-w]	move mount to park position
rmove [-w] [-c] [-n] [-j epoch]	
[-s slew_spd] -[r|h] [ra|ha] -d dec	move mount to specified location
rshift [-w] [-r delta_ra] [-d delta_dec] [-t]	shift the mount axes
fsync [-w]	move focus to sync position
fmove [-w] -[a|r] [position|offset]	move focus (in mm)
rdark [-w] -t time	take a dark image
rexpose [-w] -t time -n name	take an exposure
rabort [-w]	abort an exposure
rsettemp temp	set CCD temperature
ralert [stop]	stop a current alert
rstarlog -r ra -d dec [-j epoch]	log the current pointing
rstarlist	print the currently logged stars
rstardel -r record	delete a star record
rstarwrite -f filename	output the logged stars to disk

Table 2.1: ROTSE commands built in to rush as described by the rhelp function.

```
    mode = MOUNT_IDLE
    trkspd = 0.00
    slwspd = 0
    ra = 0.00
    dec = 0.00
    focus = 3.32 mm
    encoder ra  = 110024
    encoder dec = 1192971
camerad:
    timestamp = Tue Apr 23 15:03:41 2002
    mode = PICT_IDLE
    corner (x, y) = 0, 0
    size (x, y) = 2200, 2200
    binsize (x, y) = 1, 1
    frame number = 1
    server status = 1
    cooler status = TSTAT_OVER
    temperature (deg. F) = 19.37
    temperature (deg. C) = -7.018990
    start time = Tue Apr 23 15:02:56 2002
    exposure time (sec.) =    20.00
    filename = 020423_drk0200_3a001.fit
    readout time (sec.) =  6.29
    saturation count = 0
    bias stddev = 41.45
spotd:
    timestamp = Tue Apr 23 15:03:44 2002
```

```
    watchdog card temperature =   0
weathd:
    timestamp = Tue Apr 23 15:03:25 2002
    temp in =   79.5, temp out =   77.4
    wind spd =    7.0, wind dir = 289
    bar =  23.19, hum in =    2.0, hum out =    5.0
    rain =  0.00, dewpt =    0.6
    sky =  0.00, v. precip =  0.00
    status = GOOD
alertd:
    timestamp = Tue Apr 23 15:03:44 2002
    mode = 99
    trigger # = 2, serial # = 1
    packet time = Fri May 24 15:03:41 1968
    trigger ID = 0, time = Fri May 24 15:03:31 1968
    RA =  24.56 +/-   0.30
    dec =  25.20 +/-   0.30
    intensity =      0.000, origin = SIMGCN
```

2.5.5 rmode: The Mode Command

On startup, the ROTSE system immediately goes into automatic (AUTO) mode. To issue manual commands, `rmode` must be used.

- `rmode [manual|auto]                -- switch between auto and manual`

  In manual mode, `userd` and `rush` have control of the system. In auto mode, `astrod` has control of the system. Remember to leave the system in auto mode for normal operation! In manual mode, `astrod` simply loops until control of the system has been returned to it.

2.5.6 Clamshell Commands

- `rdome [-w] [open|close]            -- open or close the clamshell`

  The `rdome` command is used to open or close the clamshell. Any close command issued while the clamshell is open or in the process of opening will override the current command and close the clamshell.

2.5.7 Mount Commands

- `rsync [-w]                         -- move mount to sync position`

  The `rsync` command moves the mount to the home position and resets the encoder position. This command must be run before the telescope can point at the sky.

- `rstand [-w]                        -- move mount to standby position`

  The `rstand` command moves the mount to standby position as defined in the `schierd.conf` file. This is usually set to the zenith position.

- `rpark [-w]                         -- move mount to park position`

  The `rpark` command moves the mount to the park/stow position as defined in the `schierd.conf` file. This is usually set just below the horizon to the north or south, depending on the location of the telescope.

- `rmove [-w] [-c] [-n] -r ra -d dec [-j epoch] [-s slew_spd (1-100)]`

- `rmove [ ] -h ha -d dec            -- move mount to specified location`

  The `rmove` command moves the mount to the specified target, and starts the mount tracking at sidereal rate. The user can specify either right ascension and declination or hour angle and declination. The epoch defaults to J2000.0. The RA/HA units are decimal hours or [hour minutes seconds], delimited by spaces. The Dec units are decimal degrees or °/′/″, also delimited by spaces.

  The check option (`-c`) has `rush` check the validity of the coordinates before sending them to the mount. If the operator is not careful, they might find the mount pointed below the horizon! However, there are software and hardware limits to ensure the mount will not try to point outside the encoder range.

  The slew speed option (`-s`) describes the percentage of the maximum speed at which to slew. Default is 25%.

  Under normal operations, the mount will calculate the best focus from the temperature and elevation. During testing and calibration, the operator might wish to override the auto focus with the `-n` "no autofocus" option. The focus can be set manually using the focus move commands in Section 2.5.8.

- `rshift [-w] [-r delta_ra] [-d delta_dec] [-t]  --shift the mount axes`

  The `rshift` command moves the mount a specified amount on the RA and Dec axes. This command is for use before a pointing model has been constructed. The values `delta_ra` and `delta_dec` refer directly to the RA and Dec axes of the mount, and not to the celestial sphere. The units of `delta_ra` and `delta_dec` are both degrees.

  The tracking (`-t`) option starts tracking the mount at sidereal rate on the RA axis only. When the mount is reasonably close to polar alignment this works fine for short exposures, but streaking appears at long exposures. For long exposures a pointing model is necessary.

2.5.8 Focus Commands

- `fsync [-w]                        -- move focus to sync position`

  The `fsync` command moves the focus motor to the home position, and resets the focus encoder to 0. This command must be run before an absolute focus position can be used.

- `fmove [-w] -a position            -- move focus to absolute position`

- `fmove [-w] -r offset              -- move focus offset mm`

  The `fmove` command moves the focus to an absolute position or a specified offset. The focus is measured in millimeters. Under normal use, the absolute positions are easier to keep track of, although the current focus postion can always be read from an `rstat` command.

2.5.9 Camera Commands

- `rdark [-w] -t time               -- take a dark image`

  The `rdark` command is used to take a dark frame. The length of the dark (in seconds) is specified with the `-t` option. The dark will be placed in the path specified in `camerad.conf`, usually `/rotse/data/3a1/`. The name follows the standard naming convention described in Chapter 3, with "drk" as the three letter acronym, and the field replaced by the exposure time multiplied by 10. For example, the second 60 second dark image taken on March 15th, 2002 will be given the name `020312_drk0600_3a002.fit`.

- `rexpose [-w] -t time -n name     -- take an exposure`

  The `rexpose` command is used to take an exposure. The length of the exposure (in seconds) is specified with the `-t` option. The name can be up to 8 characters (this needs updating!) and replaces the three letter acronym and number in the standard naming convention. The exposure will be placed in the path specified in `camerad.conf`, usually `/rotse/data/3a1/`. The exposure counter at the end of the name is automatically updated by the daq system, so up to 999 images of the same name can be taken on the same day.

- `rabort [-w]                    -- abort an exposure`

  The `rabort` command is used to abort an exposure (image or dark) in progress. If the camera is reading out the abort will take place after the readout is finished. If no exposure is taking place this command has no effect.

- `rsettemp temp                 -- set CCD temperature`

  The `rsettemp` command is used to set the CCD temperature. This overrides the default set in `camerad.conf` (see Section 3.5).

2.5.10 Alert Commands

- `ralert [stop]                  -- stop a current alert`

  The `ralert stop` command is used to turn off alert mode. This is the same as the `aoff` command in the `astrod` scheduler. (See Section 4.4.1) This command is useful during debugging and testing sessions if the system gets stuck in alert mode for some reason.

2.5.11 Pointing Model Logging Commands

This set of commands is used to log the current nominal pointing (in encoder counts), current time, and known star position. The output format is used by both the idl two-star pointing routines and the Tpoint package. For more information on how to construct a pointing model, see Chapter 5.

- `rstarlog -r ra (dec.  hrs.)  -d dec (dec.  deg.)  [-j epoch]  -- log the current pointing`

  The `rstarlog` command logs the current telescope pointing, local mean sidereal time, and star position. This should be used when the telescope is approximately tracking a known bright star.

- `rstarlist                      -- print the currently logged stars`

  The `rstarlist` command prints to screen the star records that have already been logged.

- `rstardel -r record             -- delete a star record`

  The `rstardel` command deletes a record from the list of logged stars. This should be used to delete a mistaken entry.

- `rstarwrite -f filename         -- output the logged stars to disk`

  The `rstarwrite` command outputs a Tpoint formatted file to disk with the currently logged stars from `rstarlog`. The standard extention is `.dat` for a Tpoint file, although this is flexible.

2.6 True Realtime Status Monitoring with `rmonitor`

The `rmonitor` program is a true realtime status monitoring program that works with a low-bandwidth shell. It was conceived as an easier status monitor than using `rstat` (see Section 2.5.4) repeatedly.

`rmonitor` hooks into the daq system through `userd` with the same method as `rush`. Under normal usage, it is used as a companion to `rush` in a separate terminal window. Usage is easy; an alias will have been set up to properly link to the system files:

```
$ rmonitor
```

`rmonitor` requires the terminal window to be at least 80 columns by 24 rows (the default terminal size), or it will not run. It will take over the terminal screen and display something like the screen shot from Figure 2.1.

The key values are put on the screen in a (hopefully) easy to read format. The "Astronomical Stats" are updated every 3 seconds. (This is configurable with the -t option on the command line). `rmonitor` polls the system every 0.3 seconds (configurable with -p) to determine if any values have changed, and updates the screen accordingly. Daemons that are running are put in bold face, and those that are not are put in regular

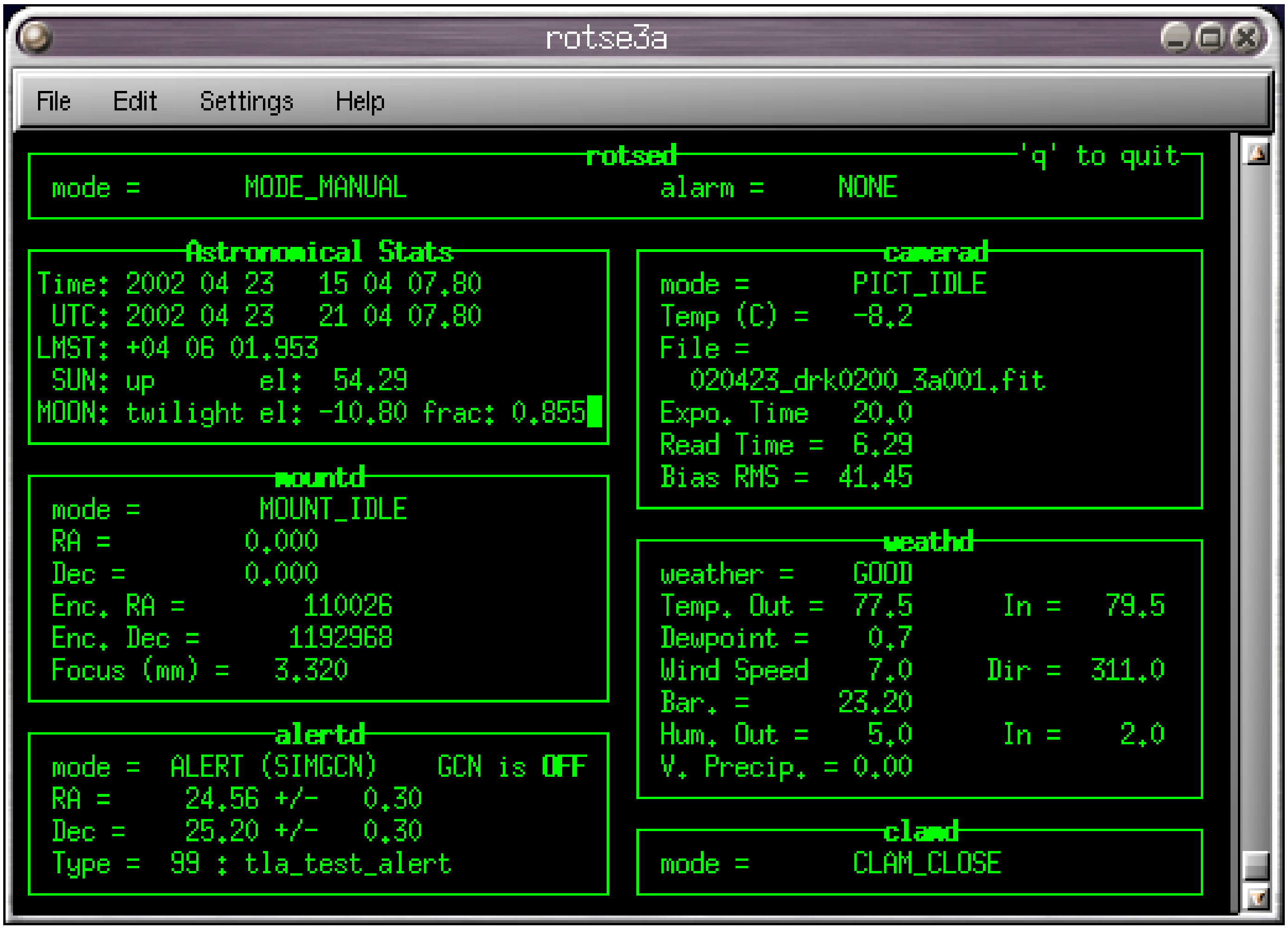

Figure 2.1: Screen shot from the popular `rmonitor` program.

type. In addition, if the screen is large enough vertically, `rmonitor` will tail the `/var/log/rotse.log` logfile in a separate ncurses window. The logfile window contains a buffer of 500 lines, and can be scrolled with the up arrow, down arrow, page up, page down, and end keys.

In addition, `rmonitor` is persistent. That is, it will remain running even if `rotsed` shuts down, and will continue to monitor the time, sun position and moon position. When another `rotsed` process is started, `rmonitor` automatically hooks in and displays the latest status information.

Chapter 3

The ROTSE-III Configuration Files

3.1 General Configuration File Information

A generic configuration file can be found in `skeld.conf`, which is a description of a hypothetical "skeleton" daemon. The configuration files are each read in with a standard reader, and each file begins with the following header and configuration parameters:

```
# configuration file for skeld
#
#
# format is "key value"
# comments start with ‘ #’ and go to end of line
# blank lines are okay
# lines cannot be continued

loglevel       1                               # 0,1,2 = terse,verbose,debug
logfile        /rotse/run/log/skeld.log        # logfile location
poll_time      0.05                            # main loop time quantum
sample_time    3.0                             # sample time in seconds
```

The `loglevel` can be set to 3 levels, which controls the amount of logging to the daemon logfile specified by `logfile`. In addition, each daemon will log important system information to `/var/log/rotse.log`.

On each pass through its main loop, each daemon will check the shared memory for new commands. At the end of the loop, it waits for `poll_time` seconds. The daemon only updates its status every `sample_time` seconds, or sooner if the status changes significantly. If all of the poll times are set too small, then too many daemons try to access the shared memory at the same time, and the system becomes unstable. Daemons for which speed of response is not an issue should have larger `poll_time` values.

In addition, multi-threaded daemons such as `alertd` and `weathd` have the following options:

```
sample_time_th    3.0                     # thread sample time
init_time         5.0                     # how long wait for children
```

The `sample_time_th` is the individual thread sample time, for the children to update their status values for the parent. The `init_time` is how long to wait for the children to start up before assuming something has gone wrong, and shutting down the system. Any such startup problems are logged to `/var/log/rotse.log`.

3.2 rotsed.conf

The Rotse Daemon is the central daemon that starts the other daemons and handles interprocess communication. The configuration has several sections: one section specific to `rotsed` and one section for each subsidiary daemon. The `rotsed` section is as follows:

```
# configuration file for rotsed
#
# format is "key   value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued
#
# stuff just for rotsed
#

loglevel         1                                  # 0,1,2 = terse,verbose,debug
logfile          /rotse/run/log/rotsed.log
cfgfile          /rotse/run/cfg/rotsed.cfg          # Location of ipc key & lockfile
poll_time        0.01                               # main loop time quantum
email_list       "observer@rotse3a.lanl.gov"
jsfile           /rotse/run/log/rotse3a.js          # JavaScript Status File
jsdelay          60.0                               # Interval (s) for js updates
```

The `cfgfile` acts as the lockfile for the daq system. If the `cfgfile` is present on the system, `rotsed` will not start and logs the error to `/var/log/rotse.log`. This file also contains the IPC key for the shared memory, so the other daemons can link up with the shared memory segment.

Every 60 seconds (set by the `jsdelay` keyword) the system writes a JavaScript formatted status file to `jsfile`. A cron job is set up on the `www.rotse.net` computer to copy this status file every minute. The JavaScript file is used to create a status webpage that is available for the public to view near real-time system status, as in Section 8.4.2.

```
# parameters to control weathd
#
weathd run       1                                  # yes/no = 1/0
weathd conf      /rotse/run/etc/weathd.conf         # confile name
weathd path      /rotse/run/bin/weathd              # path to executable
weathd tout      60.0                               # timeout for updates
weathd tinit     120.0                              # timeout for initialization
weathd tlog      300.0                              # logging period

alertd run ...

astrod run ...

camerad run ...

clamd run ...

mountd run ...

spotd run ...

skeld run        0                                  # Should not run skeld
```

```
userd run ...
```

Each daemon can be set whether to run or not. For debugging and testing it is useful to run specific daemons, but in normal operation it is vital to have all the daemons running. For each daemon the operator must specify the path of the executable as well as the path to the configuration file. If a daemon exceeds its initialization timeout on startup or its update timeout during normal operation, `rotsed` will automatically shut down the system. These timeouts should be large enough that they are never exceeded during normal operation, and they can depend on the cpu speed of the controller computer, as well as the additional load from online processing.

3.3 alertd.conf

The Alert Daemon is a multi-threaded daemon, and requires the additional thread timeouts as listed above in Section 3.1. The configuration file is divided into sections, for the real GCN monitor, the simulated GCN monitor, and the packet type definitions.

```
# configuration file for alertd
#
#
# format is "key value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued

loglevel        2                               # 0,1,2 = terse,verbose,debug
logfile         /rotse/run/log/alertd.log
poll_time       0.05                            # main loop time quantum
sample_time     3.0                             # sample time in seconds
sample_time_th  3.0                             # thread sample time
err_tout        5.0                             # timeout for ALRM_ERR condition
init_time       5.0                             # how long wait for children
email_list      observer@rotse3a.lanl.gov       # emails to receive alerts
obsfile         /rotse/run/etc/observatory.conf
```

Along with the options previously described, the operator must specify an e-mail list to receive alert reports, and the observatory file, which contains the longitude, latitude, and altitude of the observatory.

The next sections configure the mode of operation of the GCN monitor and the simulated GCN monitor. It is strongly recommended that both of these options be turned on at all times for normal operation!

```
# ================= GRB Coordinates Network monitor ===========================
#
gcnmon run      1                               # = 0/1 to run GCN monitor
gcnmon tout     0.0
gcnmon tpoll    0.1                             # GCN sample time
gcnmon twait    3600.0                          # ????
gcnmon devfile  rotse3a.lanl.gov                # host computer
gcn_portnum     5147                            # socket port for alert stat.
pkt_delay       30.0                            # not used
bind_delay      60.0                            # not used

# ================ Simulated GCN Coordinate Socket Server================ #
simgcn run      1                               # = 0/1 to run Simulated GCN server
simgcn tout     0.0
```

```
simgcn tpoll     0.1                                # SimGCN sample time
simgcn twait     3600.0                             # ????-not sure what it does
simgcn devfile   rotse3a.lanl.gov                   # host computer
simgcn_portnum   2545                               # socket port for SimGCN
```

These sections define the timeouts, device files and port numbers for the GCN Monitor and the SimGCN Monitor. For the GCN Monitor the port number must be registered with Scott Bartlemy at NASA; the SimGCN port number is arbitrary, and only needs to be known by the `simalert` program which simulates GCN triggers. The host computer should be the full name of the local computer.

The Alert Server, Alert Client, and Alert Database threads are no longer supported, and should all be turned off. Sometime in the future all references to this code will be deleted.

```
# Available GCN Trigger types of interest (deprecated values are commented out)
#
# header        name           index #     pos_div    priority
#alerttype    gcn_test            2          100         101
alerttype    gcn_imalive         3          1           101
alerttype    gcn_killpacket      4          1           101

alerttype    xrt_alexis          25         100         101
alerttype    xrt_rxte_pca        26         10000       101
alerttype    xrt_rxte_asm        28         10000       101

alerttype    grb_beppo           34         10000       105
alerttype    grb_beppo_nfi       36         10000       105
alerttype    xrt_asm_trans       37         10000       105
alerttype    grb_ipn             39         10000       105

alerttype    grb_hete_alert      40         10000       150
alerttype    grb_hete_update     41         10000       151
alerttype    grb_hete_final      42         10000       151
alerttype    grb_hete_ground     43         10000       151

alerttype    tla_test_alert      99         10000       101
```

These are the GRB alert definitions. The alert index is defined by the GCN at `gcn.gsfc.nasa.gov/sock_pkt_def_doc.html`. The `pos_div` value determines the precision of the RA and Dec positions given in the packet, also defined by the GCN. The larger values indicate more precise coordinates (although they might not be accurate!). The priority values range from 101 to 200. The SimGCN server automatically subtracts 100 from the priority for any test alert. True GCN triggers therefore always override simulated GCN triggers. In the case of a tie, the current trigger continues running, and the new trigger will wait until the current one is finished. This situation has not occurred yet.

Any trigger packet that arrives without coordinates is logged and ignored. Most HETE-2 type 40 triggers come without coordinates, although in the future they might include rough positions.

3.4 astrod.conf

For schedule configuration read Chapter 4. The scheduler also uses `burst.conf` and `landoltfields.conf`.

3.5 camerad.conf

Originally written for ROTSE-I, which had four co-mounted cameras, `camerad.conf` is written to accommodate up to four cameras, designated a,b,c, and d. For ROTSE-III, only one camera needs to be specified as follows:

```
# configuration file for camerad
#
# format is "key   value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued

loglevel        0                               # 0,1,2 = terse,verbose,debug
logfile         /rotse/run/log/camerad.log
poll_time       0.1                             # main loop time quantum
sample_time     8.0                             # sample time in seconds
err_tout        5.0                             # timeout for ALRM_ERR condition

#
cam_temp        -20                             # target temp. (in C) for cameras
csrv_wait       5                               # wait time for camserverd
```

There is one temperature that is used for all the cameras (in this case just 1). The temperature setting can depend on the season and the particular camera. Note that the first ARC Camera (serial #80, planned to go to SSO) does *not* have the correct temperature calibration, so the nominal -20°C does not actually represent the physical temperature of the CCD.

```
cama sock       1                               # 0/1 to run camera
cama port       3900                            # port for camera 'a' node
cama name       cam3a                           # name for camera 'a' node
cama path       /rotse/data/3a1/                # path to camera 'a' sky/drk images
cama alertpath  /rotse/data/3a2/                # path to camera 'a' alert images
cama image_id   3a                              # image identifier
cama ra         0.0                             # offset in RA
cama dec        0.0                             # offset in dec
cama focus      0.0                             # focus position
cama telman     Schier                          # Telescope/lens manufacturer
cama telmodel   1.9deg                          # Telescope/lens model
cama telsn      0                               # Telescope/lens serial number
cama camman     ARC                             # Camera manufacturer
cama cammodel   EEV                             # Camera model
cama camsn      0                               # Camera serial number
cama cardsn     80                              # Camera PC card serial number
```

On startup, `camerad` will send a socket connect request to the port on the camera node. The `inetd` service file is configured to automatically start up `camserverd` on receipt of this connect request. `camerad` will then wait `csrv_wait` seconds for the camera server to start up and respond. If the camera computer does not respond or if it does not respond fast enough, `camerad` will shut down the system.

This configuration file also specifies the paths for writing files. The paths should be local drives on the camera computer to remove any nfs latency. Separate hard drives are used for regular data and burst response ("alert") data so that the disk will not fill up if and when a burst trigger arrives. Each image is named as described in Appendix A.

The `ra` and `dec` offsets were used in ROTSE-I when each camera had an offset from the mount position. The `focus` option is also for ROTSE-I's fixed focus cameras. These should be set to 0.0 for ROTSE-III. The rest of the values are useful values that are stored in the FITS header, described in Appendix C.

3.6 camserverd.conf

`camserverd` controls more hardware-specific details of the running of the camera. The `camserverd.conf` file contains the following:

```
# configuration file for camserverd#
#
# format is "key   value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued

poll_time          0.05                              # main loop time quantum
sample_time        3.0                               # sample time in seconds
driverpath         /dev/astropci0                    # location of driver
confpath           /rotse/run/etc/                   # .lod files location
lockpath           /rotse/run/cfg/                   # camera driver lockfile path

# boundaries of image subframe for statistical analysis

subframe_xmin      974                               # minimum x-coordinate
subframe_xmax      1074                              # maximum x-coordinate
subframe_ymin      974                               # minimum y-coordinate
subframe_ymax      1074                              # maximum y-coordinate

# boundaries for bias subframe

bias_xmin          2
bias_xmax          4
bias_ymin          4
bias_ymax          2075
```

Along with the standard daemon settings, we also have a `driverpath`, `confpath` and `lockpath`. The `driverpath` specifies the location of the ARC camera driver. This is always set to `/dev/astropci0`. The `confpath` specifies the location of the DSP files to be loaded into the ARC camera hardware boards. These files should not be altered. The `lockpath` specifies where to put a lockfile when the camera driver is opened. If the camera readout crashes, this file is not removed. You *must* reboot the camera computer before removing the lockfile and restarting the daq system. Failure to comply with these instructions will result in a most hideous and painful death.[1]

After each image is read out, `camserverd` calculates some statistics for the image subframe and the bias subframe, specified above. These statistics include minimum, maximum, standard deviation, mean, and median, and can be a useful diagnostic for image quality and readnoise level.

```
# FITS header parameters
cdelt              0.0009                            # pixel scale (degrees)

# quality cuts on image subframe

stddev_cut         0.0                               # minimum allowed std. deviation
max_min            30000                             # highest minimum pixel value

# Miscellaneous
```

[1]Or rather, the camera computer will lock up and require a hard reboot, risking filesystem corruption.

```
docorr              0                                # don't run internal correction
amplifier           1                                # 0=left,1=right,2=both
symlinkpath         /rotse/data/3a1/links/           # location of symlinks
```

The `cdelt` header value is for World Coordinate System (WCS) positioning in the fits file. The quality cuts are useful for saving disk space. If all the pixels in the central subframe are near saturation, the image is assumed to be "bad," and only the header is written to disk. The system also calculates the number of saturated pixels in the image which can be a useful quick check of the focus quality. The `amplifier` variable specifies which amplifier should be used.[2] The readout time for dual-amplifier mode is 3s, but the ghosting of saturated pixels becomes a problem that we have not dealt with yet.

The `camserverd` system was written to be able to perform dark subtraction and flat fielding of the images in memory before they were written to disk. Unfortunately, this was too great a burden on the camera computer, and caused the system to become unstable. For this reason, `docorr` should not be turned on. Instead, a symbolic link to each image is put in the `symlinkpath`, where `sexpacman.pl` can find the files and process them automatically, as described in Section 8.2.1.

3.7 clamd.conf

The `clamd.conf` file is quite simple and needs no additional explanation:

```
# configuration file for clamd
#
# format is "key   value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued

loglevel            1                            # 0,1,2 = terse,verbose,debug
logfile             /rotse/run/log/clamd.log
poll_time           0.1                          # main loop time quantum
#
sample_time         3.0                          # sample time in seconds
email_list          "observer@rotse3a.lanl.gov" # send mail to on open/close
```

3.8 schierd.conf

The `schierd` controls the mount as well as the focus motor. It can model pointing either from a simple two-star model represented as a 3×3 matrix, or from a more involved model from the Tpoint package. The focus model comes from a simple linear model based on telescope temperature and elevation. For instructions for how to get a pointing model see Chapter 5. For a focus model, see Chapter 6.

The first part of `schierd.conf` is self explanatory:

```
# configuration file for schierd
#
# format is "key   value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued

loglevel            1                                    # 0,1,2 = terse,verbose,debug
```

[2] We use amplifier 1 rather than 0 because amplifier 0 inverts the image. It's just a bit easier this way.

```
logfile             /rotse/run/log/schierd.log
poll_time           0.1                                # main loop time quantum
sample_time         3.0                                # sample time in seconds
err_tout            5.0
#
mntman              AlanSchier                         # mount manufacturer
mntmodel            Equatorial                         # mount model
mntsn               2                                  # mount serial number
errormail           observer@rotse3a.lanl.gov          # where to send error messages
```

The next section contains numbers that should be general across the Schier mounts:

```
enctol              300                                # tolerance for encoder stop
slewacc             16.4   20.6                        # acceleration for mount (deg/s^2)
maxvel              35.0   35.0                        # slew velocity for mount (deg/s)
homevel             2.0     2.0                        # velocity to find home (deg/s)
stowpos             -85.0  35.0                        # Stow pos. in Axis coords (deg)
standbypos          -85.0 140.0                        # Standby pos. in Axis coords (deg)
deg2enc             24382   19395                      # encoder counts per deg
rarange             -185.0  0.0                        # range of the ra axis (deg)
decrange            0.0   220.0                        # range of the dec axis (deg)
foctol              0.001                              # focus tolerance
```

The acceleration and velocity parameters are stated in degrees for the RA axis (axis 0) first, and the Dec axis (axis 1) second. The `deg2enc` values were calculated from the radius of the encoder tape on each drive wheel. The RA range and Dec range were determined experimentally, and are just short of the hardware limit switches. Under normal operation, the mount controller should never be told to move the telescope to an illegal position, and we have multiple redundancies. The maximum velocities were determined by how fast we could move the telescope without any odd noises coming from the mount. (Very scientific, eh?) The home velocity is the default from Mr. Schier.

The stow position is (for the northern hemisphere) pointed north, slightly below the horizon. The standby position is very close to the zenith. These values can change depending on the polar alignment of the telescope.

```
# the following keyword defines a file that contains the 2-star rotation matrix
matfile             /rotse/run/etc/mat0529c.mat        # full path
# The following keyword defines a file that contains a tpoint model
modfile             /rotse/run/etc/mod_011211c.dat     # full path
focusmodfile        /rotse/run/etc/focusmod011214.dat  # full path
obsfile             /rotse/run/etc/observatory.conf    # site information
statdir             /rotse/data/pipeline/prod          # idlpacman stat dir.
statroot            rotse3a                            # root for statfile
```

The `matfile` matrix file specifies a two-star pointing model matrix file. The `modfile` is a Tpoint pointing model file. If a Tpoint model file is specified it will use this pointing model instead of the two-star matrix. The lines containing `matfile` and `modfile` can be commented out if no relevant pointing models exist. The modelling `schierd` decides to use is logged in `/var/log/rotse.log`. The `obsfile` is the observatory information file used for coordinate conversion.

The `focusmodfile` focus model file specifies the constants for the bilinear focus model fit. The format is described in Section 3.8.1.

Due to the homing inaccuracies of the Schier mount (the homing can drift by tens of arcminutes), the first calibrated image of the night is used to improve the home position. The automated calibration pipeline outputs a summary of each calibrated image in a file called `statdirstatroot_stat.dat`. After a home command, `schierd` checks every ten minutes for the most recent calibrated image that was taken after the most recent telescope homing, and updates the offset values. This improves pointing considerably.

3.8.1 The Focus Model Data File

The focus model consists of any number of constants for the focus fit, described in Section 6.2.2. The constants are derived from the `find_focus3.pro` program described in Section 6.5.

The focus model can consist of up to 20 terms. Each term can be an arbitrary polynomial combination of temperature (in degrees Fahrenheit), elevation (in degrees), and azimuth (in degrees). No significant azimuth dependance has been seen, and is not currently fitted.

```
# Focus model data file
#
# There can be up to 20 fit terms, with different polynomial degrees of
#  elevation "e" (in degrees), temperature "t" (in degrees fahrenheit)
#  and azimuth "a" (in degrees).
# The constant offset can be denoted with "1".
#
# Example:
# term  1       3.50    -- constant offset
# term  eet     0.00003 -- temperature*(elevation^2)
#
term       1       3.4426761
term       e       -0.0068352
term       ee      0.0000324
term       t       0.0044962
term       et      0.00005885
term       eet     -0.0000002498
```

3.9 spotd.conf

The configuration file for `spotd` is essentially the skeleton configuration file with a couple additions:

```
# configuration file for spotd
#
# format is "key   value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued

loglevel         1                             # 0,1,2 = terse,verbose,debug
logfile          /rotse/run/log/spotd.log
poll_time        0.5                           # main loop time quantum
sample_time      3.0                           # sample time in seconds

extra_pulses     30                            # extra time for mount shutdown

# The Following options are obsolete
temperaturefile  /dev/null                     # wdt501 watchdog temperature
wtempmax         120                           # temperature before logging
```

Aside from the obsolete temperature monitor options, there is an `extra_pulses` option for `spotd`. After a sudden system shutdown, this allows the mount time to park itself before the clamshell closes. This was more important for ROTSE-I where the mount did not have freedom to move with the clamshell closed. ROTSE-III does have this freedom, and therefore this option is less important.

3.10 userd.conf

The `userd.conf` file should be identical to the `astrod.conf` file, except for the logfile location.

3.11 weathd.conf

The `weathd.conf` file describes how the system receives its weather information, and what cuts are applied to determine when the weather turns "bad." The Weather Daemon can either run standalone, or as a client or server for more than one collateral telescope. The configuration file begins in the standard way:

```
# configuration file for weathd
#
# format is "key   value"
# comments start with '#' and go to end of line
# blank lines are okay
# lines cannot be continued

loglevel          0                              # 0,1,2 = terse,verbose,debug
logfile           /rotse/run/log/weathd.log
poll_time         0.1                            # main loop time quantum
init_time         50.0                           # how long wait for children
email_list        "observer@rotse3a.lanl.gov"    # send mail to
sample_time       3                              # sample time, in sec, for rotsed
sample_time_th    30                             # sample time in seconds, for threads
```

The two important threads are described next:

```
# Davis Weather Station monitor
#
dweath run        1                              # = 0/1 to run Davis weather
dweath tout       50.0                           # Davis Weather timeout
dweath tpoll      20.0                           # Davis weather station sample time
dweath twait      3600.0                         # time from last detection to okay given
dweath devfile    /dev/ttyC2                     # serial port for weath stat.

# Vaisala Precipitation monitor
#
vprecip run       1                              # = 0/1 to run Vaisala precip
vprecip tout      10.0
vprecip tpoll     1.0                            # Vaisala rain detector sample time
vprecip twait     3600.0
vprecip devfile   /dev/ml16pa-adc4
```

The Davis Weather Station is controlled via a serial interface, and reads the temperature, barometric pressure, windspeed, humidity, and has a rain gauge. The Vaisala Precipitation Detector is read from a PCI card, and will be coded to run with the CIO-CTR05 card that controls the Akerlof NightSky Monitor. This code has not been written, but better be there pretty damn quick!

In the near future `weathd` will also control the Nightsky Monitor. The configuration file contains legacy support for the old precipitation monitor, the "Wren Cloud Monitor," and a weather database. These are not implemented for ROTSE-III.

Any precipitation detected sets the weather to "bad," and the clamshell is automatically closed. The limits for weather station data are described at the end of the file:

```
tempi_min      -20.0                        # Min. Temp. Inside
tempi_max      100.0                        # Max. Temp. Inside
tempo_min      -20.0                        # Min. Temp. Outside
tempo_max      110.0                        # Max. Temp. Outside
dewdiff         5.0                         # max allowed difference between tempo and dewpt
wspd_min       -0.2                         # Min. Windspeed
wspd_max       25.0                         # Max. Windspeed
bar_min        20.0                         # Min. Bar. Pressure: set for 7000 elevation!
bar_max        28.0                         # Max. Bar. Pressure
humi_min       -0.1                         # Min. Humidity Inside
humi_max       200.0                        # Max. Humidity Inside
humo_min       -0.1                         # Min. Humidity Outside
humo_max       85.0                         # Max. Humidity Outside
rain_max       100.00                       # rain thresh. for davis sensor
maxsky          3.0                         # Akerlof sky monitor max. (not implemented yet)
minvprecip      2.5                         # Vaisala precip. detector min. (***)
mincloud        0.0                         # Wren cloud monitor min. (not implemented)
```

And finally there is the configuration information for the client/server options for `weathd`.

```
# Weather Server updater
# ( only sends badweather flag)
#
weathserver run      0                       # = 0/1 to run
weathserver tout     3.0
weathserver tpoll    1.0                     # sample time
weathserver twait    3600.0
weathserver devfile  3902                    # socket port number

# Weather Server2 updater
# (sends weather data: temp, humi, wind, dewpt)
# (disabled if weather station monitor is turned on)
#
weathserver2 run      0                      # = 0/1 to run
weathserver2 tout     50.0
weathserver2 tpoll    20.0                   # sample time
weathserver2 twait    3600.0
weathserver2 devfile  3903                   # socket port number

# Weather Client monitor
#
weathclient run       0                      # = 0/1 to run
weathclient tout      3.0
weathclient tpoll     1.0                    # sample time
weathclient twait     3600.0
weathclient devfile   rotsei.lanl.gov
numserver             0                # server number (0=none,1=server1,2=server2,3=both)
```

There are two weather server options. The plain `weathserver` serves the "badweather" flag only, usually from the Vaisala precipitation detector. The client need not know what went bad, only that it is damn important to close the clamshell as soon as possible. The fancier `weathserver2` serves Davis Weather Station data every few minutes. This allows the remote machine to use the outside temperature and pressure, which is essential for focusing.

As a weather client, the `weathd` can connect to one or both weather servers at a nearby location. This is useful if there is only one Vaisala Detector for more than one telescope, while each telescope has its own weather station.

This functionality was created for the early stages of ROTSE-III testing when it was sited next to the working ROTSE-I system. This will probably not be used at the remote ROTSE-III locations, although people at SSO or HESS are welcome to tap into our weather data.

Chapter 4

Rotse-III Scheduling

4.1 Introduction

The ROTSE-III scheduler is a modified queue scheduler implemented in `astrod`, described in detail in Section 1.2.8. Although the priority of a specific schedule item can be set (this is essential for fast alert response), `astrod` typically uses a scoring algorithm to decide which schedule item should be carried out next.

There are three levels of scheduling: guest user observations, prompt burst responses, and the regular telescope operations such as sky patrols. The instructions for these operations are stored in three files: `astrod_rtrig.conf`, `astrod_rsched.conf`, and `astrod_lsched.conf`. These files should only be altered directly by a qualified ROTSE technician. For guest observers, there is a Perl script, called `user_sched.pl` (See Section 4.6) that provides an interface to implement and manage their requested observations.

The standard ROTSE operational instructions consist of "trigger" definitions, which define a sequence of images to be taken, and "schedule" definitions, which provide a set of parameters to `astrod` to let it decide when and how to activate the trigger protocols. Standard triggers include burst responses, dark runs, runs to construct flats, and focus runs, as well as a "sky patrol" on a regular grid on a specified region of sky. Guest users may define triggers and schedules to carry out observing programs on specific desired targets. Each trigger has an associated three letter acronym (TLA) that becomes the root of the image filename for easy recognition and sorting later. How to set these various configuration parameters is described in this chapter.

The schedule is read in from `astrod.conf` on system startup (when `rotsed` is first launched), and near the start of each night's observing (when the sun dips below 5° above the horizon). In addition, although you should never do this under normal circumstances, sending a `SIG_HUP` signal to `astrod` erases the current schedule and re-reads the configuration file. This can be accomplished with the following (the process id is only an example):

```
$ ps auxww | grep astrod
root 20345 ...  /rotse/run/bin/astrod ...
$ kill -HUP 20345
```

I should write a little perl script that does this automatically.

4.2 `astrod` and `rotsed`

The details of how `astrod` communicates with `rotsed` are a bit complicated. Although the end-user should not need to know these details, they can create subtle errors and knowledge of the inner working will help facilitate any debugging that might be necessary.

The Astronomical Scheduler Daemon must send out commands, verify that they are received, and wait for them to finish. One of the main difficulties is the configurable nature of the system. For debugging purposes, the operator is not required to run one or more of the camera, mount, or clamshell. Therefore, a non-response from one of these daemons should not be assumed as a system failure.

When astrod sends a command, for example a MOVE comand for schierd, it fills a command structure and copies it to shared memory. At this point rotsed copies the command to a location in shared memory where schierd can receive it. Upon receipt of the command, schierd sets the system status MOVE bit "high." Now astrod knows the command has been received, and waits for schierd to reset the MOVE bit.

Unfortunately, if astrod does not wait long enough to see the MOVE bit set, it can assume the mount is not running, and can follow up with a PICTURE command. This results in a streaky image taking partially while the mount is still in motion. At the moment, astrod is set up with a min_wait configuration option. This is the minimum amount of time astrod will wait to see if a command is received. Currently set to 2 seconds, the optimal value depends on processor speed and total system load.

4.3 astrod.conf

4.3.1 Daemon Parameters

The general parameters are familiar from the other descriptions of the configuration files, plus the min_wait command from Section 4.2:

```
# Configuration file for Astronomical Scheduler Daemon =======================
#  ROTSE-III Version
#      The format is ``key   value''.  Comments start with `#' and go to end of
# line.  Blank lines are okay, and lines cannot be continued.
#
loglevel           2                            # 0,1,2 = terse,verbose,debug
logfile            /rotse/run/log/astrod.log
poll_time          0.05                         # main loop time quantum
sample_time        3.0                          # sample time in seconds
err_tout           150.0                        # max time in ALARM state
# -minimum wait period for 'rotsed' to acquire command
min_wait           2.0                          # (in seconds)
obsfile            /rotse/run/etc/observatory.conf
```

4.3.2 Exposure Parameters

The next section concerns the exposure parameters:

```
# -speed used during slewing
slew_spd           25                           # speed relative to max. (in percent)
slew_spd_alert     100                          # speed when initiating alert response
# -definitions of exposure lengths in seconds:
tshort             5                            # short exposure length
tmedium            20                           # medium exposure length
tlong              60                           # long exposure length
max_moon           0.50                         # max moon fraction to run a tlong;
                                                # otherwise tmedium is substituted
# -cuts on angular positions of frame, sun and moon
min_elev           20.0                         # min elev. for frame center (in deg.)
sun_elev           0.0                          # max elev. of sun (in deg.)
moon_dis           30.0                         # min distance to moon (in deg.)
```

First, the slew speeds are specified. For both normal slews and alert slews, the percentage of the maximum (as defined in schierd.conf) is given.

Next, the exposure length parameters. We have decided on standard values of 5 seconds, 20 seconds, and 60 seconds. Normal patrol operations use long (60s) exposures. However, when the moon is bright, the sky brightness can nearly saturate a long 60s exposure. Therefore, when the moon is up at its fraction is greater than `max_moon`, the exposure lengths are automatically stopped down to tmedium. Note, the system *will not* automatically take long exposures when the moon is brighter than that specified, either for regular sky patrols or for burst responses.

Finally, general cuts are made on what the telescope will image. A field must be at least a certain altitude above the horizon: an azimuthially-dependent number due to the elevation angle of the Declination limit switch on the mount yoke. Simply put, the telescope cannot swing too far opposite the pointing axis, or it will hit the yoke. In other directions, it can image closer to the horizon. The sun must of course be below a designated negative altitude, and the field must be at least 30° from the moon. These cuts are usually only relevant for prompt burst responses. More stringent cuts can and should be used for normal schedule items.

4.3.3 Sky Patrol Parameters

This section puts limits on the sky patrol:

```
# -sky patrol scheduling parameters
long_names        1                    # 0/1: Use long names for the sky patrol
field_of_view     1.9                  # field of view (in deg.)
min_overlap       0.1                  # min. overlap of generated fields
ra_lim            0.0,24.0             # ra lim (in dec. hrs) for sky patrol
dec_lim           -1.5,1.5             # dec lim(in dec. deg) for sky patrol
glat_cut          30.0                 # galactic latitude cut (deg.)

ndarks            6                    # num. darks of each length for a dark run
foc_lim           3.04,3.50            # Limits for focus run
foc_step          0.02                 # Stepsize for focus run
focstandby        3.25                 # standby focus position
```

The sky patrol grid is created by an algorithm that tiles the whole sky with the given field of view (in degrees) and minimum overlap (in degrees) between fields. After the entire sky is gridded, the sky patrol list is cropped according to the RA limits and Dec limits specified. Finally, the list is cropped again to remove low galactic latitude fields below `glat_cut`. There can only be one sky patrol list of fields specified. It is strongly recommended to keep the sky patrol region small enough that the same fields are imaged every night for at least a month. With no constraints, the scheduler will only image the lowest airmass fields, which change from night to night.

The paramter `ndarks` specifies the number of darks of each length taken in a dark run. The focus limits and focus step describe the focus settings used in a focus run. The focus limits used depend on the configuration of the system and the temperature range expected during the night. For more information please see the focus how-to documentation. Finally, the focus standby position is the default focus position when the mount is in standby mode. This should be close to the median focus position for faster gamma-ray burst alert responses.

4.3.4 Burst Follow-up Parameters

The scheduler will automatically schedule follow-up images to burst alerts, and these parameters are described here:

```
burstfile         /rotse/run/etc/burst.conf   # Burst data logfile
live_days         3                    # Number of days to follow up burst
fup_priority      10                   # Priority of follow-ups
fup_trigger       follow_up            # imaging sequence for follow-ups
fup_sunel         -15.0                # Max Sun elev. for follow-ups
powerlaw          -1.5                 # Assumed powerlaw for image intervals
fup_frac_change   0.1                  # Predicted flux change to schedule followup
```

```
photfile            /rotse/run/etc/landoltfields.conf # List of photometry fields
```

Each burst alert that is received is logged to the burstfile. For more information on the format of this file please see Section 4.5.

The scheduler will plan follow-ups for live_days after a burst. After that it is ignored in the burst file. The imaging sequence for a follow-up is described by the fup_trigger keyword. Currently set to 30 long exposures, see Section 4.4.1 for information on how to change this.

The timing of the follow-ups is based on an assumption of a power-law decay of index powerlaw. We only plan to image a burst when the counterpart has a chance to fade significantly. The minimum fractional change of the flux can be set with fup_frac_change.

After every burst response and burst follow-up, the telescope automatically takes an image of a photometry check field on the equator with a large number of Landolt standard stars. These fields are described in the photfile file. The photometry trigger sequence is described in Section 4.4.1.

4.3.5 Scheduling Parameters

These are the constants used for calculating field scores:

```
cadence_const      10.0                        # Importance of cadence in scoring
targ_const         10.0                        # Importance of specified targets
elev_const          5.0                        # Importance of airmass
dec_const           5.0                        # Importance of hard-to-image fields
```

The default values seem to work quite well, although they may be tweaked in the future. Please note the score for a field is only calculated if there are no higher priority fields to image, and if the field is not vetoed for reasons of elevation, moon position, etc.

4.4 Triggers and Schedules

The way the ROTSE-III system plans observations is through defining a “trigger” type, which is a protocol for a sequence of images to be taken, and then any number of “schedule” items can call on a given trigger protocol in a highly configurable set of circumstances. A trigger type might say “take six images at 20 seconds each”, but the schedule item tells where to point the telescope, how often, and under what conditions to take those six images. There are a number of standard ROTSE triggers that are stored in the file astrod_rtrig.conf and standard schedule items are stored in astrod_rsched.conf. The rest of this section uses the contents of astrod_rtrig.conf and astrod_rsched.conf to explain how triggers and schedules work, as well as show the command syntax. Guest users should be familar with these formats, but should never modify the system files. Guest user triggers and schedules are defined in the file astrod_lsched.conf, and this file can be modified with a Perl script, as explained in Section 4.6.

4.4.1 Triggers: Defining Imaging Sequences

This is the imaging sequence (“trigger”) description from the astrod.conf file:

```
#       Each trigger has a name, a three-letter-acronym ("tla"), a
# livetime (how long to hold in the queue before giving up), and a
# scripted protocol which describes the kind of observation sequence to
# be performed.  The observations are described compactly with the
# following vocabulary:
#    js = 'jiggled, short' type of exposure
#    ts = 'tile, short' (ie. tile = rastered 2x2 box around coord.)
#    jm = 'jiggled, medium'
#    tm = 'tile, medium'
#    jl = 'jiggled, long'
```

```
#    tl = 'tile, long'
#    aoff = turn current alert off
#    calib = do dark run
#    fr = do focus run
#
#    example: 10jl = take 10 long, jiggled exposures.
#
# Note: the number before each element indicates the number of times to
#       run that element.  If you spevify "12ts", it will take 12 tiled
#       short exposures, covering the entire area only 3 times. (12/4tiles)
#          Name             TLA    Lifetime     Protocol
#          ----             ---    --------     --------
#
trigger    sky_patrol       sky      0           2jl
trigger    dark_run         drk      0           calib
trigger    focus_run        foc      0           fr
trigger    home_check       hom      0           1js
trigger    twi_flat         twi      0           1jm
trigger    pointing         tpt      0           2jm
trigger    follow_up        fup      0           30jl
trigger    photometry       pht      0           2jl
```

The information from the configuration file is a good explanation of the format, and the is trigger items available are summarized in Table 4.1. See Section 4.6 for step-by-step instructions for defining and implementing as a guest user your own triggers and three-letter acronyms for supernova searches, AGN monitoring, and other monitoring schemes. A couple of comments should be made about the operation. The lifetime parameter should be set to 0 for normally scheduled triggers. This parameter is only used in burst alerts triggers (see below).

Jiggling is the normal mode of operation. During jiggled exposures a random offset of up to 10 pixels is added to the position of each image. This greatly reduces the number of false detections due to hot pixels.

The special trigger sequence `calib` indicates a dark run. During a dark run `ndark` dark exposures are taken of each tshort, tmedium, and tlong exposure times.

The special trigger sequence `fr` indicates a focus run. During a focus run, images are taken at `foc_step` intervals in the boundaries defined by `foc_lim`. For more information on how to create a focus model see the focus how-to documentation.

The special trigger command `aoff` turns off a current running alert in `alertd`. Once `aoff` has been run, any new trigger will interrupt the current observing sequence.

There must be a `photometry` item in the trigger list for Landolt field alert follow-ups. If this item is missing the scheduler will not start up and an error is logged to `/var/log/rotse.log`.

```
# Test triggers
trigger    tla_test_alert   tla      1           2jm,aoff
# -HETE GRB trigger response
trigger    grb_hete_alert   gha      600         10js,10jm,50jl,aoff
trigger    grb_hete_update  ghu      600         10js,10jm,50jl,aoff
trigger    grb_hete_final   ghf      600         60jl,aoff
trigger    grb_hete_ground  ghg      600         60jl,aoff
```

These are the standard burst alert triggers. Every alert response will be followed by a sequence of follow-up images as defined in Section 4.3.4. For every alert type defined in `alertd.conf` there must be a corresponding trigger entry in `astrod.conf`. If a valid burst alert arrives and it is not configured here, then only a sequence of follow-up images will be taken.

The lifetime here specifies the maximum time (in minutes) to wait for a burst field to become available before scrapping the prompt response. A burst field might not be available due to the sun, elevation, or

Abbreviation	Name	Description
js	jiggled, short	short exposures, jiggled <10 pixels from field center
ts	tiled, short	short exposures, rastered 2×2 box around field center
jm	jiggled, medium	medium exposures, jiggled
tm	tiled, medium	medium exposures, tiled
jl	jiggled, long	long exposures, jiggled
tl	tiled, long	long exposures, tiled
aoff	alert off	turn system alert mode off
calib	dark run	dark calibration run. ndark dark exposures taken for each exposure length
fr	focus run	take evenly spaced focus images of a given field. See Section 4.3.3.

Table 4.1: Listing of the various trigger options in astrod.conf, and their description

weather. Some bursts in the wrong hemisphere will never become available, but are held in the queue anyway. If a prompt response is deleted, follow-up sequences will be run as soon as the field is available.

4.4.2 Defining Schedule Items

This is the actual definition of the schedule. In general, you should always use the Perl script (See Section 4.6) to process your schedules, but this information will enable you to construct your schedules as desired before submitting them:

```
# -----------------------------------------------------------------------
#     Schedule List - This should be useful to folks
# -----------------------------------------------------------------------
# Format:
#         sched  trigger  "flags"
# eg:
#         sched  pointing  "-r 10,0 -d 80,0 -e 2000.0 -f 120"
#
# Schedule items are queued and scored on their relative merits (elevation...)
#  -can set priority which overrides scores.  Alerts have priority 100+.
#
# The flags are:
#     -r hr,min    :  RA, with no space between hr,min
#          -d deg,min   :  Dec, ""
#          -e epoch     :  epoch
#          -a azimuth   :  Azimuth in degrees (useful for focus, etc)
#          -l elevation :  Elevation in degrees
#          -t max_times :  Max # times to run the schedule item
#          -p priority  :  priority.  Most should be default (0)
#          -L user ID number  :  The local user's ID number
#          -m min_elev  :  Only image if it is above min. elevation
#          -u max_sun   :  Max. elevation of the sun for the item
#          -s           :  Is a Sky Patrol.  These have less weight
#          -i interval  :  Minimum time (minutes) between running
#          -c cadence   :  Exact time (minutes) between running
#          -T           :  Check camera temperature at target
#          -M min_dmoon :  Minimum distance to the moon for exposure
#          -S           :  check if the moon has set
```

```
#          -P           :  do a "photometry" frame
#
#  Obviously, you cannot specify both ra/dec and az/el!
#---------------------------------------------------------------------------
```

Each schedule item is tied to a specific trigger. If a schedule item is given without a corresponding trigger, an error will be logged to /var/log/rotse.log and astrod will shut down.

When specifying a trigger field, the operator can either specify RA/Dec or Az/El. This second mode is used for focus runs. Schedule items with specified positions are given more weight than sky patrol items under the assumption that if the operator cared enough to type it in, it must be important. The additional weight is given by the value targ_const in Section 4.3.5.

The difference between a cadence -c and an interval -i is the weight given by cadence_const. If a cadence is specified, it means that the field should be imaged at that specified cadence, which becomes more important than airmass. If -t 1 is specified then the cadence and interval have no meaning. You can only specify one of these three options, or an error will be logged and the system will not start.

A local user must set the priority -p option, as well as the -L user ID number. The photometry option -P is also not frequently used.

The options -u,-m,-M,-S are used to override the system defaults. It is always important to remember the -u option, or the images might be taken in early twilight before it is dark!

```
sched sky_patrol  "-s -u -18.0 -t 3 -c 30 -S -m 30.0"      # skypatrol (-s)
sched home_check "-u -10.0 -t 1 -r lmst -d 0,0 -e 2000.0"
sched dark_run "-t 1 -u -15.0 -i 300 -T"
```

In the default schedule listed above, it should first be noted that the order of schedule items or option parameters is *not* important.

The sky_patrol item is designated by the -s option. The sky patrol will only begin when the sun is below $-18°$. Each field will be imaged 3 times with a cadence of 30 minutes. The moon must be below the horizon to do the sky patrol (-S), and the field must be at least 30° elevation.

The home_check item runs just once, when the sun is 10° below the horizon. It points at 0° declination, at the local mean sidereal time, or due south on the equator. This single short image is calibrated and is used by schierd to check the home position. Any calibrated image can be used for this purpose, but it is nice to have a single image taken before any sky patrols or burst responses.

The dark_run item runs just once, when the sun is 15° below the horizon, when there is no stray light. If the camera has not reached its target temperature (from the -T option), the scheduler will wait to take the dark calibration run.

The actual order of operation here would be the home check, the dark run, and then the sky patrol for the whole night. Any target fields specified would interrupt the sky patrol where appropriate.

4.5 burst.conf

The burst.conf file can be used to input burst coordinates for follow-up when no GCN alert was received. This can happen because the system was down, or because only a GCN circular was issued, with no socket notice.

```
# Burst configuration file
# --the comments cannot be changed
#
#grb  date  type  num.   R.A.       Dec.    Epoch  Err.  UT    Observations
#                       (dec.deg) (dec.deg)        (')  (dec.hr)
#---------------------------------------------------------------------------
grb  011122  34       0 173.62540 -76.03740 2000.0   5.0  0.7828
```

This file is read in by `astrod` at the start of every evening. The operator can force a re-reading of the entire schedule (including `burst.conf`) by sending a hangup signal as described in Section 4.1.

Each burst is given a date, type, serial number, position, error (in arcminutes) and UT at time of burst in decimal hours. The observations column is filled in by the scheduler and should not be altered by the operator. The scheduler will then use the timing information to schedule follow-ups as described above in Section 4.3.4.

4.6 Guest User Schedule Submission

In order to avoid the confusion of having multiple users all trying to modify the same `astrod.conf` file to request their observations, we've established a "scheduler" account to which you can log in for the purpose of account management. Within the home directory of that guest account, you should create a directory for yourself, to avoid confusion with other guest users. Each guest user should be assigned a user number from 1 to 50 by the host institution; you may wish to use this in your directory name. It will become critical later on.

All of your interaction with the scheduler files should be through the Perl script `user_sched.pl`, which will be in the path for the guest account. Call it without any options to get a simple description of its usage. In order to schedule ROTSE-III observations, you will write your own local schedule file containing triggers and schedules (as defined above), and you must get this script to agree that there are no mistakes or problems within your list. The script cannot, of course, confirm that your schedule will perform the observations you have in mind, if you have misunderstood the command formatting, but it will ensure that you can't crash the system or confuse your observations with others'.

This is what you will see if you run the script with no options:

```
Welcome to the ROTSE-III scheduler script.

Options may be in any order.
Use -e to extract your schedule to your local directory.
Then modify that file to match what you want to schedule.
Then use -s to submit your modified file.
If you do not specify a file name, local_user_file.txt is assumed.
Make sure the first line of this file is your user_cap.
Use -t with -s to test submission but not to modify lsched.
For a test submission, output is written to local_test_file.txt in the working directory.

Syntax: user_sched.pl -u <Your user ID number> [-s|e] [optional file name] [-t]
```

To get more than the usage instruction from the script, you must use the `-u` option to give it your user number. The script can then perform two operations: it will extract all triggers and schedules from the standard configuration file `astrod_lsched.conf` into a local file in your directory, or it can insert your local file into `astrod_lsched.conf` to be used in planning the night's observations. You distinguish between these two operating modes by using either the `-s` or `-e` command line options. One of these two options must be present. You may include a file name after this option call, if you wish. If not, the script will assume a default file name of "`local_user_file.txt`". There is a final option that you can use, `-t`, which runs in conjunction with option `-s`. It will check your local file for submission, but instead of replacing the official configuration file with your updated version, it will write the updated version of `astrod_lsched.conf` to your local directory under the name "`local_test_file.txt`". Use this option if you want to test your schedule file for errors, but don't want to actually put your instructions to the telescope yet.

You should always begin your session by running in extract mode, even if you have never used it before. The returned file will always start with a line called "`user_cap`". This line establishes what percent of the nightly observing time has been allocated to you. The first number on the line will be your user number. The second number should be your percentage (a number between 0 and 30), and you can put any information you like after a number sign. Your real name would be a useful thing to put in this comment area. The "`user_cap`" line should always come first in your schedule file.

If you run "`user_sched.pl -u 5 -e`", your first local schedule file will have the single line:

```
user_cap 5 [your allowed percentage] # [comments, like your name]
```

Once you have a `user_cap` line in place, you can define as many triggers and schedules as you want. When you run `user_sched.pl` in extract mode, it reports a list of all the triggers that have already been defined by other people. You may not use any of these trigger names or three-letter-acronyms. Construct your trigger types and schedules as described above in Sections 4.4.1 and 4.4.2. The only difference is that your trigger type definitions must include "`userNN`" (where "`NN`" is your user ID number) in the comment field. If you omit this, the Perl script will insert it for you, and remind you to include it in the future. Also make sure you include your user number (`-L NN`) and schedule priority (`-p #`) in the options for your schedule items. The trigger name given in your schedule item must match one of the trigger names given earlier in this file. In general, the order of your line items in this file is irrelevant, except that a trigger item definition must come earlier than the use of that trigger name in a schedule item.

When you are satisfied with your schedule file, you may check it for errors by running `user_sched.pl` in test submit mode (`-s [file name] -t`). The script will print to standard error a running description of its progress as it evaluates your file. Please keep a close eye on these output lines for errors. Any errors discovered *should* (we hope) be clearly reported for easy fixing. If the script finds no errors, it will write a complete local schedule file (`local_test_file.txt`) to your local directory for your perusal. If everything looks in order, run `user_sched.pl` one last time without the `-t` option, and you are done.

A sample local user schedule file might look like this:

```
user_cap   5   25    # I am Simon, my ID number is 5, and I get 25% of the time
trigger  simons_kvars  skv  0  2jm,4js # user5 -- I want 2 20 s exposures and 4 5 s
trigger  supnovae  sne  0  2jl # user5 -- I want 2 60 s exposures
sched  simons_kvars "-r 23,12 -d -12,25 -e 2000.0 -p 2 -L 5 -i 1000 -m 45"
sched  simons_kvars "-r 10.5 -d 10,2 -e 2000.0 -p 1 -L 5 -i 1000 -m 45"
sched  supnovae "-r 1 -d 56 -e 2000.0 -p 1 -L 5 -i 1000 -m 45"
```

If you were to submit such a file, using the command "`user_sched.pl -u 5 -s -t`", you would see the following output:

```
Submit mode: Sched file local_user_file.txt will be submitted.
Reading file user5file.txt
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Checking: "user_cap   5   25    # I am Simon, my ID number is 5, and I get 25% of the time"
Line clears successfully.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Checking: "trigger  simons_kvars  skv  0  2jm,4js # user5 -- I want 2 20 s exposures and 4 5 s"
---------------------
Parsing trigger type : "trigger  simons_kvars  skv  0  2jm,4js "
This trigger first will take 2 jiggled medium exposures.
And then it will take 4 jiggled short exposures.
---------------------
Line clears successfully.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Checking: "trigger  supnovae  sne  0  2jl # user5 -- I want 2 60 s exposures"
---------------------
Parsing trigger type : "trigger  supnovae  sne  0  2jl "
This trigger first will take 2 jiggled long exposures.
---------------------
```

```
Line clears successfully.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Checking: "sched  simons_kvars "-r 23,12 -d -12,25 -e 2000.0 -p 2 -L 5 -i 1000 -m 45""
**********************
Parsing schedule options "-r 23,12 -d -12,25 -e 2000.0 -p 2 -L 5 -i 1000 -m 45"
Parse successful.
**********************
Line clears successfully.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Checking: "sched  simons_kvars "-r 10.5 -d 10,2 -e 2000.0 -p 1 -L 5 -i 1000 -m 45""
**********************
Parsing schedule options "-r 10.5 -d 10,2 -e 2000.0 -p 1 -L 5 -i 1000 -m 45"
Parse successful.
**********************
Line clears successfully.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Checking: "sched  supnovae "-r 1 -d 56 -e 2000.0 -p 1 -L 5 -i 1000 -m 45""
**********************
Parsing schedule options "-r 1 -d 56 -e 2000.0 -p 1 -L 5 -i 1000 -m 45"
Parse successful.
**********************
Line clears successfully.
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
Congratulations!  I could find no errors in your submitted file.
Writing submission to local_test_file.txt
```

Please note that this final submission *must* be completed before the sun falls to 5° above the horizon at the telescope site, or your schedule will *not* be read into the DAQ for the evening's observing procedures. Also be aware that it's quite possible for someone to submit their file after you have extracted, but before you have resubmitted, yours. If that person happens to use a trigger name or TLA that you are adding to your schedule, you will get an error when you resubmit, despite the fact that when you checked out, the trigger was free.

Chapter 5

Polar Alignment and Pointing Models

5.1 Introduction

This chapter contains (at the moment rough) instructions for polar alignment of the telescope, as well as constructing crude and sophisticated pointing models. Any questions should be sent to erykoff@umich.edu.

Polar alignment requires a couple of strong people and two car jacks to safely raise and lower the telescope.

Please see Chapter 2 for information on starting the system and using the `rush` shell environment.

Once the telescope is polar aligned and a basic pointing model has been calculated (either from two stars or multiple stars), it is easy to bootstrap a more complicated pointing model. This method is described in Section 5.4.

5.2 Polar Alignment

5.2.1 General Information

Once the telescope is initially aligned with the eye, the key to fixing the polar alignment is measuring the alignment from the celestial sphere. Finding good stars can be a hassle, however. For initial star-finding, a TelRad attached to the OTA is essential.

This section is divided into information for TelRad alignment, the Two-Star Pointing method, Tpoint Pointing method, and finally tips on how to adjust the telescope polar alignment. A Two-Star model can be created much faster than a Tpoint model, which requires at least 4 or 5 stars.[check this!] The disadvantage of the Two-Star method is that it requires an external measurement of the telescope pole in encoder counts. Two ways to make this measurement are described in Section 5.2.3.

5.2.2 TelRad Alignment

The first task is to align the TelRad for quicker starfinding. Note that adjustment of the secondary tilt for optical alignment purposes causes the TelRad to go out of alignment quickly. However, once the telescope has been polar aligned you should not need the TelRad again.

The outer ring through the TelRad sight is 1 degree, and the inner ring is 0.5 degrees. [Perhaps]

TelRad Alignment:

1. Identify a bright star on the sky that is above the horizon.

   This can be accomplished with a star chart or a knowledgeable individual. It will be necessary to know these coordinates for pointing model creation.

2. Move the telescope using `rshift` to point at the star.

   The star should be as close to the center of the TelRad as possible without taking too many iterations.

3. Take a short exposure of the star:

   ```
   rush> rshift -t              ; this starts tracking if it isn't already
   rush> rexpose -t 2 -n star1
   ```

   A couple of seconds is plenty of time to image a bright star. To view the image, open a separate terminal window and type something like the following:

   ```
   $ cd /rotse/data/3a1/        ; this could be 3b1, or 3c1 depending on the system
   $ ds9 date_star1_3a001.fit
   ```

   Hopefully the star is in the image! If it is not, then you just have to play around with jiggling a degree or so. If it is, then...

4. Center the Star on the CCD

   Move the telescope with `rshift` until the star is centered on the CCD. Remember, the field of view is 1.9°. Depending on the orientation of the camera, you will have to figure out which axes move the star in which direction. Once these have been discovered, a little post-it note with the axis orientations is helpful to affix to the enclosure monitor.

5. Record the star with `rstarlog`

   This star can act as the first star in the pointing model, so record its position to the log.

6. Adjust the TelRad

   Adjust the TelRad so that the laser sight is centered on the star, which is centered on the CCD. This will make the following steps much easier.

5.2.3 Two-Star Pointing

Finding the Telescope Pole Offset - Mechanically

The telescope axis can be determined by rapidly slewing the telescope along the RA axis during an exposure, at 50% maximum slew speed. If the telescope is pointing along the dec axis, the stars should create circles around the center of the CCD. Finding this position can take a few trials. It need not be precise, (up to $\sim 0.5°$ is fine, for there are other large systematic errors in the two-star rotation matrix.

When the telescope is pointing along its axis, record the axis 1 encoder position using `rstat` described in Section 2.5.4. Convert this to degrees by dividing by the conversion factor in `schierd.conf`, which should be 19395. Subtract the converted value from 90°, and you have the offset.

Finding the Telescope Pole Offset - with Tpoint

If a Tpoint model has already been constructed with several stars, the polar offset value is in the Tpoint model file created as in Section 5.3. The `ID` value described in Section 5.5 is the declination axis offset in arcseconds. To use this value as the telescope pole offset for two-star pointing, simply divide `ID` by 3600. For the prototype (3a) telescope this is around $-3.9°$. Each telescope will have a different value because this number depends on the precise location of the dec axis homing limit switch.

Calculating a Two-Star Matrix

The steps here are very similar to those in Section 5.2.2, where the individual steps are detailed. We assume that the operator has already logged the first star from Section 5.2.2.

1. Identify a bright star far from any previously logged stars.
2. Move the telescope using `rshift -t` to point the TelRad at the star.
3. Take a short exposure of the star with `rexpose`.
4. Repeat until the star is centered on the CCD.

5. Record the star with `rstarlog`.

6. When two stars have been recorded, output the file with `rstarwrite`.

7. Run the interactive idl program `twostartp.pro`.

   This program calculates a two-star matrix using a file written by `rstarwrite`. It can also calculate the telescope pole location. The syntax is as follows.

   ```
   IDL> twostartp
   syntax - twostartp,starfile,rconv=rconv,dconv=dconv,lon=lon,the_offset=the_offset
      Reads in tpoint formatted files
   ```

   The `lon` option is the telescope longitude, in degrees. The default is LANL. The `the_offset` option is the telescope polar offset as determined in Section 5.2.3. The default is -3.8°, for the 3a telescope. After starting, the program presents a menu of options.

8. Chose option 'c' to find the telescope pole.

   This is where some experimentation needs to be done to figure out whether the numbers mean the telescope should be moved up, down, east or west. This will be added to later revisions of this chapter.

9. Chose option 'f' to save the matrix file to disk.

5.2.4 A Simple Tpoint Model

To create a simple Tpoint model, begin by following steps 1-5 in Section 5.2.2. Continuing from there, follow the steps below:

1. Repeat for several stars, certainly at least 4, perhaps 5.

2. Output the file with `rstarwrite`.

3. Create a pointing model as described in Section 5.3.

4. Use the `ME` and `MA` values in the model to find the polar offset.

   The units of `ME` and `MA` are in arcseconds. Here I quote from the Tpoint documentation:

   - `MA` Polar Axis Misalignment in Azimuth

     In the northern hemisphere, positive `MA` means that the pole of the mounting is to the right of due north.

     In the southern hemisphere, positive `MA` means that the pole of the mounting is to the right of due south.

   - `ME` Polar Axis Misalignment in Elevation

     In the northern hemisphere, positive `ME` means that the pole of the mounting is below the true (unrefracted) pole. [This is the alignment we have been using, which is easier for our wide-field systems.]

     In the southern hemisphere, positive `ME` means that the pole of the mounting is above the true (unrefracted) pole.

5.2.5 Adjusting the Polar Alignment

To raise and lower the telescope, using the car jacks is very handy. Just place the carjacks under the telescope bars, to stabilise and move the telescope while the bolts are unfastened. To move the telescope left and right requires a strong back and a lot of luck. There are usually a few iterations before the alignment is acceptable.

If Tpoint was used for the first iteration to calculate the mechanical offset, then the two-star method can be used for the following iterations of the polar alignment. We ran quite well with a 0.7° polar offset at LANL with no noticable ill-effects. The modelling takes the misalignment into account, and the telescope motor is able to track both axes. A permanent installation would probably desire better polar alignment, however.

5.3 Using Tpoint

The Tpoint (TM) program by Patrick Wallace is a tricky beast. After much trial and error, I have come up with a way of getting a pointing model that is quite usable, especially when the data from the whole sky is available. I describe here my ad-hoc method of using Tpoint to get a satisfactory pointing model. If you make a mistake, the easiest thing to do is quit Tpoint and restart the fitting procedure. This section is meant to be used in conjunction with Section 5.2.4 or Section 5.4. What is done with the output model depends on whether or not the telescope has been polar aligned.

1. Star Tpoint In the directory with the .dat file output by `rstarwrite`

   ```
   $ tpoint
   ...
   TPOINT ready for use:  type HELP for assistance, END to quit.
   *
   ```

2. Load in the Tpoint data file.

   ```
   * indat file.dat
   ROTSE-III Prototype
   :EQUAT
   :NODA
   :ALLSKY
   35 52 8.4  2001 5 30

   14 16  0.00  19 12  0.00 20 20 30.00 164  4   0.00 13 40.87
   ...
   ```

   A list of all the stars logged should be printed to the screen.

3. Set Tpoint to adjust star positions to fit telescope positions. This is the modelling that `schierd` uses.

   ```
   * adjust s
   Model will adjust stars to fit telescope.
   ```

4. First, have Tpoint fit to the index offsets for the ra and dec axes. The “use” feature tells the Tpoint to fit the stars to these particular variables.

   ```
   * use ih id
   ```

5. Next, do the fit to these parameters. The `fit` command should be run several times, until the errors reported stop shrinking.

   ```
   * fit
          coeff       change     value    sigma
     1     IH         -0.000-319288.44  852.791
     2     ID         -0.000 -12374.29  654.834
   Sky RMS =1464.25
   Popn SD =1890.34
   ```

   When the “change” column sits at 0, you can move on.

6. Fix the index offsets with `fix`, and now have Tpoint use the collimation and polar alignment errors for its next fit.

   ```
   * fix ih id
   * use np ch me ma
   ```

7. Fit with the `fit` command several more times until the change goes to 0, as before.

8. Use the whole suite of parameters, by unfixing `IH` and `ID` and `fit` several more times until Tpoint settles on a solution.

   ```
   * use ih id
   * fit
   ...
   ```

9. Save the model to disk and exit.

   ```
   * outmod modelfile.dat
   * end
   ```

10. The model file can now be viewed with a textfile viewer to extract notable parameters, or can be used verbatim by `schierd.conf` as described in Section 3.8. The model file should look something like the following:

   ```
   ROTSE-III Prototype
   S  1862248.2854    0.000   0.0000
     IH      -320481.5133   114.58002
     ID      -15074.2754    15.45639
     NP        -284.0514    24.71350
     CH        -356.0703   109.36628
     ME       -2233.0871    18.88705
     MA        -728.2158     9.84367
   END
   ```

   The column to the right of the fit parameter names contains the fit values, in arcseconds. The third column contains the rms error in the fit for that parameter.

5.4 Refining the Pointing Model

Once the telescope has been polar aligned and a rudimentary pointing model has been created (either with the two-star matrix or Tpoint), refining the model is easy. Since the ROTSE-III telescopes have such wide-fields, astrometric calibration can be done from the bright USNO A2.0 catalog stars in the field, and superfine accuracy is not essential. Pointing errors of over 30 arcminutes are usually handled comfortably by the calibration software, but are not desirable for normal operation.

With a simple pointing model installed in the `schierd.conf` configuration file, the telescope can point to celestial coordinates accurately with `rmove` and the astronomical scheduler daemon. With a single night of all-sky imaging, a more refined pointing model can be bootstrapped on top of the simple model.

The `astrod.conf.pointing` sample file contains a list of hundreds of fields evenly gridded across the entire celestial sphere. Whatever fields that are visible from the local site will be used to image the whole sky. An idl program called `cobj_to_tp.pro` converts the calibrated file output to a form to input into Tpoint, and a refined pointing model can be created. Here are the steps to accomplish this:

1. Copy the pointing fields from `/rotse/run/etc/astrod.conf.pointing` into the current `astrod.conf` file.

2. Run on a reasonably clear night. Moon illumination is not important, unless the all-sky data is also being used to calculate a flatfield image.

   For information on how to run the system and start the pipeline, see Chapter 2 and Chapter 8.

3. Copy the object list (`cobj`) files from the pipeline directory to a temporary directory.

   The pipeline directory should be set to `/rotse/data/pipeline/prod/`.

4. Create a text file with the list of `cobj` files.

   ```
   $ cd temporary_directory
   $ ls --color=never date_*cobj.fit > cobj.list
   ```

5. Start idl and run `cobj_to_tp.pro`.

   ```
   $ idl
   IDL> cobj_to_tp
   syntax - cobj_to_tp,listfile,outname,ptgstr,...
   IDL> cobj_to_tp,'cobj.list','tpointfile.dat',pointing_structure
   ```

6. The file `tpointfile.dat` (or whatever you chose to name it) now has hundreds of stars for Tpoint to create a pointing model.

7. Run Tpoint on `tpointfile.dat` as described in Section 5.3

8. Update `schierd.conf` as described in Section 3.8 to use the new Tpoint model file.

5.5 Tpoint Formulas

This is taken verbatim from the Tpoint (TM) manual by Patrick Wallace.

Here I describe the formulas that are related to the pointing model implemented in `schierd` for the ROTSE-III telescope. This is for quick reference only, and details can be found in the Tpoint manual.

- `ID` Index Error in Declination

  Declination index error in an equatorial mount: the zero-point in δ.

  $\Delta\delta = +\mathtt{ID}$

- `IH` Index Error in Hour Angle

  Hour angle index error in an equatorial mount: the zero-point of h.

  $\Delta h = +\mathtt{IH}$

- `NP` HA/Dec Non-perpendicularity

  In an equatorial mount, if the polar axis and declination axis are not exactly at right angles, east-west shifts of the image occur that are proportional to $\sin\delta$.

  $\Delta h \simeq +\mathtt{NP}\tan\delta$

- `CH` East-West Collimation Error

  In an equatorial mount, the collimation error is the non-perpendicularity between the nominated pointing direction and the declination axis. It produces an east-west shift that is constant for all declinations.

  $\Delta h \simeq +\mathtt{CH}\sec\delta$

- `ME` Polar Axis Misalignment in Elevation

  Vertical misalignment of the polar axis of an equatorial mount: a rotation about an east-west axis equal to coefficient `ME`.

  $\Delta h \simeq +\mathtt{ME}\sin h\tan\delta$

  $\Delta\delta \simeq +\mathtt{ME}\cos h$

- `MA` Polar Axis Misalignment in Azimuth

  Misalignment of the polar axis of an equatorial mount to the left or right of the true pole: a rotation about an axis through ($h = \delta = 0$) equal to coefficient `MA`.

  $\Delta h \simeq -\mathtt{MA} \cos h \tan \delta$

  $\Delta \delta \simeq +\mathtt{MA} \sin h$

Chapter 6

Building A Better Focus Model

6.1 Introduction

This chapter contains instructions for creating a focus model for the ROTSE-III telescope system. At least one night must be dedicated to taking focus data, and some follow-up focus data must be taken at different temperature ranges. Focus data is built up with many "focus runs", described in Section 4.3.3 and Table 4.1. Each focus run is a set of exposures taken on a field at regular focus position intervals.

Please see Chapter 2 for information on starting the system. Chapter 4 should also be consulted on how to make modifications to the the `astrod.conf` scheduling configuration file.

6.2 The Focus Model

6.2.1 The Focus System

The ROTSE-III focus system consists of a stepper-motor and a movable diaphragm which pushes the secondary mirror. The diaphragm has a travel range of around 1 mm. The stepper motor has an arbitrary zero-point that can be set with a screwdriver. The stepper motor puts pressure on the diaphragm, and shifts the focus. When the stepper motor is moved back the pressure is released, and the focus shifts back.

The most important thing to note about the focus system is the focus motor position in milimeters does not directly describe the position of the secondary mirror. Depending on the position of the zero-point, there can be several milimeters of travel (typically $\sim 2-3$ mm) before the motor applies pressure to the diaphragm. When pressure is applied, the focus changes. Around 1 mm from the initial pressure, the diaphragm cannot be stretched further, and the motor will stop. When the focus motor stops off-target, `schierd` will log an error and the system will shut down.

Once telescope alignment has been completed (if that is possible!) care should be taken to note both where the focus motor engages the diaphragm, and where the maximum focus position is located.

6.2.2 The Basics

After various focus tests, we determined that the telescope focus depends on the temperature and the elevation. There does not appear to be a focus dependence on azimuth, and hence all focus data is taken pointing south or north. The final "focus model" is the term used for the parameters for a bilinear fit of the best focus as a function of temperature and elevation.

During a focus run, (see Section 4.3.3 and Section 6.4), images are taken of a single field at regular focus intervals. With a typical focus run, images are taken at a fixed azimuth and elevation several times during the night.[1] It is important to make sure the configured focus motor range covers both sides of focus for the entire night. Also, any focus images taken while the motor has not yet engaged the diaphragm are not useful.

[1] Actually, the fixed azimuth and elevation refer to the field location at the *start* of the focus run. At the end of a ~5 minute focus run, the azimuth and elevation do change slightly.

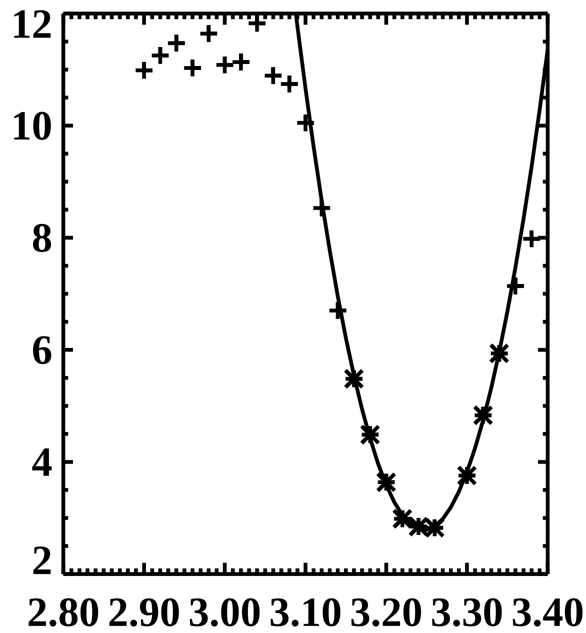

Figure 6.1: Plot of FWHM (pixels) vs. Focus Position (mm) for a sample focus run. The stars are the points used to fit the parabola.

The quality of the focus is determined by the median of the central subregion full-width-half-maximum (FWHM) measurements of the PSF made by SExtractor. For a given focus run, a parabola is fit to the plot of FWHM as a function of focus position, as in Figure 6.1. The best focus is determined to be the minimum of the parabolic curve. The error is determined to be the width at which the parabolic fit changes by 10% from the minumum value.

When enough focus values have been tabulated for a range of temperatures and elevations, we can begin a focus model fit. The model can be improved with more than one night of focus data to cover a wide temperature range.

The focus model is created with a least-squares fit to the best focus as a function of temperature, elevation, elevation squared, and the various cross terms. The idl program that performs this task is described in Section 6.5.

The focus model implemented as described in Section 3.8.1 is flexible, and can handle any polynomial combination of temperature, elevation, and azimuth, if such fits become desirable in the future.

6.3 Manually Monitoring the Focus Gradient

During testing, watching the focus gradient across the image is important. To this end I have written a utility, `focus_gifs.pro`, which generates a gif with the subimage in the four corners and the center, as well as a measurement of the FWHM in these regions. This section describes how to use this program. However, the exact plan to remove focus gradient has not yet been determined: is it camera tilt, secondary tilt, corrector tilt, or something else? A neverending work in progress...

1. Take a focus image(s) with `rush`, while the sexpacman pipeline is running.

   See the `rush` documentation in Section 2.5 for more information.

2. In the pipeline parent directory (/rotse/data/pipeline/), create a listfile of the images you wish to monitor.

   ```
   $ cd /rotse/data/pipeline/
   ```

   ```
   $ ls color=never image/filename.fit > file.list
   ```

3. Start up idl in the /rotse/data/pipeline/ directory, and run `focus_gifs.pro`.

   `$ idl`

   `IDL> focus_gifs,'file.list'`

4. Open a separate terminal, and go to /rotse/data/pipeline/image/ to view the gifs.

   `$ cd /rotse/data/pipeline/image/`

   `$ ee filename*.gif &`

   The Electric Eyes program (`ee`) has a useful slideshow option when multiple files are specified on the command line. You can now page through the images, and watch the various subregions as the telescope is brought through focus. If the various subregions go through best focus at significantly different focus positions, something is out of alignment.

6.4 Setting Up an Automated Focus Run

The first time an automated focus run is performed, an entire night's observing should be dedicated to the focus procedure. Starting early is important to get a large temperature range through the night. Follow-up focus runs should be performed when the nighttime temperature changes significantly, for greater coverage of the temperature domain.

Setting up a set of focus runs is easy in the scheduler:

1. Make sure a focus run item is defined in `astrod.conf`, as described in Section 4.4.1.

   There should be a trigger sequence of the following form:

   ```
   trigger          focus_run      foc      0      fr
   ```

2. Set the focus run parameters in `astrod.conf`, as described in Section 4.4.2.

   The configuration parameters `foc_lim` and `foc_step` determine the range and stepsize for the focus run. Be sure that `foc_lim` liberally covers both sides of focus (usually a range of 0.5 mm is necessary). A stepsize of 0.02 mm should be adequate.

3. Put the following schedule items in `astrod.conf`:

   ```
   sched focus_run "-t 4 -p 1 -u -17.0 -c 120 -l 85.0 -a 180.0 -e 2000.0 -M 45.0"
   sched focus_run "-t 4 -p 1 -u -17.0 -c 120 -l 65.0 -a 180.0 -e 2000.0 -M 45.0"
   sched focus_run "-t 4 -p 1 -u -17.0 -c 120 -l 45.0 -a 180.0 -e 2000.0 -M 45.0"
   sched focus_run "-t 4 -p 1 -u -17.0 -c 120 -l 35.0 -a 180.0 -e 2000.0 -M 45.0"
   sched focus_run "-t 4 -p 1 -u -17.0 -c 120 -l 25.0 -a 180.0 -e 2000.0 -M 45.0"
   ```

   By specifying azimuth (`-a`) and elevation (`-l`), we ensure that we fix the elevation (at the start of each run) through the night. Specific fields specified by RA/Dec would drift through the night, making the focus model calculation difficult.

   In this focus sequence, each elevation is run 4 times (`-t 4`) during the night at 2 hour intervals (`-c 120`). An elevation is skipped if the field is within 45° of the moon position. The sequence is started when the sun is 17° below the horizon.

6.5 Constructing a Focus Model

Once an automated focus run has been performed as in Section 6.4, a focus model (described in Section 6.2.2 can be constructed. This is accomplished with the idl program `find_focus3.pro`, which returns a structure with the best focus positions (for debugging purposes), as well as plotting the linear fits to the screen and printing to the terminal the constants for the focus model.

1. Go to the pipeline directory (or the directory that contains corresponding image/ and prod/ subdirectories with the appropriate image and header files) and create a list of the focus images.

   ```
   $ cd /rotse/data/pipeline/
   $ ls --color=never /rotse/data/pipeline/image/date_foc*.fit > focus.list
   ```

2. Start idl and run `find_focus3.pro`.

   ```
   $ idl
   IDL> find_focus3,'focus.list',best_focs
   MRDFITS: Binary table...
   ...
   ```

3. As it is processing, fits like that in Figure 6.1 are plotted. After each plot, the user is asked to confirm that the program has found a good fit. This helps reduce problems from cloudy data and other systematics.

4. When it completes, a residual plot like that in Figure 6.2 is made. The focus model constants (and configuration file abreviations) are also printed:

   ```
   ...
   Offset       ("1")  :        3.4426761
   Elev         ("e")  :    -0.0068352453
   Elev^2       ("ee") :    3.2436731e-05
   Temp         ("t")  :     0.0044962081
   Temp-Elev    ("et") :    5.8851819e-05
   Temp-Elev^2 ("eet"):   -2.4983650e-07
   ```

5. Put these constants into a focus model file as described in Section 3.8.1.

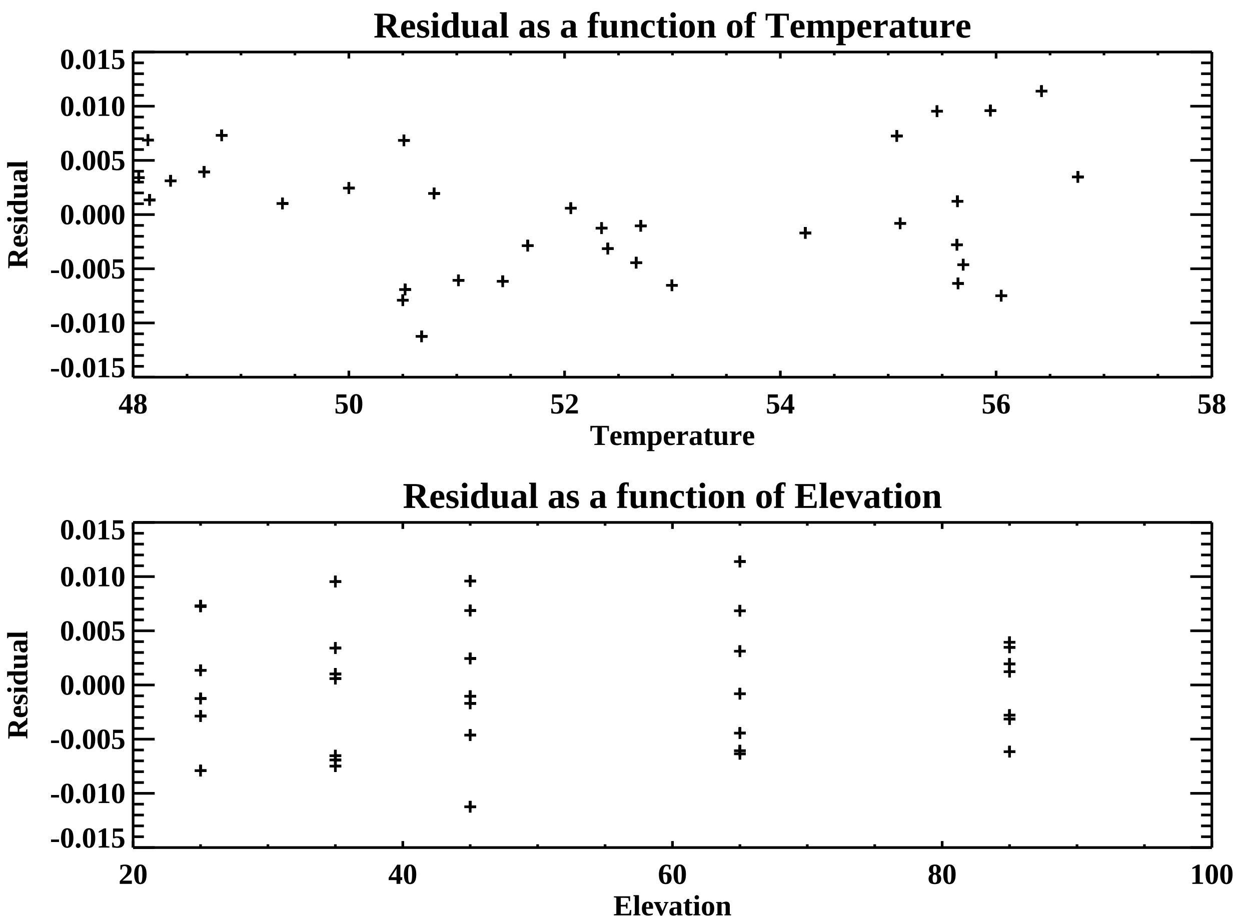

Figure 6.2: Plot of focus residuals as a function of temperature and elevation.

Chapter 7

Image Correction and Calibration Images

7.1 Introduction

Fast, on-line image correction and calibration is the key for the success of the ROTSE-III program. In this chapter I describe our current strategy for creating calibration frames (darks and flats), as well as using these calibration images quickly in an on-line pipeline. A full discussion of the online analysis pipeline, is in Chapter 8.

The ROTSE-III system runs with an unfiltered, thinned CCD. This, combined with the wide-field, creates difficulties in getting good quality flat-field images. Not only do we have to deal with the vignetting from the wide-field and the pixel-to-pixel quantum efficiency variations,[1] strong night-sky emission lines create an unpleasant interference fringe pattern in the images. The final plan for obtaining flat-field images will most likely change from the process currently described in this chapter.

For use in the automated pipeline, calibration files are put in the directory:
`/rotse/data/pipeline/cal/`

7.2 Dark Images

7.2.1 Obtaining Dark Images

Dark images are a measure of the dark current in the CCD at a specific temperature for a specific exposure length. Due to the bias level and read noise, it is difficult to scale one dark exposure to another exposure length. Dark images are therefore taken at each of the configured exposure lengths described in Section 4.3.2. Dark images are subtracted from image frames to remove the bias level, the dark level, and to subtract hot pixels to improve photometry. We also plan to use dark frames to create a hot pixel map to flag false identifications from hot pixels.

Dark images are obtained by the automated scheduler `astrod`. The `ndarks` configuration parameter (Section 4.3.3) describes the number of darks taken at each exposure length. The default is 6. In addition, a dark run must be scheduled, as described in Section 4.4.2. One dark run per night should be fine for normal operation.

7.2.2 Creating a Median Dark

Once a set of dark images are obtained, the next task is to create a median dark image from the frames. This is accomplished by the perl script `makedark`. Running `makedark` is easy:

```
$ cd directory_with_darks                    ; Usually /rotse/data/3a1/
```

[1] We do not believe that intra-pixel variations of qe should hurt us. [why?]

```
$ makedark
        This script will median all the darks of similar
        type located in the directory given as the argument.  A
        specific type can be specified if desired.

        Usage: makedark -d darkdir [-t datestr]
$ makedark -d .
```

`makedark` thens scan through the directory specified (usually the current directory), groups the dark frames by temperature and exposure length, takes the median of the dark images to create a dark with the filename of the form: `YYMMDD_drk0200_3a.fit`. (See Appendix A) The number after "drk" is the exposure time multiplied by 10.

It is worth the time to take a look at the median darks with `ds9` to check for any light leak problems, read noise problems, or other systematics. Only good darks should be used in the automated pipeline.

7.3 Flat Fields

7.3.1 Twilight Flats

Currently, our best flat-field images are generated from twilight images. To create a twilight flat, we image high elevation fields at twilight, away from the setting sun. We then take the median of these images to remove the stars. Because the light in the twilit sky is dominated by scattered broadband sunlight, fringing from strong night-sky lines is not evident.

A sample twilight flat is shown in Figure 7.1. Along with the vignetting (on the order of ~10%), other features can be seen that appear to be real.

Scheduling a Twilight Flat

Creation of a twilight flat requires a clear, moonless twilight period. Here I describe the scheduling parameters necessary to create a twilight flat. For more details on using the scheduler, see Chapter 4.

1. Check that a twilight flat trigger is defined in `astrod.conf`, as described in Section 4.4.1.

   ```
   trigger    twi_flat         twi      0          1jm
   ```

2. In the schedule list, add the following lines:

   ```
   sched twi_flat "-t 5 -u -10 -a 90.0 -l 70.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 88.0 -l 70.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 86.0 -l 70.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 92.0 -l 70.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 94.0 -l 70.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 90.0 -l 72.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 90.0 -l 68.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 92.0 -l 68.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 94.0 -l 72.0 -e 2000.0 -i 5"
   sched twi_flat "-t 5 -u -10 -a 88.0 -l 72.0 -e 2000.0 -i 5"
   ```

   The first several images will probably be saturated. This is not a problem, as there are plenty to go around. We will need at least 30 to create a decent twilight flat. Each field is imaged once with a medium length (20 s) exposure, and the telescope returns to the same azimuth and elevation after at least five minutes. This allows the stars to drift so we do not image the same stars in the same pixel from image to image. I'm not sure if the azimuth should be changed from $\sim 90°$ to $\sim 270°$ for the southern hemisphere.

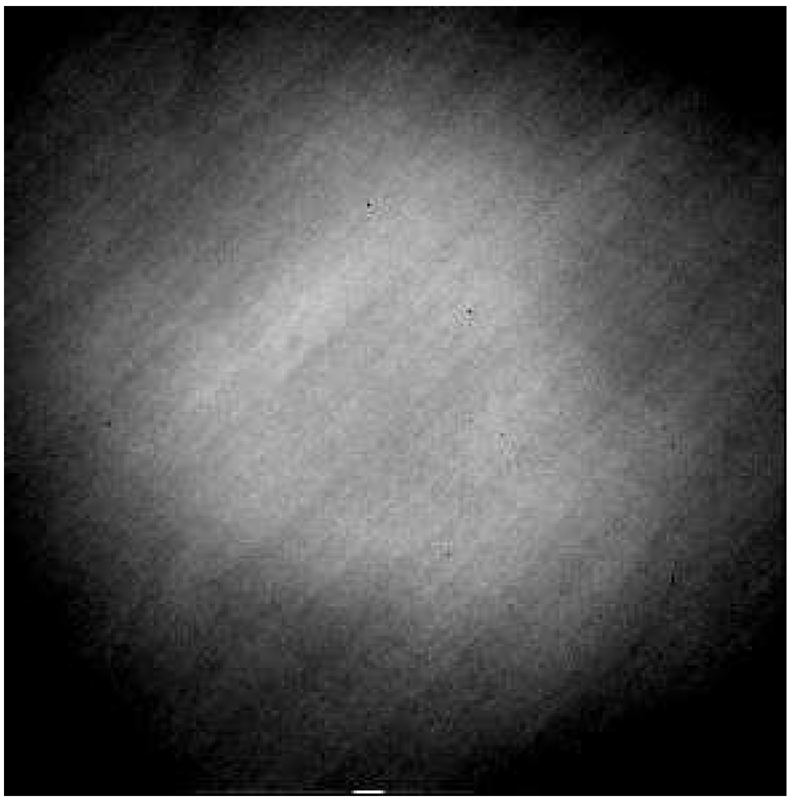

Figure 7.1: A sample twilight flat. The diagonal lines appear to be real variations in the flat.

3. On a clear, moonless twilight, run the system.

   It is important that these images are taken on schedule. If they are taken late, the night sky lines will dominate and you will end up with a sky flat (see Figure 7.2) instead of a twilight flat.

Making a Twilight Flat

There is a handy perl script called `makeflat3` that creates a median flat from a list of images. The program automatically normalizes each image from its central subframe to ensure that changing background light levels (due to the darkening of the sky at twilight) do not systematically bias the flat. `makeflat3` requires that a median dark image from that day's output is in the current working directory. If a current dark is not present, we can create a pointer to an archived dark. Following the directions here:

NOTE: The instructions for this section need to be checked when we make another flat...I'm not sure which version of makeflat is the one that works!

1. Ensure that a present day 20 s dark is available. If it is *not* available, either make one as described in Section 7.2 or create a link to an archived dark.

   ```
   $ ls /rotse/data/pipeline/cal/             ; Is today's dark there?
   ...
   $ cd /rotse/data/pipeline/cal/
   $ ln -s today_drk0200_3a.fit old_drk0200_3a.fit
   $
   ```

2. Check the `makeflat3` syntax.

   ```
   $ makeflat3
          Usage:  makeflat -d flatdir [-f framenum -t datestr -r darkdir -n namestr -c]
   ```

3. Run `makeflat3` on the data.

   ```
   $ makeflat3 -d /rotse/data/pipeline/image/ -t today -r /rotse/data/pipeline/cal/ -n twi
   ...
   ```

   This should take a few minutes. Please note the specification of the `twi` TLA, which is the standard for twilight flats. This should be substituted when running other flats.

7.3.2 Sky Flats

A sky-flat image is generated from all-sky imaging data, typically in conjunction with the building of a pointing model as in Section 5.4. A typical sky-flat, as shown in Figure 7.2, is a convolution of the "real" twilight flat with a fringing pattern, described in Section 7.3.3.

Making a Sky Flat

The procedure to create a sky flat is similar to that mentioned above in Section 7.3.1:

1. Run an all-sky imaging sequence. A pointing model sequence as described in Section 5.4 works well, if taken when the moon is set.

2. Ensure that a present day 20 s dark is available. (This is the standard exposure length for the `tpt` pointing data).

3. Check the `makeflat3` syntax. Here we will use it in a slightly different way.

   ```
   $ makeflat3
          Usage:  makeflat -d flatdir [-f framenum -t datestr -r darkdir -n namestr -c]
   ```

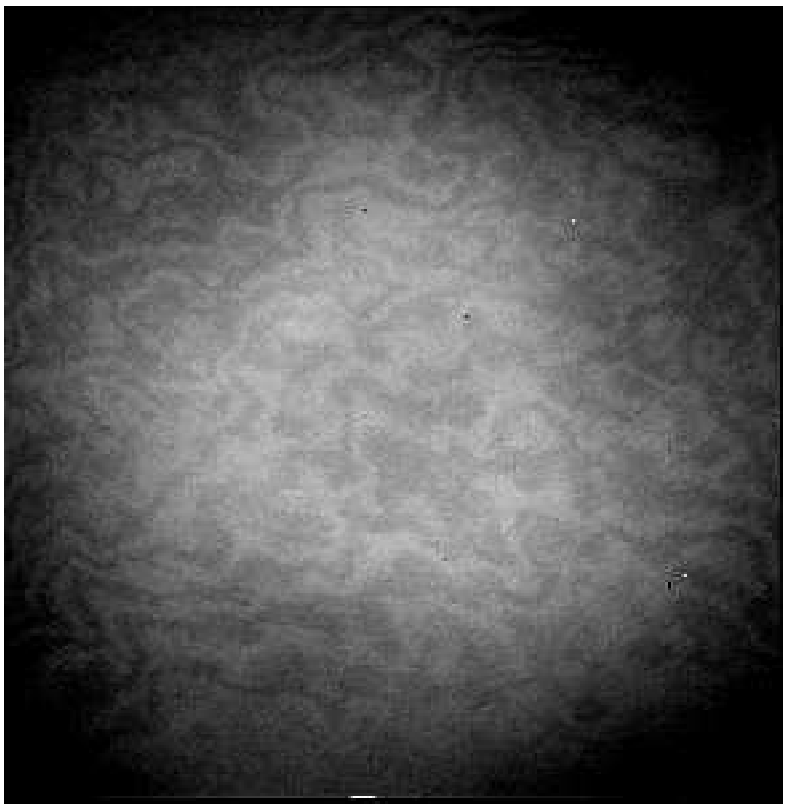

Figure 7.2: A sample sky flat. The fringe pattern from the night sky lines is clearly evident.

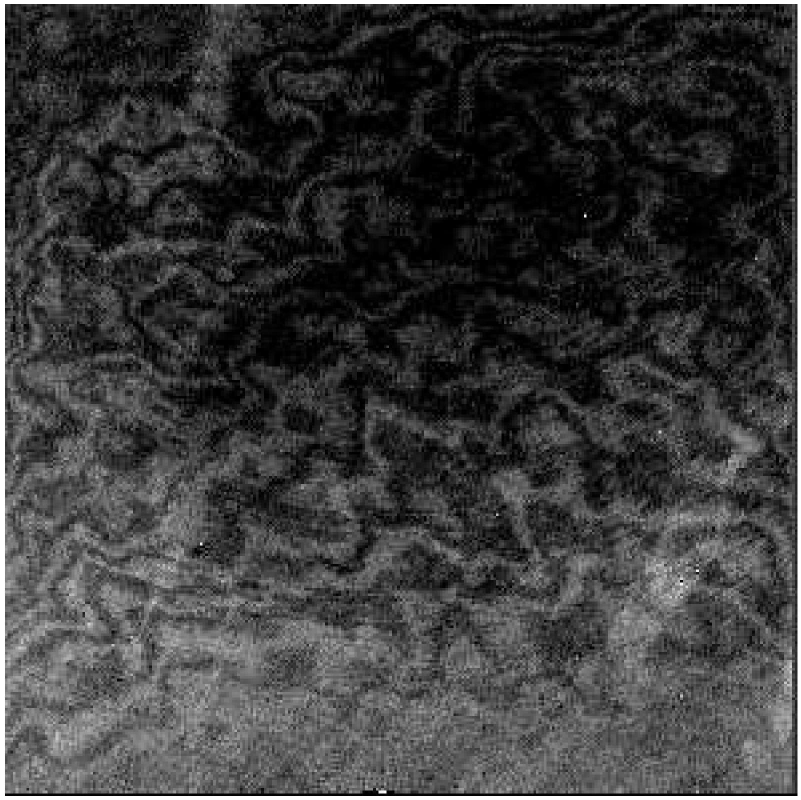

Figure 7.3: A sample fringe map. The image has been arbitrarily normalized. This pattern is consistent from image to image, although the normalization changes.

4. Run `makeflat3` on the data.

   `$ makeflat3 -d /rotse/data/pipeline/image/ -t today -r /rotse/data/pipeline/cal/ -f 001 -n tpt`

   We must specify that we only wish to use the first frame (001) of each pair. We do not want to use pairs of images, as the stars land on the same pixels in image pairs, thus biasing the median.

7.3.3 Fringe Maps

One of the unfortunate aspects of a thinned CCD is the problem of interference fringes. When light of certain wavelengths hit the CCD, some of the light is reflected and interferes with itself. The scale of the fringe pattern is wavelength dependent, and like the rainbow on an oilslick, you get a fringe pattern like that in Figure 7.3.

The night sky is the greatest source of this fringe pattern. Because most of the night sky brightness is contained in certain emission lines, some of these lines create a strong fringe pattern. Light from the sun, moon, and stars are broadband, and so will not show the fringe pattern. However, since we are running an unfiltered system, the fringing subtly effects the photometry of the stars in unpredictable ways.

One cannot leave the fringe pattern in the sky flat, as in Figure 7.2, as this would amplify the effect of the fringing on the photometry. At the moment, we have decided to find the fringe scaling factor and subtract the fringe pattern. This helps, but is not perfect. One solution (to many problems!) would be running with an R-band filter, but this would greatly decrease our sensitivity to faint sources.

Making a Fringe Map

A simple program to make a fringe map has not yet been written. I promise I will do this when the telescope is back. It is actually quite easy, you just take the sky-flat and subtract the twilight flat, set the median to 0,

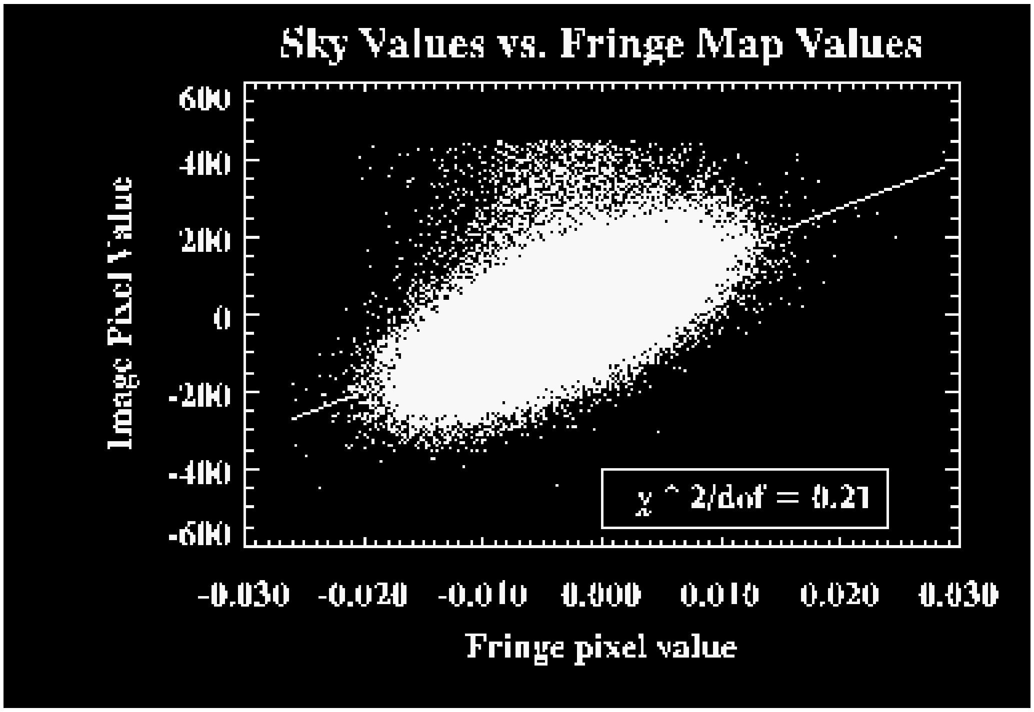

Figure 7.4: A plot of typical image values vs. fringe map values. The higher image values typically correspond to higher fringe map values. The slope of the line gives the fringe scaling factor, and the fringe map is then subtracted from the image.

and normalize it. You end up with something like Figure 7.3.

7.3.4 Fitting the Fringe Map

To remove fringing from the images, the fringe pattern must be scaled and subtracted from the image. The level of fringing depends on the sky brightness, lunar illumination, and cloud cover, and an analytical calculation for the fringe level is not possible. At the moment we have implemented a simple linear fit to find the fringe scale.

The image pixel value is plotted against the fringe map pixel value for the central subregion of the image, as in Figure 7.4. Some of the image pixels contain starlight, so the image values are cropped at the 3σ level, creating the sharp cutoff at the top of Figure 7.4. Under ideal circumstances, the brighter sky values correspond to the higher fringe map values, and a simple linear fit determines the fringe scale. If there is a strong sky gradient across the image, as when the moon is bright and nearby, this can produce a false fit. At the same time, the bright broadband moonlight tends to wash out the fringing pattern. Our fringe subtraction technique might need to be improved to achieve our desired photometric accuracy for faint stars.

7.4 Updating the Automated Pipeline Files

How often should the calibration frames used in the automated pipeline be updated? Good question. Our current plan has new calibration frames to be obtained on a monthly basis, or after any telescope alignment change. When we have a good schedule plan in place, we will describe it right here.

7.5 Performing Image Correction: `corr_im_fast` and `new_dfc`

7.5.1 Introduction

The old ROTSE-I image correction was handled by a perl script called `corr_image` that was very slow. The guts of `corr_image` have been replaced by a C program called `new_dfc` (From New Dark-Flat-Crop) which uses the `cfitsio` library for faster fitsfile processing. The new `corr_im_fast` perl script is a wrapper for this program.

7.5.2 What `corr_im_fast` Does

This is a short description of the steps that `corr_im_fast` takes. Further details can be gleaned from the perl script itself.

1. `corr_im_fast` is provided with directories for image and calibration files. For more information on the command line, see Section 8.2.1.
2. It scans the calibration directory for the best dark image. This image must have the same exposure time as the data image, the date must be on or before the data image, and the camera temperature must be within 2° of the data image.

   If a proper dark is not found, the program will exit with an error.
3. It scans the calibration directory for the best flat image. This image must have been taken with data on or before the date of the data image.

   If a proper flat is not found, the program will exit with an error.
4. It checks which amplifier (or both) was used with the image to determine the proper image cropping parameters. The cropping removes the bias areas and leaves only image data.
5. It runs new_dfc.

7.5.3 What `new_dfc` Does

This is a short description of the steps that `new_dfc` takes.

1. `new_dfc` must be provided with the image, output file, flat, dark, fringe map, and thumbnail file names.
2. The program first subtracts the dark frame from the image.
3. It next divides the image by the flat-field image to remove vignetting and account for interpixel sensitivity variations.
4. It next scales the fringe map as described in Section 7.3.4, and subtracts the scaled fringe map.

   If the fringe map fit is considered to be "bad," (the χ^2/d.o.f. > 2), then the fringe map is not subtracted and this is noted in the FITS header. (See Section C.2.)
5. It next crops the image as defined on the command line. The cropping algorithm is flexible enough to handle a central bias region that results from dual-amplifier readout.
6. The program now "compresses" the image with the FITS standard b-zero and b-scale parameters, to change the image from a 32-bit floating point image to a compressed 16-bit integer image, as described in Appendix B.1.
7. A thumbnail jpeg image is created for easy display on the web page. The typical thumbnail is 8 k.
8. The program exits and returns to `corr_im_fast`.

All the operations performed to correct an image are recorded in the FITS header. It will soon be possible to reverse the correction process, however, with a loss of bias region information.

For more details on running the correction software, see Chapter 8.

7.6 Uncorrecting Images: `uncorrect`

At the current moment, we plan to archive only the corrected data and the calibration files. The image correction process is completely reversible, with the exception that the bias information is lost. The bias subframe statistics are retained in the image header, described in Section C.1.

The program `uncorrect` performs the uncorrection procedure. It is run as follows:

```
$ uncorrect
usage:  uncorrect -i filename [-o outfile] [-C caldir] [-c subr_x -d subr_y -m xmask]
```

`uncorrect` will only use the dark, flat, and fringe files that are explicitly named in the corrected image header. The path to these files defaults to the paths specified in the header, or can be overridden with the `-C` option. The output file defaults to the image root with a `_raw.fit` extension, so as to distinguish it from the true uncorrected file. The subregion cropped from the original defaults to that specified in the header [need to write this!] or the default cropping from `corr_im_fast`. This program is fairly fast, but if the online analysis works as planned we should only need to run this program infrequently.

Chapter 8

Realtime Data Analysis and Monitoring Telescope Operations

8.1 Introduction

Once the telescope is configured, scheduled, aligned, and focused, normal operations can commence. ROTSE-III is designed to be an automated, robotic telescope; most nights, it should take hundreds of images without necessary human intervention. Once the images are recorded, there is a suite of analysis programs to extract useful information from these images. Ultimately, we hope to make a full closed-loop automated analysis program, in which the automatic software processes the results gleaned from images to instruct the daq in the scheduling of future images, as well as to distribute discovery alerts, but for the moment, only some of this process is fully automated. The entire analysis pipeline sequence is diagrammed in Figure 8.1. In this chapter, we describe the automated analysis code and resulting data products, a few steps that must be performed manually after the automated steps are complete, and the methods we have established to monitor system behavior during operation.

We have written several sets of programs to analyze ROTSE-III data as it is recorded to disc. These programs in concert provide calibrated object lists for observed fields. After a GRB alert has been received, sets of object lists for multiple observations of the target field are automatically compiled into "match structures". These match structures enable us to track the change in intensity of all objects in the field. They can be easily scanned for highly variable objects, as a GRB is expected to be, and they are a useful tool for weeding out spurious artifacts that only appear in a single image. Although match structures are produced in response to a GRB alert, the primary output of the automated realtime processing is the calibrated object list, so we describe in Section 8.2 how these lists are created, and defer all discussion of match structures to Section 8.3, when the manual manipulation of data products is described.

8.2 Realtime Automated Analysis

There are two programs that act in tandem to produce calibrated object lists, and they are referred to as "pacman" programs, because they "eat" one file at a time as they are produced during the analysis process, in analogy to the arcade game figure that eats a row of little pellets.

The process begins when `camserverd` records a new image to disc and creates a link to that image file in the specified link directory (Section 1.2.4). A Perl script called `sexpacman.pl` monitors that directory for new links, and when one appears, it corrects the image with the available dark- and flat-fields, and processes the result through `SExtractor` to produce a list of objects in the field. This list is called an `sobj` file. `sexpacman.pl` then adds the name of the `sobj` file to a list of other `sobj` files, sorted according to priority (most images are processed chronologically, but GRB follow-up images outrank other images, and prompt burst images outrank late-time follow-up images. See the explanation for the scheduler in Section 4.3.4 for more information on these triggers.).

The list of sobj files is monitored by an idl process called idlpacman.pro, hereafter referred to simply as idlpacman. The first file in the list is read by idlpacman and calibrated against the USNO A2.0 catalog to produce a list of estimated R-band magnitudes and celestial locations for these sources. These lists are called cobj files. The name of the processed sobj file is removed from the list, and idlpacman moves to the next file name on the list, or waits for a name to be added, if no more are present. The time elapsed to the creation of a calibrated object list is typically 45 s from the first appearence of the raw image.

8.2.1 sexpacman

The Perl script sexpacman.pl takes one argument on the command line: the configuration file name (usually, but not necessarily, called sex.conf). The default command has been aliased to startsexpac, which also writes any output to a file sexpac.log for later inspection (See also Section 2.3.3). This script will run in an infinite loop, and the configuration file consists of a list of directories where it should look to find the particular files it needs. Here is an example of what such a file should look like:

```
# This is a configuration file for sexpacman.pl
#
linkdir        /rotse/data/3a1/links         # directory for image links
listfile       /rotse/data/pipeline/sobjlist # file for the output list
sobjdir        /rotse/data/pipeline/prod     # directory for sobj files
cimdir         /rotse/data/pipeline/image    # directory for _c files
datdir         /rotse/data/pipeline/cal      # directory for raw image files
caldir         /rotse/data/pipeline/cal      # location of darks and flats
thumbdir       /rotse/data/pipeline/thumbs   # directory to put thumbnail JPGs
```

In essense, sexpacman.pl is a wrapper program for corr_im_fast (Section 7.5) and SExtractor. It periodically scans the linkdir (defined in the configuration file) for new links to ROTSE-III images, created by camserverd (See Section 3.6). First, sexpacman.pl calls corr_im_fast to perform the dark- and flat-field corrections. Once an _c file has been created, sexpacman.pl calls SExtractor to process the corrected image. The resulting output yields three files: a list of objects (called an sobj file), an image which contains a measurement of the sky brightness of the image in a 64×64 grid (called a sky file), and a file (usually called sobjlist) which is a list of the sobj files. This list file is monitored by idlpacman (Section 8.2.2) in order to calibrate the source lists against the USNO A2.0 catalog.

The process also creats a small, "thumbnail" version of each image in JPEG format which is copied via a cron job to the web server at UofM, and can be viewed in an archive on the web page at www.rotse.net. The most recent thumbnail is also displayed on the front page at that web site, and can be viewed through pressing a button on the real-time status monitor web page (Section 8.4.2). An example thumbnail is shown in Figure 8.2.

8.2.2 idlpacman

We have a license for one run-time idl process per telescope control computer, so we have constructed an idl data analysis program that can run with no command-line input. This program takes the source lists that are output by SExtractor (sobj files) and calibrate them against the USNO A2.0 catalogue, producing calibrated object lists. These lists, along with the corrected image headers and the _sky files, are saved into FITS files called cobj files. The idlpacman function is activated through the following command:

```
prompt%> echo idlpacman | idl
```

This command has been defined into an alias called startidlpac (See also Section 2.3.3), which also pipes the output into a log file called idlpac.log, which can be inspected to monitor the program's activity. idlpacman looks for a configuration file in the current working directory. (See Section A.6 for the standard directories for realtime operations.) This file *must* be called idlpac.conf. This file contains the names of the files and

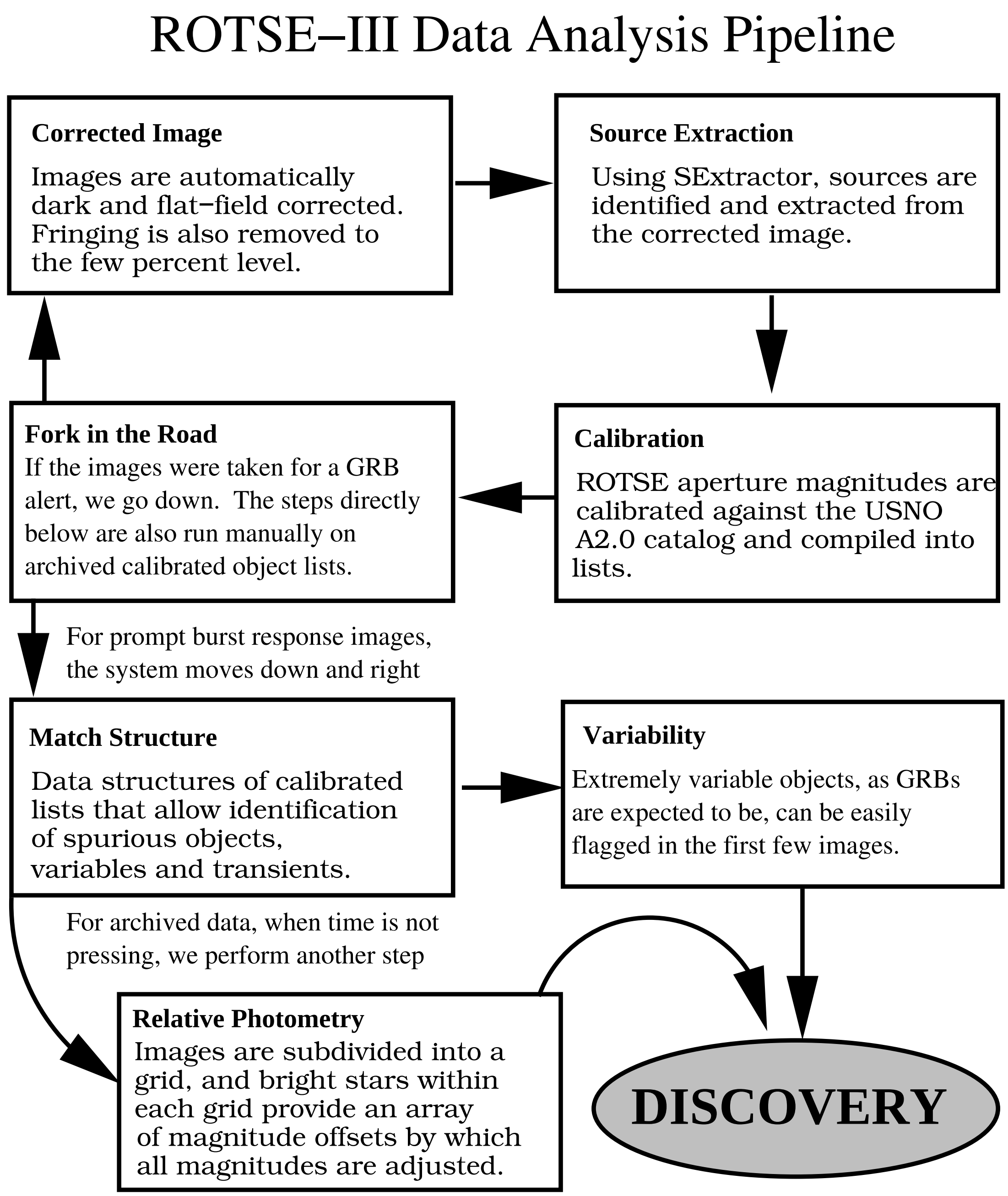

Figure 8.1: Flowchart depicting the ROTSE-III data analysis pipeline. Most of these steps have been implemented in an automated realtime process. Based on the file names, the software decides how far to pursue the analysis; prompt burst response data will be compiled into match structures and scanned for highly variable, uncataloged objects. Object lists from other kinds of triggers are created and stored for future processing. The relative photometry process is only intended to be run manually on these archived data.

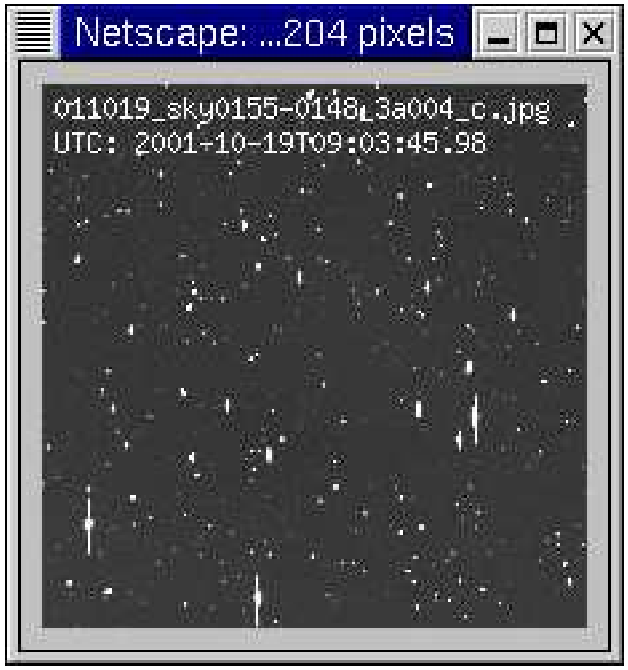

Figure 8.2: Thumbnail image produced by the ROTSE-IIIa DAQ system during the systems operation testing in October of 2001. The filename as well as the date are displayed on the image.

directories in which `idlpacman` will find its input and write its output. Here is an example of a potential `idlpac.conf` file (the format does not support comments in the file):

```
sobjlist sobjlist
workdir /rotse/data/pipeline
sobjdir /rotse/data/pipeline/prod
cobjdir /rotse/data/pipeline/prod
imgdir /rotse/data/pipeline/image
statdir /rotse/data/pipeline/prod
statroot rotse3a
```

The parameters do not have to be in this order, but they must all be present. `sobjlist` is the output file from `sexpacman.pl` that contains a list of the `sobj` files to be processed by `idlpacman`. `workdir` is the working directory for the process (might be, but not necessarily, the directory from which is was called, which is where `idlpac.conf` should be). `sobjdir` is the directory whence the `sobj` files are read, and `cobjdir` is the directory where the `cobj` files are written. `imgdir` is where the original corrected images are stored.

The last two elements enable us to monitor the analysis status and image quality in near real time. Within the `statdir`, a file called "`[statroot]_run.dat`" is created that keeps a running log of various diagnostic quanities: the image name, the time, the temperature, the difference between the catalog and ROTSE-III positions of stars, the FWHM of the PSF, the elevation of the pointing direction, and the limiting magnitude of the image. Four of these quantities are also graphed in a GIF image named `status_mon.gif`, which is regularly transferred to Michigan for display over the WWW (Section 8.4.2). Such an image is displayed in Figure 8.3. The log file is also used by the mount in its homing operation, see Section 3.8.

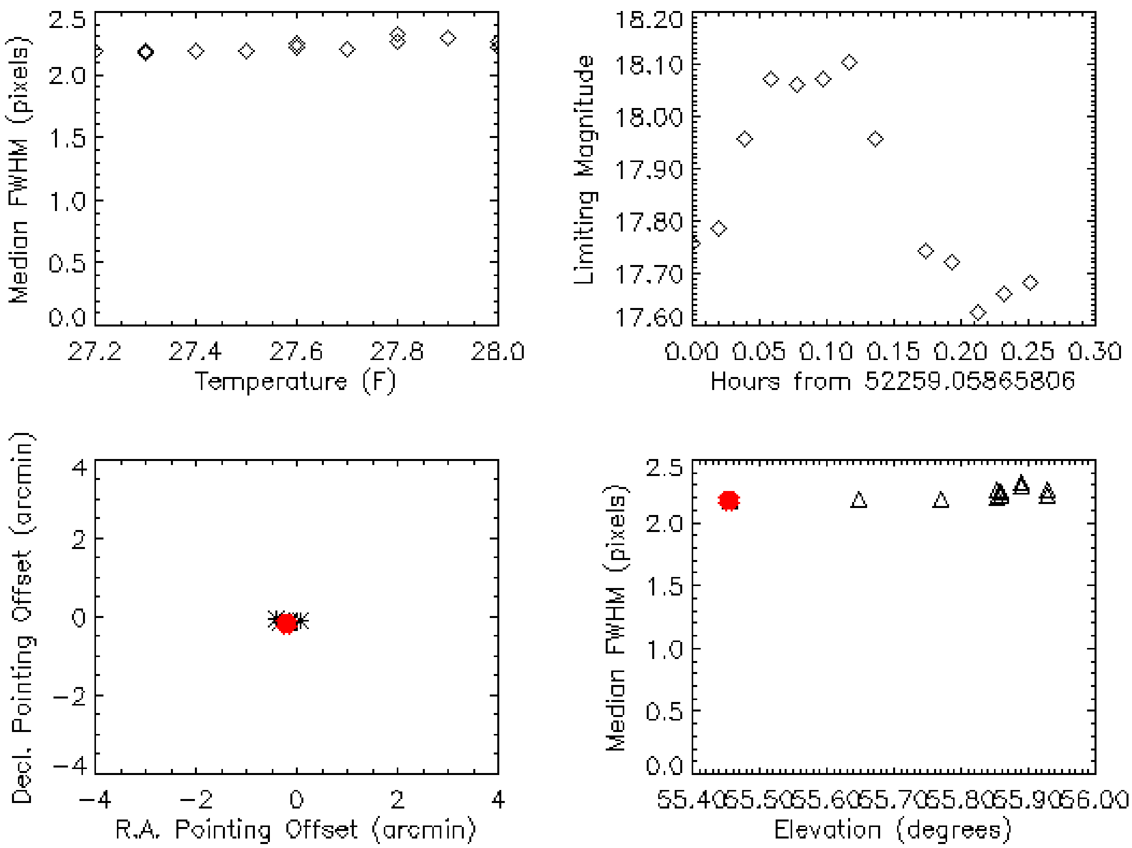

Figure 8.3: Four status variables are displayed graphically for near-real-time viewing over the WWW. Clockwise from top left: the median FWHM as a function of temperature, the image limiting magnitude as a function of time, the median FWHM as a function of elevation, and the pointing offset between mount and true coordinates. In the bottom figures, the most recent data point is plotted in red. These graphs help us determine when problems arise during the night, as might be evidenced by a drift in the pointing accuracy or a dramatic change in the limiting magnitude (which usually indicates clouds).

8.2.3 Burst Response Analysis in Real-time

If `idlpacman` recognizes the image under analysis as an image taken in response to a realtime burst alert (from the three letter acronym in the file name – see Appendix A), it performs additional operations to those outlined in the previous section. This procedure is still evolving, but currently involves compiling match structures, writing short binary files with information about sources of interest near the burst location, and cropped JPEG images of the region. These small files are updated after every ten images, and copied back to `www.rotse.net/burst_response/` for remote perusal by interested users.

8.3 Manual Data Analysis

8.3.1 Creating Object Lists Manually

Should you wish to run `sexpacman.pl` manually, you can set the configuration file to point to directories that contain archival data, and set up a number of symbolic links to the image files you wish to analyze. `sexpacman.pl` is smart enough to skip the field correction if the link contains an `_c` element in its name. When all the links have been processed, you can kill the `sexpacman.pl` process. Alternately, you can run `corr_image` and `SExtractor` on each image file yourself. To do so, you will need to have the EXTRACT_PAR and EXTRACT_CONFIG environment variables defined to point to the directories where the `rotse3.sex` and `rotse.par` files are located. These files define the paramaters for `SExtractor`'s analysis and the output format, respectively.

The function calls look like this:

```
corr_im_fast -d [linkdir] -i [image name] -r -f -b -c [caldir] -t [cimdir] -C -F fringe.fit
   -T -H [thumbdir]
```

```
sex [file] -c [EXTRACT_PAR]/rotse3.sex -PARAMETERS_NAME [EXTRACT_PAR]/rotse.par
   -FILTER_NAME [EXTRACT_CONFIG]/gauss_2.0_5x5.conv -CATALOG_NAME [sobj_file_name]
   -CHECKIMAGE_NAME [sky_file_name]
```

Similarly, if you create a list of `sobj` files and the appropriate `idlpac.conf` file, you can run `idlpacman` on any set of archival data to create `cobj` files as well as the associated log file of diagnostic parameters and the `status_mon.gif` image.

8.3.2 Match Structures

Once more than one calibrated list for a given field is available, these lists are compiled into a "match structure"; a data structure that enables us to monitor the intensity variations of celestial objects, as well as filter out candidate objects that do not appear at the same location in sequential observations. Although match structures are automatically produced for sets of images taken in response to GRB alerts received over the GCN, they are primarily constructed manually, often combining many days' of sky patrol data to create light curves for the sources within the wide-field survey region. In this section, we summarize the properties of match structures and describe how to create them.

A match structure identifies objects by matching their celestial coordinates from observation to observation. The telescope aspect is shifted through a small, random vector between observations. This ensures that hot CCD pixels and cosmic ray events are unlikely to be mistaken for celestial objects, as their CCD locations will change relative to the sky. One can then apply a relative photometry algorithm (Section 8.3.3) to stabilize the magnitude estimates and calibrate the systematic errors. This reduces the scatter in light curves for stable bright objects to <1%, and allows us to reliably identify variable sources. We can also flag known distractions such as asteroids (that move from night to night) and "masked" stars (stars that appear in ROTSE images but are too close to bright stars for sensitive cataloged surveys to resolve).

Match structures are saved in FITS files that consist of two extensions: the first extension contains the match structure itself; a structure of arrays that hold astrometric, photometic and ancillary data about the sources within the field of view (for the gory details of the contents of match structures, see Section C.5). The second extension contains a array of structures that carry all the information from the header of each `cobj` file, preserving information about the original image from which it was produced, as well as which processes were run in order to create it (Section C.4).

A match structure consists of a set of arrays that contain information on each object per observation (like magnitude), as well as some mean properties of each object over all observations (like celestial location), and a few quantities that apply to each observation (like limiting magnitude). These arrays are defined in Table C.3. As each new calibrated object list is added to the match structure, each array is expanded to include the new data. Object magnitudes are copied from the `cobj` file. A −1 is assigned as the magnitude if an object already in the match structure is not detected in the new list. If a known object is outside the field of the new image, a −2 is assigned. As new objects are added to an existing list, their magnitudes in prior observations are also assigned values of −1 or −2, as appropriate. Mean and RMS quantities are recalculated as each observation is added.

The primary program to construct a match structure is `regmatch3_list.pro`. There is another program, `update_match.pro`, which is intended to process large numbers of fields into sets of match structures. The main workhorse, `regmatch3_list.pro`, is a flexible program that is called with this sequence:

```
IDL> regmatch3_list, match, stat, list=list, namelist=namelist, pair=pair, limits=limits,
   save=save, over=over, append=append, archive=archive, error=error, template=template
```

The final match structure is stored in `match`, and an array of structures that convey the header variables for the included observations is stored in `stat`. Either `list` or `namelist` *must* be set. The former is a single string: the name of a file that contains the list of the `cobj` files to be included. The latter is an array of strings in which each element is the name of a `cobj` file.

The other variables and keywords are optional. `/pair` will activate pair-matching: that only objects that appear in sequential pairs of images are included[1]. If this option is activated, the program expects an even number of `cobj` files. If an odd number is input, the last will be ignored and an error will be logged. If `/append` is activated, the program will assume that `match` is not empty to begin with, and the `cobj` files in the list will be appended to what is already in `match`. `limits` is a four-element array that gives the coordinates of a limited region ([R.A. min, R.A. max, Decl. min, Decl. max]), if the whole field is not desired. Sources outside these limits will be discarded. `template` is an array of coordinates for sources of interest. Sources whose celestial locations in the `cobj` files do not match any coordinates in `template` will be discarded.

The other keywords instruct the program what to do with the match structure when it is complete. Files are named according to the naming convention outlined in Section A.5. `save` will save the contents of `match` and `stat` into a FITS file in the working directory, but not if a file of that name is present already. `over` will overwrite any currently present file with the same name. `archive` is the same as `save`, only instead of the working directory, the standardized archival directory is used (see Section A.6). Saved match structures are FITS files with two extensions. The first extension contains the match structure itself, while the second contains the stats structure.

If you have a large number of images from different fields that you would like to compile into match structures, `update_match.pro` may be more useful. It is called as follows:

```
IDL> update_match, all, dir=dir, pair=pair, archive=archive, startroot=startroot, sky=sky,
   init=init, relphot=relphot, matchdir=matchdir
```

The first variable, `all`, is a required output variable. When the program exits, the file names (minus the first and fourth name extension, see Appendix A) will be stored as a string array in this variable. If `startroot` is defined (it should be an array of strings), only files whose root names match the elements of this array will be considered. If `/sky` is set, only Sky Patrol images will be processed. Otherwise, *all* images in any subdirectories of the working directory will be processed. If `dir` is defined, the working directory is changed to `dir` before beginning. The final match structures are saved as defined above in the working directory, unless `/archive` is set, in which case they will be saved in the standardized archival directory (if you don't want them to go in either the working directory *or* the archival directory, you can define an output directory with `matchdir`). If `/init` is set, the program will start from scratch, otherwise it will look to see if match structures already exist, and will only append new files to them if they are not already included. If `/relphot` is set, the program will also apply the relative photometry procedure, as described in Section 8.3.3.

[1]Not *any* sequential order is acceptable in pair-matching. Images must be odd-even sequentially indexed.

8.3.3 Relative Photometry

A match structure as described above is not yet finished. To minimize systematic errors in source intensities, it is helpful to perform a "realtive photometry" procedure to remove photometric mismeasurements due to such difficulties as the presence of thin haze over a portion of a frame, poor fringe subtraction (Section 7.3.3), etc. The `idl` program `relphot3.pro` will perform this procedure. For each object in each observation, `relphot3.pro` determines a small relative photometric correction, or in more extreme situations calculates a systematic error and sets flags to specify the problem and allow one to exclude the affected regions.

The program begins by selecting a set of good template sources in each matched list. To qualify as a template source, an object must be detected in more than 75% of the epochs at which the source location was imaged. Next, the median magnitude of all good observations of the template objects is calculated. Each image is then divided into subtiles, 200 pixels on a side, and a photometric offset are calculated for each template source, grouped by subtile, that has a statistical uncertainty in its magnitude measurement less than 0.1 mag. The number of template sources within a typical subtile is ~ 50. The relative photometry map is then calculated to be the array of median offsets for each subtile. The RMS deviation of the offsets in each subtile, which for good subtiles is typically around 0.03 mag, is used to estimate the systematic error. In addition, a subtile is classified as "bad" if there are less than five template sources within it, or if the RMS deviation of the template offsets is greater than 0.1 mag. Each magnitude measurement of each object in each observation is then corrected by the offset map using a bilinear interpolation, and the new systematic error estimate is added in quadrature to the previously estimated systematic error for each object in the observation. If the subtile is flagged as bad, then the object's observations are likewise flagged (flags are defined in Table C.2).

The function call for this procedure is:

```
IDL> relphot3, match, newmatch, rpmap, stat=stat, save=save, archive=archive, over=over,
init=init
```

The variable `match` contains the input match structure (or it may also be the file name of a FITS file containing a match structure), while `newmatch` is where `idl` will put the corrected match structure. If the `match` variable is a match structure, then the `stats` variable *must* be set to the array of `cobj` header structures (Section 8.3.2) that was created with that match structure. If the `match` variable is a file name, the array of structures in the second extension will be read into the `stat` variable. The variable `rpmap` is a structure that contains the map of offset values as well as several diagnostic parameters. This structure is defined in Section C.6, and it is saved as the third extension to the `relmat` FITS file (Section A.4).

The application of the `save`, `archive`, and `over` keywords are the same as with `regmatch3_list.pro`, above. If `init` is set, then the program will blindly apply the procedure. Otherwise, it will check to make sure the procedure has not already been applied to `match`, and exit with a warning if it seems that relative photometry is not necessary in this case.

8.4 Operational Status and Monitoring

The procedure for configuring, starting, and running a ROTSE-III telescope are described in Chapters 2, 3, and 4. Once you have a evening's session underway, there are several ways to stay in touch with what the system is doing. If one has the access capability one can, of course, log in as observer and check the .log files that are produced by the daemons `rotsed` and `astrod` (Sections 1.2.1 and 1.2.8), or look at the data files being produced. However, there are other methods to be aware of what the system is up to that do not require logging into the control computer. A program called `rotsepager` monitors GCN activity and distributes alerts when events of interest occur. Also, `rotsed` (Section 1.2.1) provides up-to-the-minute status reports that can be viewed over any WWW browser. We describe these programs in this section.

8.4.1 `rotsepager`

This program is run on `rotse1.physics.lsa.umich.edu` (the same computer that hosts `www.rotse.net`, see below) by the `rpager` account. It establishes its own connection to the GCN network and is primarily responsible for alerting members of the ROTSE team to the distribution of GCN alert messages. It processes the GCN packets and applies a set of filters to determine whether the target coordinates for the alert are visible from each ROTSE-III site (it takes weather into account by scanning the JavaScript status files described in Section 8.4.2).

The configuration file, `pager.conf`, allows each user to determine which messages he or she wishes to receive and to which address they should be sent. This file defines three global variables, and then contains entries for each user who wishes to receive pages. Here is a sample user entry:

```
active      1                        # 1/0 = on/off
owner       somename                 # the holder of the pager
address     someaddress@place.com    # an email address for the pager
home        somename@umich.edu       # the home email address of the holder
sitemask    0x0001                   # only look at Australia
statmask    0x0007                   # check all status options
typemask    0x0001FFF0               # page on most useful trigger types
```

The three mask fields are bitmasks in hexadecimal format to allow the user to configure filter preferences. The sitemask field identifies the sites for which the owner wishes to receive alerts. The active bits are defined in Table 8.1. The statmask field indicates which status values should be checked before sending a pager. Status mask bits are defined in Table 8.2. The typemask field identifies the types of GCN alerts for which the owner wants to receive a page. Currently supported types are defined in Table 8.3.

As a self-diagnostic, `rotsepager` offers two types of special alerts that can be sent if you wish to remain aware of its status. Since the GCN in normal operation sends an IMALIVE (Type 3) packet once a minute, `rotsepager` will send out a page once a day at a particular time (this time is defined by the field `dailysod` in the same `pager.conf` file) upon receipt of an IMALIVE packet within a minute of the designated time. In case the connection go down, `rotsepager` keeps track of the time elapsed between the receipt of IMALIVE packets, and should this time become larger than ten minutes, it will send out an alert that GCN activity has ceased for ten minutes. Only one such alert will be sent. The user can decide whether to receive this alert as an email or as a page. To configure this self-diagnostic, the user should set the appropriate bits[2] as defined in Table 8.3.

8.4.2 Realtime Status Monitoring over the WWW

There is a computer at UofM that is configured to act as a web server for the URL `http://www.rotse.net`. Once a minute, a PERL script runs on this computer. It queries each ROTSE-III control computer in sequence over a socket connection with the `rotsed` on each system. It extracts a packet of status variable values and writes them to a local text file in JavaScript format. These variables give information about the state of the entire telescope, which daemons are running, what alarms are currently active, what the weather conditions are, and various header values from the most recent image recorded. The JavaScript files are accessed by the web pages on our site in order to display up-to-the-minute tables of the system status. See Figure 8.4 for an example of what an active display might look like.

Along with the JavaScript status variables, we also copy the thumbnail images that are produced by `sexpacman` (Section 8.2.1), as well as the analysis status monitor GIF file produced by `idlpacman` (Section 8.2.2). These images can be accessed through the pushing of buttons on the status display table page, as shown in Figure 8.4. Figure 8.2 shows a typical thumbnail, and Figure 8.3 shows an example of an analysis status image.

Upon receipt of a pager alert message from `rotsepager`, you may wish to monitor telescope response by periodically reloading `http://www.rotse.net/burst_response/`, which is automatically updated by software both at the site and on `rotse1`. These pages will contain tables listing any uncatalogued or highly variable sources near the published GRB position, along with light curve plots and cropped images for each observation. The presence of a candidate GRB counterpart should be immediately obvious from these data. See Sections 8.2.3 and 8.4.1 for more information.

[2]Unless you really, really want them, I recommend leaving type 2 off

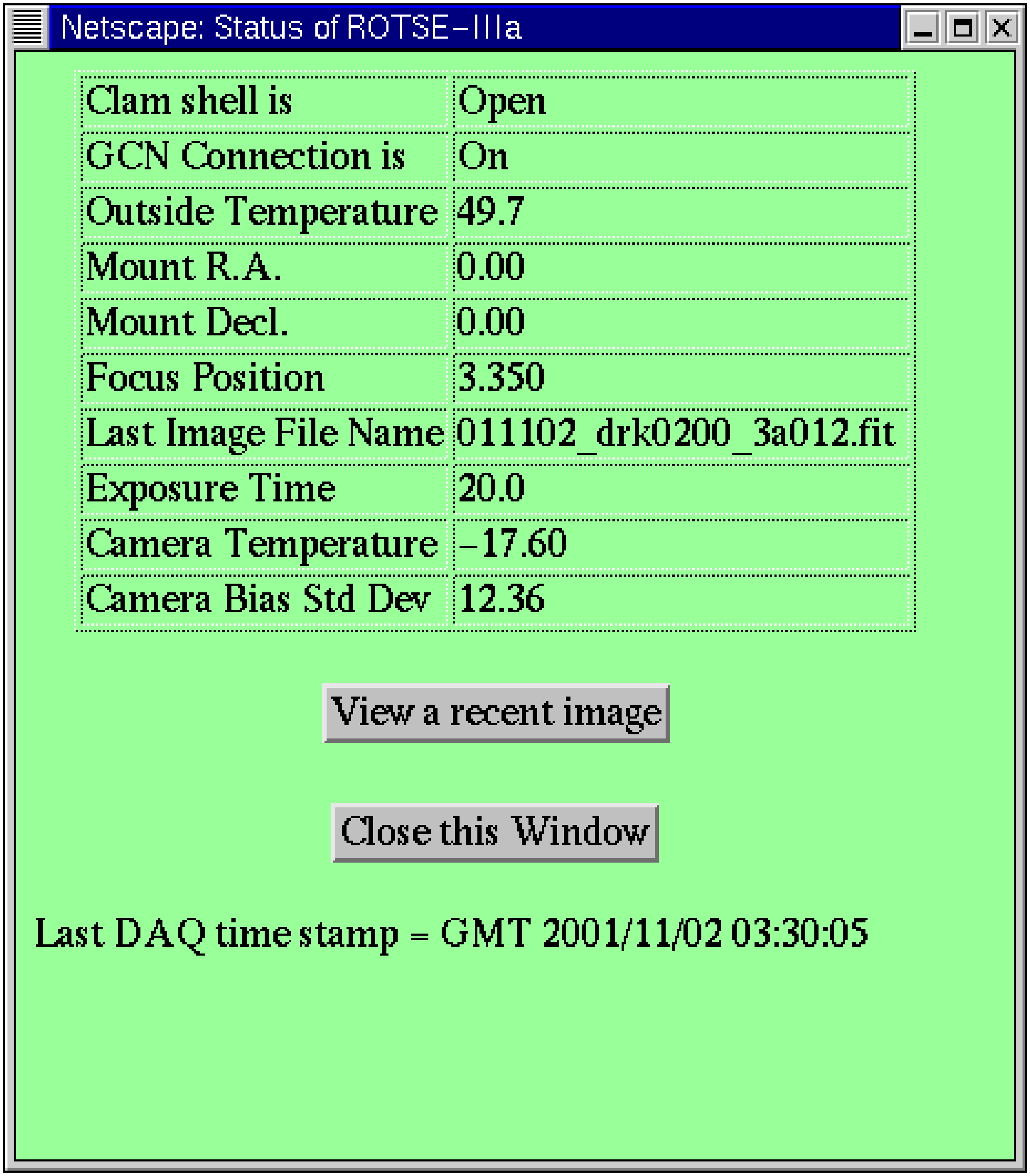

Figure 8.4: Sample window for a WWW display of current ROTSE-IIIa system status. This display is somewhat outdated. The most recent version also includes a button to display the analysis status figure (as shown in Figure 8.3) as well as the most recent thumbnail (Fig. 8.2). The contents of the table have been expanded and made more intelligent as well; some parameters are only shown if they take on “interesting” values.

Bitmask	Meaning
0×0001	Accept alerts for Australia
0×0002	Accept alerts for Namibia
0×0004	Accept alerts for Turkey
0×0008	Accept alerts for Texas
0×0010	Accept alerts for Maui

Table 8.1: Bitmasks for the ROTSE-III sites.

Bitmask	Meaning
0×0001	Check the weather status
0×0002	Check the clam shell status
0×0004	Check the camera status

Table 8.2: Bitmasks for status to check.

Bitmask	Meaning
0×0001	Daily I'm Alive Pages
0×0002	Socket Down Notification
0×0004	ALL Types (does not include daily alerts or error notification)
0×0008	Test Alerts: Types 2, 44, and INTEGRAL Tests. Not recommended.
0×0010	Type 27: RXTE–PCA GRB
0×0020	Type 29: RXTE–ASM GRB
0×0040	Type 39: IPN Position
0×0080	Type 40: HETE–ALERT
0×0100	Type 41: HETE–UPDATE
0×0200	Type 42: HETE–FINAL
0×0400	Type 43: HETE–GROUND
0×0800	Type 45: GRB_CNTRPART
0×1000	Type 51: INTEGRAL–POINTDIR (No GRB information)
0×2000	Type 52: INTEGRAL–SPIACS (GRB Timing information only)
0×4000	Type 53: INTEGRAL–WAKEUP
0×8000	Type 54: INTEGRAL–REFINED
0×00010000	Type 55: INTEGRAL–OFFLINE

Table 8.3: Bitmasks for the types of ROTSE-III alerts.

Chapter 9

Troubleshooting

9.1 Introduction

In this chapter I describe various possible failure modes of the ROTSE-III system, and what to do about them. Please report all bugs to the authors. This is by no means a comprehensive list of all permutations and combinations of things that can (and sadly, will) go wrong, but it will grow as more problems are found, and shrink as certain problems are fixed.

Please read Section 9.2 first, as it contains helpful wisdom to isolate the problem, which (hopefully) will be discussed in one of the following sections.

9.2 Tips and Tricks

The essential logfile in the ROTSE-III system is the system logfile `/var/log/rotse.log`. Use `tail -f` to watch the file go by, and `less` to explore it in more detail if the system has gone down.

When the system shuts down (either intentionally or unintentionally) all the daemons receive a `SIG: 15`, which is the standard Linux `TERM` signal. Seeing all the daemons report that they `got SIG: 15` is normal. We will need to find out which daemon started the avalanche.

To isolate the offending daemon, start at the end of the logfile (in `less`, the `>` command goes to the end) and search backwards for which daemon timed out. This can be accomplished by doing a backwards search in `less`, with the `?timeout` command. You will see output similar to the following:

```
...
Aug 12 21:14:50 rotse3a schierd:  Non-0 when resp = @Status2RA0000, 0003, 0000 39B3
Aug 12 21:14:50 rotse3a schierd:  error bit 3 set on axis 0
Aug 12 21:14:53 rotse3a schierd:  @RecentFaults Axis 1 High Output 8/12/2002, 20:19:36.277;
Aug 12 21:14:53 rotse3a schierd:  MOUNT_ERROR: Running error_recover()
Aug 12 21:14:53 rotse3a schierd:  Brake on axis 0
Aug 12 21:14:53 rotse3a schierd:  Drive amp disabled on axis 0
Aug 12 21:14:53 rotse3a schierd:  Reset failed.
Aug 12 21:14:53 rotse3a schierd:  unrecoverable error
Aug 12 21:14:53 rotse3a schierd:  begin shutdown, errno:  Interrupted system call
Aug 12 21:14:53 rotse3a schierd:  Clearing mount command...
Aug 12 21:14:53 rotse3a schierd:  Sending carriage return
Aug 12 21:14:53 rotse3a schierd:  closing mtfd
Aug 12 21:14:53 rotse3a schierd:  closing focfd
Aug 12 21:15:00 rotse3a rotsed:  daemon timeout (15.0098 > 15.00) exceeded, schierd
Aug 12 21:15:00 rotse3a rotsed:  obtain_status failed
...
```

In this case, `rotsed` is reporting that `schierd`, which controls the mount, is having a problem. The lines preceeding this timeout usually contain an error report from the daemon as it went down. In this case, the key lines are:

```
Aug 12 21:14:50 rotse3a schierd:  error bit 3 set on axis 0
Aug 12 21:14:53 rotse3a schierd:  @RecentFaults Axis 1 High Output 8/12/2002, 20:19:36.277;
```

This particular error will be discussed in Section 9.5.3.

If you are having problems with the camera, refer to Section 9.4 for specific information on the camera log file.

9.3 Help! The system won't start!

This section is for startup problems in general. Each subsection describes a different common problem, how to identify the problem, and how to fix it.

9.3.1 Problem: The system is already running

If startup fails, the following message might appear:

```
Aug 20 21:56:27 rotse3a rotsed:  open(/rotse/run/cfg/rotsed.cfg, O_WRONLY|O_CREAT|O_EXCL,
00664) failed, File exists
```

This is an indication that the `rotsed.cfg` file (which acts as a lockfile) is present. It is very possible that the system is already running! If it is not, then check the following Section 9.3.2.

9.3.2 Problem: The system crashed and left a lockfile

If this is the problem, you might see the error message from the previous section, or you might see `rmonitor` fail on startup with a comment such as:

```
error:  failed to get shared memory
```

This means that the `rotsed` system is *not* running, but the lockfile is present, perhaps from a previous system failure. The solution is simple: remove the lockfile with:

```
$ rm /rotse/run/cfg/rotsed.cfg
```

Then, you can safely restart the system.

9.3.3 Problem: A daemon configuration file has an error

A daemon will not finish initialization if its configuration file has an error, or has a missing field. The following comment is typical:

```
Aug 27 11:30:54 rotse4 rotsed:  error:  thumbfile not set in config file
Aug 27 11:30:54 rotse4 rotsed:  rotsed_conf failed
```

First, the daemon (in this case, `rotsed`) reports which field has not been set ("thumbfile"). Then the daemon reports that it's configuration routine failed, and the system will shut down. To solve this problem, please set the keyword in the proper configuration file.

Under normal running conditions the operator should not see this error, as configuration files are fairly static over time. However, this error often pops up after a recent software update– in that case, a template value should be available in the master configuration file in `/rotse/etc/`.

9.3.4 Problem: A daemon hasn't died properly from a previous run

Sometimes `weathd` or less frequently `spotd` get "stuck" on shutdown, and so a new system cannot start. The way to check if this is the problem is with a `ps` call:

```
$ ps auxww | grep rotse
```

If old daemons show up in the process table, that's the problem! Try the following:

```
$ killrotse -dammit
```

And if that doesn't work, occasionally you just need to reboot the control computer.

9.3.5 Problem: A temporary problem with the i/o box

Sometimes `clamd` or `spotd` (and occasionally `weathd`) has trouble connecting to the i/o box (on ROTSE-IIIa). This will result in a log message as the following:

```
Aug 12 21:36:17 rotse3a spotd: open of /dev/ml16pa-digital failed: No such device
...
Aug 6 23:21:09 rotse3a rotsed: daemon init timeout (10.0045 > 10.00) exceeded, clamd (or spotd)
```

If this happens, wait at least 30 seconds and try to start the system again. It should work.

9.4 Help! The camera isn't working right!

9.4.1 The `camserverd` logfile

Problems with the camera hardware, which is run by `camserverd`, need to be explored on the camera computer. The system log on the camera computer is also called `/var/log/rotse.log`, and is very similar to the logfile on the control computer. You should look in this logfile to debug any camera problems.

9.4.2 Problem: The image is saturated

If the image is completely saturated, or fails basic quality cuts (e.g., the minimum value in the central subframe is above some cut-off, usually 30000), the system will only write the fits header to disk. In the logfile on the camera computer you will see a message similar to:

```
Aug 21 01:07:16 cam3a camserverd: image screwed up, not writing it: 0
```

When you look at the image with `ls`, you will see:

```
-rw-rw-r-- 1 observer observer 8640 Aug 21 08:30 020817_sky1409+0148_3a001_c.fit
```

Reading the fits header will make the problem obvious: for this image, the mean in the central subframe was 33035, very close to saturation.

The solution to this problem is simple: take a shorter exposure, or wait for clear weather!

9.4.3 Problem: The power supply has died

It can happen that the image will fail quality cuts, as mentioned in the previous Section 9.4.2, but the problem might be more serious. When the camera is not getting proper power, there are two important diagnostics:

1. The camera temperature is not read out correctly, usually reverting to -280° C. Our cooling system, however, obeys the laws of thermodynamics.
2. The MEAN in the central SUBFRAME is fixed at 32000, while the STDDEV is 0.0
3. The chip isn't cooling at all, the temperature stays at 13.4° C, and you see lots of hot pixels.

A likely problem is that one of the air filters on the camera power supplies has clogged up, and there is insufficient airflow to cool the power supply. First, check which power supply is not running. (You need to listen carefully to figure this out). Next, try to clear the air filters. If that works, great!, if not, please contact the authors and Bob Leach at ARC cameras (leach@astro-cam.com) for information on resurrecting the power supplies.

9.5 Help! The mount is having "issues"!

9.5.1 Problem: The mount control software is not running

It has happened that the mount control software (on the mount computer running Windows NT) is not running, or equivalently, the mount computer is not powered on. In this case, on startup you will see errors such as:

```
Aug 23 11:43:42 rotse3a schierd:  No response to:  $HaltRAe4a3M̂
```

This message means the mount computer did not respond at all to the command to halt the RA axis of the mount. The solution is to reboot the mount computer, which cannot be done remotely.

9.5.2 Problem: The encoder is dirty or misaligned

If `schierd` is having problems, sometimes the error log has the following line:

```
Aug 11 16:40:49 rotse3a schierd:  Error reported on RA Axis:
Aug 11 16:40:49 rotse3a schierd:  @RecentFaults Axis 1 Encoder Failure 8/11/2002, 15:45:12.767;
```

What exactly does this mean? When confronted with the problem, Alan Schier quipped "Well, it sounds like the Axis 1 encoder failed." It should be noted that axis 1 is the RA axis, and axis 2 is the Dec axis.

There are two possible problems. There could be some dirt on the encoder tape, causing the encoder head to lose track of the tape. The solution here is to clean the encoders as well as possible.

Alternatively, the encoder head could be slightly misaligned, so that the head loses contact with the tape at a certain point. This misalignment might be very small, and the range over which contact is lost might be very small as well. The solution is to very gently and carefully re-align the encoder head. This might take some time (over an hour), but it is time well spent.

9.5.3 Problem: The balance is off or a cable is snagged

If `schierd` is having problems, the error log might have the following line:

```
Aug 12 21:14:53 rotse3a schierd:  Error reported on RA Axis:
Aug 12 21:14:53 rotse3a schierd:  @RecentFaults Axis 1 High Output 8/12/2002, 20:19:36.277;
```

In this case, the RA axis (axis 1) motor has tried pushing too hard. This is usually caused when the mount is trying to track through a region where the balance is bad; the motor tries to keep the axis steady, and fails. On telescope set-up, we should have a systematic check to ensure that the telescope can track through the entire hemisphere.

It is also possible that there is a cable snag. After making adjustments to the mount, make sure that all cables are securely locked away.

Appendix A

ROTSE-III File Naming Conventions

In order to convey useful information about ROTSE-III files at a glance, as well as to avoid the danger of overwriting important files, we have implemented a standardized naming convention for all files in the data analysis pipeline. These files are usually FITS format, and thus take the `.fit` extension at the end of the file name. The name itself is broken into four parts, separated by underscore symbols:

`[date]_[tla][id numbers]_[instrument designation][index number]_[optional modifier].fit`

The date is straightforward: a six-digit representation of the year, month, and day in that order: YYMMDD. The other parts will be explained in the following sections, along with the standardized file system structure for where they are located.

A.1 Three-Letter Acronyms (TLA)

Each type of imaging program that a ROTSE-III telescope might execute is identified through a three-letter acronym (often called a TLA). These identify from the file name the reason a particular image was taken. GRB responses have TLAs that identify the satellite that originated the alert, as well as the type of alert that was disseminated. Sky patrol, burst follow-up, as well as various housekeeping observations all have their own TLAs. The full list of standardized ROTSE-III TLAs is given in Table A.1. As described in Section 4.4.1, the user must define his or her own TLAs through the `astrod` daemon for scheduled targeted observations.

A.2 Coordinates or ID Numbers

For prompt burst response images, the TLA is followed by an index number that identifies the GCN serial number of the alert of that particular type (as identified by the TLA, see Section A.1). This number will have at least four digits, but may have more. In the case of a simulated burst alert, the numbers will run sequentially from a number that is determined by `simGCNalert`, one number for each alert. For real GCN alerts, there is a unique number included in the packet array to identify the alert, and this number is passed on to the file name by the DAQ system.

In contrast, for most other types of images, including sky patrol, targeted observations, and burst follow-up images, the TLA is followed by a nine-digit field which gives the coordinates of the center of the field to four-digit accuracy. The Right Ascention is given as HHMM, and the Declination is given by ±DDMM. Fractional minutes are rounded to the nearest whole number.

Finally, dark images (`drk`) do not have coordinates. In this case, the four-digit identifying number following the TLA is the exposure time in tenths of seconds.

A.3 Instrument Designation and Index Number

There is a two-digit code to identify the ROTSE instrument from which a given image originated. For ROTSE-III, these digits will be either "`3a`","`3b`","`3c`", or "`3d`", depending on which of the four ROTSE-III telescopes

TLA	Explanation
drk	Dark Image
foc	Focus Image
hom	Home Check
twi	For making a Twilight Flat
tpt	For making a Pointing Model
pht	Photometry Verification after Burst Response
tla	Test alert
gha	HETE-2 alert
ghu	HETE-2 Update alert
ghf	Final HETE-2 alert
ghg	Alert from HETE-2 Ground Analysis
gbb	Alert from BeppoSAX WFC
gbn	Alert from BeppoSAX NFI
sky	Sky Patrol
fup	Burst Follow-Up

Table A.1: ROTSE-III standard three-letter acronyms to indicate the reason for why a particular image was recorded. Additional TLAs are defined by the user for targeted observation sequences.

Modifier	File type
c	Corrected Image Files
sobj	Object lists output from SExtractor
cobj	Calibrated object lists
cal	Calibration data (obsolete – now included in cobj file)
sky	Average sky background (also in cobj file)
match	Match structures
relmat	Relative Photometry-corrected match structures

Table A.2: Optional file modifiers for ROTSE-III data products.

is appropriate. The choice is specified in the `camerad.conf` file (Section 3.5). If more ROTSE-III telescopes are implemented, the extension of the sequence is obvious.

The index number simply distinguishes between images taken on the same day, for the same reason, of the same field, and with the same telescope. This number increments with each image taken as part of a given imaging sequence (Section 4.4.1) in response to a given command. For sky patrol images, the number increases with each image taken of a particular field on a particular night. For burst alerts, this is simply a chronological numbering of all images taken in response to the same alert.

The median dark images described in Section 7.2.2 do not have index numbers after the instrument designation, but the individual images that make up a dark run will be ordered in the sequence in which they were taken.

A.4 Optional Modifiers

As each image is processed by the data analysis, modifier strings are appended to the basic file name to distinguish each stage in the process while still conveying the file's origin in a particular image. These modifiers are summarized in Table A.2, and the details are explained in the description of the analysis software in Chapter 8.

Directory or File Path	Description of Contents
/rotse/run/bin/	Directory with Binary Files
/rotse/run/etc/	Directory with Configuration Files
/rotse/run/cfg/	Directory with rotsed lockfile
/rotse/run/log/	Directory with daemon logfiles
/var/log/rotse.log	ROTSE logfile

Table A.3: Standard definitions for the locations of ROTSE-III system files.

Directory Path	Description of Contents
/rotse/data/3a1/	Directory with non-alert image files
/rotse/data/3a1/links/	Directory with links to raw image files
/rotse/data/3a2/	Directory with alert image files
/rotse/data/pipeline/	Automated Pipeline Parent Directory

Table A.4: Standard definitions for the locations of ROTSE-III data products. The letters "3a" should be replaced with the appropriate two-letter designation for the telescope in question, see Section A.3.

A.5 Match Structures

Since match structures (Section 8.3.2) will contain data from many images, potentially on different dates, match structure file names will not contain either the date field or the index number. The rest of the name (such as "ghu0003_3a" or "sky0350–0148_3a") is called the "root" name, and so the match structure file will be named either "ghu0003_3a_match.fit" or "ghu0003_3a_relmat.fit", depending on whether or not the relative photometry procedure (Section 8.3.3) has been applied to it.

A.6 Standard Data Directory Structure

On each telescope control computer, programs, configuration files and data products are put in standard places so that they can be easily accessed. The locations of the files necessary for telescope operations are described in Table A.3. Image data are placed on different discs, and these directories are described in Table A.4. The first two directories are on the camera computer, and the third is on the control/analysis computer. The directories on the camera computer contain the raw images as recorded from the camera via `camserverd`. The images are not separated into subdirectories, and the only standard subdirectory is called `links` in `3a1`, which contains the symbolic links which are processed by `sexpacman`.

Within the `pipeline` directory identified in Table A.4 are several important standard directories and data files. These files are summarized in Table A.5 and are described here in more detail. The `pipeline` directory contains six subdirectories: `image`, which contains all the corrected images, `prod`, which contains the `sobj` and `cobj` files (as well as a directory `match` for any archived match structures), `html` for the WWW-formatted output from `idlpacman`'s burst response, `cal`, which contains dark and flat field files and anything else needed for calibration, `thumbs`, which contains the thumbnail JPEG images produced by `sexpacman.pl`, and `transfer`, which will be used hold files intended for transfer back to Michigan. The `pipeline` directory also contains the configuration and log files used for `sexpacman` and `idlpacman`, as described in Sections 2.3.3, 8.2.1, and 8.2.2.

Directory or File	Description of Contents
`image`	Directory for corrected image files
`prod`	Directory for `cobj` and `sobj` files
`prod/match`	Directory for match structure files
`html`	Directory for burst response products to view
`cal`	Directory for dark, flat, and fringe field files
`thumbs`	Directory for JPEG thumbnails
`transfer`	Directory for things to transfer to UofM
`sex.conf`	Configuration file for `sexpacman.pl`
`sex.log`	Log file for `sexpacman.pl`
`sobjlist`	Output list of `sobj` files created by `sexpacman.pl`
`idlpac.conf`	Configuration file for `idlpacman.pro`
`idlpac.log`	Log file for `idlpacman.pro`

Table A.5: Standard products to be located within the pipeline directory identified in Table A.4.

Appendix B

FITS File Information

B.1 FITS File Description

The Flexible Image Transport System (FITS) format was created by NASA and is the standard format for images and data structures in the astronomical community. More information can be found online at `fits.gsfc.nasa.gov`.

A FITS file contains an unlimited number of extensions, each of which can contain image data or binary tabulated data. Each extension has a meta-data "header" that describes the data format for that extension. The FITS file itself contains a primary header that can contain several standard keywords, as well as unlimited user-definable keywords. The ROTSE-III keywords are described in Section C.1 and C.2. The FITS format is, as the name suggests, extremely flexible.

The primary extension (number 0) can only contain an image, and cannot contain an arbitrary data structure. Each additional extension can contain either image data or binary tabulated data. None of the ROTSE-III FITS files contains image data beyond extension 0. The FITS standard also allows for simple image compression from 32-bit floating point values to 16-bit integer values using simple scaling. The standard header keywords `BZERO` and `BSCALE` are used to compress and uncompress an image in the following way:

Compress:	Integer Value = (Float Value - `BZERO`) / `BSCALE`
Uncompress:	Float Value = (Integer Value) * `BSCALE` + `BZERO`

FITS images conform to the World Coordinate System (WCS) standard. With WCS information in the image header, an image viewer is able to translate x and y coordinate values to RA and Dec. Currently, we only have a simple WCS rotation matrix implemented, so the RA and Dec positions are only approximate. Extensions to WCS for a more refined transformation exist, but have not yet been implemented in our analysis code.

The IRAF program suite contains many tools for working with FITS images. For various reasons, we do not use the IRAF suite. Many of our basic image processing tools (`makedark` and `makeflat` for example) are based on Stuart Marshall's "miraf" code. For other functions, we have our own pipeline code that runs much faster in C and IDL.

B.2 Accessing FITS Files: Command Line

B.2.1 Primary FITS Headers: `mhead`

To access the fits header, Stuart Marshall's `mhead` program is simple to use, and prints the entire header to the terminal. Combining `mhead` with `grep` returns a given keyword line.

```
$ mhead imagefile.fit
SIMPLE  =                    T / file does conform to FITS standard
...
```

```
$ mhead imagefile.fit | grep FOCUS
FOCUS   =               3.29648 / Focus position (mm)
$
```

B.2.2 Image Data: ds9

For FITS image viewing, there are many choices out there. We use `ds9`[1] as it combines flexible features with a lightweight package. The program `gaia` has more analysis features (notably point-and-click FWHM estimates), but is much slower, so is inconvenient for day-to-day use. To use `ds9`, simply call it from the command line. `ds9` is fully compliant with the WCS standard.

B.2.3 Binary Structure Data

There are command line tools to view FITS binary tables, but we don't have them installed. For these structures, using IDL is quite useful.

B.3 Accessing FITS Files: IDL

FITS file access from IDL is accomplished with the IDL astronomy users library. Detailed documentation can be found on the web at `idlastro.gsfc.nasa.gov/contents.html`.

B.3.1 Primary FITS Headers: headfits.pro

The primary FITS header can be accessed with the `headfits.pro` function. Individual keywords can be accessed with `sxpar.pro` as below:

```
IDL> hdr=headfits('filename.fit')
IDL> print,hdr
SIMPLE  =                     T / file does conform to FITS standard
...
IDL> focus = sxpar(hdr,'FOCUS',count=count)     ;count returns 0 if the keyword is not found
IDL>
```

B.3.2 Image Data: readfits.pro

The primary image extension is accessed with the `readfits.pro` function. This function automatically scales the raw pixel values with the FITS standard `BZERO` and `BSCALE` parameters. Image display is accomplished with the `rdis_setup.pro` and `rdis.pro` routines.

```
IDL> im=readfits('filename.fit',hdr)    ;the hdr (header) argument is optional.
IDL> rdis_setup,im,pls                 ; set up the rdis plot structure
IDL> rdis,im,pls                        ; plot the image with hist.  eq.
```

B.3.3 Binary Data: mrdfits.pro

The secondary binary structure extensions are accessed with `mrdfits.pro`. Please note that `mrdfits` does *not* properly use `BZERO` and `BSCALE`, and should not be used to read in image data! The binary structures come out exactly as they were written, with the same structure tags.

```
IDL> cobj=mrdfits('cobjfile.fit',1)     ;The 1 signifies the 1st extension
MRDFITS: Binary table.  7 columns by 4323 rows.
IDL> help,cobj,/str
```

[1]SAOimage ds9 is the third SAOimage viewer. The first was called `saoimage`, and the second tng. I am not sure if Voyager or Enterprise is the next version...

```
RA          FLOAT        21.6110
...
IDL>
```

B.4 Accessing FITS Files: C

Accessing FITS files from C is accomplished in the ROTSE daq system with the `cfitsio` C library. We are currently running version 2.0, although many more features are now available. For detailed information visit the website at `heasarc.gsfc.nasa.gov/fitsio/`.

Appendix C

ROTSE-III Data Products

C.1 Raw Images

The raw images contain a two-dimensional array of 16-bit unsigned integers that characterize the photon flux at each pixel on the CCD chip. The image is located at the primary FITS extension, as described in Section B.1. Each FITS image has FITS header that contains both standard keywords and ROTSE specific keywords. The header values are listed below, with a few fields left blank to be filled in when the image is corrected. These values are described in Section C.2.

BITPIX	number of bits per data pixel
NAXIS	number of data axes
NAXIS1	length of data axis 1
NAXIS2	length of data axis 2
EXTEND	FITS file may contain extensions
BZERO	offset data range
BSCALE	default scaling factor
OFFSET1	Camera upper left frame x
OFFSET2	Camera upper left frame y
XFACTOR	Camera x binning factor
YFACTOR	Camera y binning factor
DATE-OBS	Date of start of OBS in GMT
LOCTIME	Exposure local start time
MJD	Julian Date -- 2400000.5
OBSTIME	Start Time of OBS in GMT (sod)
ALTITUDE	Observatory altitude (meters)
LATITUDE	Observatory latitude (deg)
LONGITUD	Observatory longitude (deg)
NCOADD	Number of co-added images
CTYPE1	RA, TAN projection used
CRPIX1	pixel at reference point
CRVAL1	RA at the reference point
CDELT1	increment per pixel (degrees)
CUNIT1	physical units of axis 1
CTYPE2	DEC, TAN projection used
CRPIX2	pixel at reference point
CRVAL2	DEC at the reference point
CDELT2	increment per pixel (degrees)
CUNIT2	physical units of axis 2
PC001001	coordinate rotation matrix (WCS)
PC002002	coordinate rotation matrix
PC001002	coordinate rotation matrix
PC002001	coordinate rotation matrix
CAM_ID	Two-digit ID tag for instrument
OFFSTRA	Camera offset in RA (deg*cos(dec))
OFFSTDEC	Camera offset in Dec (deg)
OPTICS	Optics manufacturer and model
OPTICSN	Optics serial number
FOCUS	Focus position (mm)
SATCNTS	Saturation Level
NSAT	Number of Saturated Pixels
ENCRA	Encoder RA
ENCDEC	Encoder Dec
CAMTYPE	Camera manufacturer and model
CAMSN	Camera serial number
CARDSN	PC card serial number
PATH	Path name of FITS file
FILENAME	FITS file name
NFRAME	Frame number
EXPTIME	Exposure time, seconds
EFFTIME	Eff. co-added time, seconds
COOLER	TE cooler status
CAMTEMP	Camera temperature (C)
AMP	0:Left, 1:Right, 2:Both
MEAN	Mean of SUBFRAME
STDDEV	Std dev. of SUBFRAME
MEDIAN	Median of SUBFRAME
MIN	Minimum pixel value of SUBFRAME
MAX	Maximum pixel value of SUBFRAME
SUBFRAME	SBF bounds (xmin:xmax,ymin:ymax)
BMEAN	Mean of BIASFRAME
BSTDDEV	Std dev. of BIASFRAME

BMEDIAN	Median of BIASFRAME	NEARTILE	Four nearest Sky Patrol frames
BMIN	Minimum pixel value of BIASFRAME	MOUNT	Mount manufacturer and model
BMAX	Maximum pixel value of BIASFRAME	MOUNTSN	Mount serial number
BIASFRAM	BSF bounds (xmin:xmax,ymin:ymax)	MOUNTRA	Right Ascension J2000 (deg)
OBSTYPE	Observation type	MOUNTDEC	Declination J2000 (deg)
TRIG_NUM	Trigger number	MOUNTERR	Mount position error (deg)
TRIG_T	Trigger time (sec-of-day)	TEMPOUT	Outside temperature (F)
PKT_T	Time trigger sent (sec-of-day)	WINDSPD	Wind speed (mph)
TRIG_RA	Trigger RA (deg)	WINDDIR	Wind direction (deg)
TRIG_DEC	Trigger Dec (deg)	BAROM	Barometric pressure (inches Hg)
TRIG_ERR	Trigger Error (deg)	HUMIDITY	Outside humidity (%)
INTEN	Trigger intensity	DEWPOINT	Dew point (F)
LPHASE	Lunar Phase (= -1.0 when down)	SKYMON	Sky Monitor voltage
DMOON	Ang. distance to Moon (deg)	CLOUDS	Cloud Monitor voltage
ELEV	Elevation (deg)	VPRECIP	Vaisala Precipitation voltage
AZIMUTH	Azimuth (deg)		

C.2 Corrected Images

After applying the dark, flat, and fringe field corrections, a final image is written with an "_c" naming extension (Section A.4) in a samilar FITS format as the raw images. The corrected images contain 32-bit floating point values compressed to 16-bit integers using the BZERO/BSCALE method described in Section B.1. The header keywords are the same as the raw image files, only the following keywords are now be filled in:

DARKNAME	Dark File name
FLATNAME	Flat File name
FRINGE	Fringe File name
FSCALE	Fringe Scaling
FCHISQ	Fringe Chisq

If the fringe fit was not good, as described in Section 7.5, an exclamation point ("!") is placed before the fringe file name to indicate it was *not* subtracted.

C.3 sobj Files

This kind of FITS file is output from SExtractor as described in Section 8.2.1. An sobj file consists of a primary header and an array of structures. The latter is saved as the first FITS extension. Use the IDL program mrdfits('sobjname.fit',1) as described in Section B.3 to access the object list. The primary header consists of the FITS standard keywords, and the following additional keywords:

HISTORY		SEXBKDEV	MEDIAN RMS (ADU)
EPOCH		SEXBKTHD	EXTRACTION THRESHOLD (ADU)
OBJECT		SEXCONFF	CONFIGURATION FILENAME
ORIGIN		SEXDETT	DETECTION TYPE
SEXIMASX	IMAGE WIDTH (PIXELS)	SEXTHLDT	THRESHOLD TYPE
SEXIMASY	IMAGE HEIGHT (PIXELS)	SEXTHLD	THRESHOLD
SEXSTRSY	STRIP HEIGHT (LINES)	SEXMINAR	EXTRACTION MINIMUM AREA (PIXELS)
SEXIMABP	FITS IMAGE BITPIX	SEXCONV	CONVOLUTION FLAG
SEXPIXS	PIXEL SCALE (ARCSEC)	SEXCONVN	CONVOLUTION NORM. FLAG
SEXSFWHM	SEEING FWHM (ARCSEC)	SEXCONVF	CONVOLUTION FILENAME
SEXNNWF	CLASSIFICATION NNW FILENAME	SEXDBLDN	NUMBER OF SUB-THRESHOLDS
SEXGAIN	GAIN (IN E- PER ADU)	SEXDBLDC	CONTRAST PARAMETER
SEXBKGND	MEDIAN BACKGROUND (ADU)	SEXCLN	CLEANING FLAG

Bit value	Tag Name	Meaning
1	NEIGHBORS	Not used with MAG_APER
2	BLENDED	Object was originally blended with another one.
4	SATURATED	At least one pixel of the object is saturated.
8	ATEDGE	The object is truncated (too close to an image boundary).
16	APINCOMPL	Object's aperture data are imcomplete or corrupted.
32	ISINCOMPL	Object's isophotal data are incomplete or corrupted.
64	DBMEMOVR	Memory overflow occured during deblending.
128	EXMEMOVR	Memory overflow occured during extraction.

Table C.1: Possible values for the FLAGS bitmask field in the raw object list structures.

SEXCLNPA	CLEANING PARAMETER
SEXCLNST	CLEANING OBJECT-STACK
SEXAPERD	APERTURE DIAMETER (PIXELS)
SEXAPEK1	KRON PARAMETER
SEXAPEK2	KRON ANALYSIS RADIUS
SEXAPEK3	KRON MINIMUM RADIUS
SEXSATLV	SATURATION LEVEL (ADU)
SEXMGZPT	MAGNITUDE ZERO-POINT
SEXMGGAM	MAGNITUDE GAMMA
SEXBKGSX	BACKGROUND MESH WIDTH (PIXELS)
SEXBKGSY	BACKGROUND MESH HEIGHT (PIXELS)
SEXBKGFX	BACKGROUND FILTER WIDTH
SEXBKGFY	BACKGROUND FILTER HEIGHT
SEXPBKGT	PHOTOM BACKGROUND TYPE
SEXPBKGS	LOCAL AREA THICKNESS (PIXELS)
SEXPIXSK	PIXEL STACKSIZE (PIXELS)
SEXFBUFS	FRAME-BUFFER SIZE (LINES)
SEXISAPR	ISO-APER RATIO
SEXNDET	NB OF DETECTIONS
SEXNFIN	NB OF FINAL EXTRACTED OBJECTS
SEXNPARA	NB OF PARAMETERS PER OBJECT

The object list consists of an array of structures: one element for each object. A single structure in this array has the following fields:

NUMBER	Object number
MAG_APER	Aperture Magnitude
MAGERR_APER	Uncertainty in same
BACKGROUND	Local Background (ADU)
FLUX_MAX	Peak Object Flux (ADU)
MU_MAX	??
ISOAREA_IMAGE	Isophotal Area of Object
X_IMAGE	X-coordinate of object (FITS Coord + 1)
Y_IMAGE	Y-coordinate of object (FITS Coord + 1)
FWHM_IMAGE	Measuring the PSF
FLAGS	Bitmask of SExtractor flags

The FLAGS values are described in the `SExtractor` documentation and are summarized in Table C.1.

C.4 `cobj` Files

Once the object list has been calibrated against the USNO A2.0 catalog, we are in a position to know how to map the image coordinates to the sky, and we can fill in all of the image file headers. The following fields now have meaningful values that define a (very rough) WCS rotation matrix to perform this conversion: `PC001001`, `PC002002`, `PC001002`, and `PC002001`. These numbers are used by `ds9` to calculate world coordinate system units for display, but are not accurate enough for real analysis.

A `cobj` FITS file has a primary header and two data extensions. The primary header is taken directly from the corrected image. The first data extension contains the calibrated object list. The second data extension contains the "calibration structure," or "cal" structure.

The calibrated object list at extension 1 is an array of structures, one element per object, each element of which contains the following fields:

RA	Right Ascention (deg)
DEC	Declination (deg)
X	Horizontal Pixel coordinate
Y	Verticle Pixel coordinate
M	Calibrated magnitude
MERR	Uncertainty in above

Bit value	Tag Name	Meaning
1	HOTPIX	Object falls on hot pixel in this observation
2	USNOCAT	Object is present in USNO Catalog
4	ASTEROID	Object is most likely an asteroid
8	BADPOS	Position centroid for this observation is too far from mean position
16	NOTEMPL	Relphot subframe failed to identify sufficient template objects
32	PHOTSDEV	Relphot subframe RMS correction is large
64	BADIMAGE	A catch-all image-wide problem flag

Table C.2: Possible values for the RFLAGS bitmask field in the calibrated object list structures.

FLAGS	Same as in `sobj` file
RFLAGS	Additional ROTSE flags (See Table C.2)

The cal structure at extension 2 contains a single structure with much information. The bulk of the cal structure consists of the corrected image header keywords converted from ASCII to a binary structure. In addition are several important keywords from the `sobj` header. Finally are calibration diagnostic values described here:

BADPIXFILE	The name of the bad pixel file
NMATCH	The number of matched objects
OFFSET_X	
OFFSET_Y	
POS_SIGMA	The RMS error in position
RA_LOW	Image range on sky
RA_HIGH	
DEC_LOW	
DEC_HIGH	
ZP_OFFSET	Magnitude offset
ZP_SIGMA	Uncertainty in offset
M_LIM	Limiting magnitude
FNAME	Name of file
RAC	Coordinate of center
DECC	
KX	Third order rotation matrix
KY	to convert coords
SAT_MAG	Saturation magnitude
NOBJ_BADPIX	Objects that fall on bad pixels
SKY	64×64 array for sky bkgd

C.5 Match Structures

A match structure FITS file consists of two extensions. There is no useful information in the header. The second FITS extension is simply an array of all the cal structures that were stored in the second extensions of the `cobj` files that went into creating the match structure.

The first FITS extension contains the match structure itself, which is an array of structures, one element for each object in the calibrated object lists that compose the match structure. The fields in each structure are defined in Table C.3, where *nobj* is the number of objects in the structure, and *nobs* is the number of observations included in the match.

Field Name	Datatype	Format	Contents
KX	FLOAT	Array[$nobs$, 4, 4]	Coordinate Conversion Matrix
KY	FLOAT	Array[$nobs$, 4, 4]	Same
JD	DOUBLE	Array[$nobs$]	Date in Modified Julian Day
EXPTIME	FLOAT	Array[$nobs$]	Exposure Time
IMAGENAME	STRING	Array[$nobs$]	Image file name
RAC	FLOAT	Array[$nobs$]	Center of image
DECC	FLOAT	Array[$nobs$]	Center of image
RAL	FLOAT		Image limits
RAH	FLOAT		
DECL	FLOAT		
DECH	FLOAT		
M	FLOAT	Array[$nobs$, $nobj$]	Object magnitude
MERR	FLOAT	Array[$nobs$, $nobj$]	Uncertainty in M from SExtractor
FLAGS	INT	Array[$nobs$, $nobj$]	SExtractor flags
DRA	LONG	Array[$nobs$, $nobj$]	Observation Position * 1000000
DDEC	LONG	Array[$nobs$, $nobj$]	Same
RFLAGS	BYTE	Array[$nobs$, $nobj$]	ROTSE flags
MSYS	BYTE	Array[$nobs$, $nobj$]	Systematic Error Estimate
RA	DOUBLE	Array[$nobj$]	Mean location of object (deg)
DEC	DOUBLE	Array[$nobj$]	Same (deg)
NUMOBS	INT	Array[$nobj$]	# of obs in which object was in FOV
CONSEC	BYTE	Array[$nobj$]	Was object in consecutive obs?
NGOOD	INT	Array[$nobj$]	# of good obs in which obj was detected
MAVG	FLOAT	Array[$nobj$]	Mean magnitude of each object
MSTD	FLOAT	Array[$nobj$]	RMS scatter in M for each object
M_LIM	FLOAT	Array[$nobs$]	Limiting magnitude of each obs

Table C.3: Fields that go into a ROTSE-III match structure. Values that vary for every object in each field, such as magnitude, are two-dimensional arrays, while values that only vary from observation to observation, such as the average magnitude, are one-dimensional arrays. The overall limits on the R.A. and Decl. coordinates are four floating point numbers. The variables $nobs$ and $nobj$ refer to the number of observations and the number of objects, respectively, contained in the match structure

C.6 Relative Photometry-Corrected Match Structures

A match structure upon which the relative photometry algorithm (Section 8.3.3) has been applied still contains the exact same structure as the original match structure (although some of the data values in that structure have been changed, of course). What is new is a third FITS file extension has been added, to preserve useful information about the correction process. This third extension is a single structure with the following fields:

NPIX	Number of sub-fields used	OFFSET	Array of mean offsets per subfield
NXBIN		ERROR	
NYBIN		NTMP	
NTHR		SDV	
NOISE			

Index