Software Manual
Spartan IR Camera for the SOAR Telescope

Edwin D. Loh

Department of Physics & Astronomy
Michigan State University, East Lansing, MI 48824

Loh@msu.edu 517 355–9200 x2480

22 March 2006
Revised 13 November 2006
Revised 17 April 2007
Added Health of Instrument 7 April 2009

Contents

1 Software Overview

The software uses LabView, a graphical language, and we adopt the terminology of LabView.[1]

[1] A program or subroutine is called a virtual instrument (VI). A VI has a front panel, which is for operating it, and a diagram, which is the "code." The front panel may be visible, the case for VIs that the observer

Much of the documentation for using the software is on the front panel of the VI. SpartanGUI has **tip strips** and **context-based help**, which explain most controls and indicators. A tip strip for a control or indicator pops up with a message when the mouse moves over it. To see the help for controls or indicators, press <cntl>+H to turn on context-based help. Then move the mouse over a control or indicator. Pressing <cntl>+H another time turns off context-based help.

1.1 Getting Started

There are two user interfaces:

SpartanGUI is the control panel for the observer and the engineer. In normal operation, the observer need only look at this window. Status information is in the top section. The observing functions are in the observing panel. The notebook maintains a record of the observations; the observer can log comments in the notebook. To start the software, open `\homeSpartan\SpartanGUI.vi`, and the window in Figure 1 should appear. Then select the **Help** tab control for instruction. SpartanGUI has several tab controls.

Observing (Figure 1) is for the observer. The observer presses buttons on SpartanGUI to set exposure time, take pictures, change filters, and switch between the wide-field and high-res modes.

Setup (Figure 3) is to setup the detector and the mechanisms. The operations are (1) choose the detectors, (2) load the operating voltages for each detector, (3) initialize the mechanisms, (4) test the home positions of the mechanisms, and (5) find and store new home positions. The parameters that are unlikely to need changing are in a tab control, the two tabs of which are named Detector and PlugIn. Select the button FauxHardwareOK to run without any hardware. Observers should not normally use this tab.

forMechanismEngineer (Figure 4) is for monitoring the mechanisms and more detailed control of them. Normally, the mechanism moves by the amount needed to change the optic, an example of which is the 20° to move between filters. This panel allows movement by steps of 0.002°. Observers should not normally use this tab.

---

controls directly, or hidden. The front panels of most VIs are kept hidden, since the observer does not need to see the internal details of the software. Controls are devices on the front panel by which the user controls the software. Examples of controls are buttons and boxes in which to enter text. Indicators are devices on the front panel by which the software gives information to the user. Examples of indicators are lights and boxes in which the software writes data.

Glossary contains definitions of terms, an optical schematic, and a map of the detector layout.

InstLog maintains a record of mechanism movement and unexpected problems. The observer may add comments that will help diagnose problems.

Help (Figure 5)

In order to reduce clutter, most of the VIs run without a visible window. To make a VI (CameraControl, LogTempPressure, MechanismInitiation, MechanismHoming, MechanismMoving, PGauge, TUI Link) visible, use the Window pulldown menu of SpartanGUI. (The pulldown menus are not visible in the figures.) Seeing these VIs is useful for diagnosing problems but not useful for observing.

StartanTUI is a text-based user interface for the instrument. It allows the user to run a sequence of operations using a script. To start the text-based user interface, open `\homeSpartan\SpartanTUI.vi`. The window in Figure 2 appears. Then press the run button, which is a fat arrow at the top of the window, or press <ctrl>+R. See §2 for detailed instructions.

To test your command or script for syntax errors, press the button **Test Command**. To run the command or script, press the button **Run Command**.

1.2 Operating Other Components

Beside the user interfaces, these are the other main software components are:

CameraControl (Figure 6) controls the detector. It sends commands to the detector controller cards and receives status and images from them.

MechanismInitialization initializes the mechanisms by initializing the motor controllers and reading the locations of each mechanism from files on the disk. The mechanisms are the filter wheel, pupil wheel, field-mask wheel, f/12 camera mirror, and f/21 camera mirror. The motor controller number and axis number on the controller are shown for each mechanism.

The status for each mechanism indicates the optic that is selected, whether the positioning is correct, and whether the position is known. The position may not be known if the computer crashed while the mechanism was being moved. The position may be incorrect if the mechanism was moved between two optics; *e. g.* between two filter positions.

Also shown are the states of the reverse and forward limit switches. If both limit switches are engaged, which is physically impossible, then the rotation stage is not installed: both switches are not electrically connected to the motor controller and therefore open.

MechanismHoming (Figure 7) locates the reverse limit of a mechanism in order to test or set its home position. Since the mechanisms do not have feedback, the position is inferred from the home position and the number of steps the motor moves.

To find home, the mechanism (1) moves off of the reverse limit, if needed, (2) moves in the reverse direction until the limit switch engages, (3) moves forward slowly to disengage the limit switch, (4) moves slowly in the reverse direction to find the limit switch a second time, (5) moves to +400 steps from the reverse limit, backing into the final position to eliminate backlash. (See Figure 7 for a plot of position *vs*. time during homing.) Moving slowly onto the reverse limit (step 4) minimizes errors such as bouncing and delay between switch closure and sensing it. The **history** indicator shows the success or failure of each step.

To save subsequent plots of the motion to find the reverse limit, press the button **save plot**. The plots are saved in the directory for instrument logs, and the file name is FRL-yyyy-mm-ddThhmmss.png. For example, if the date is 1 Nov 2006, and the time is 23:15:13, then the file name is FRL-2006-11-01T231513.png.

Two parameters for this VI may require changing, if the motor speed is changed. The velocity override is the percent of normal speed at which the mechanism approaches the reverse limit to find it accurately. The timeout is the maximum time to wait for the mechanism to find home for step 1. It is set at 90 s; the time to turn 180,000 steps (360°) at 2600 step/s is 70 s.

MechanismMoving moves the mechanisms.

To save subsequent plots of the motion, press the button **save plot**. The plots are saved in the directory for instrument logs, and the file name file name is Move-yyyy-mm-ddThhmmss.png. For example, if the date is 1 Nov 2006, and the time is 23:15:13, then the file name is Move-2006-11-01T231513.png.

FITSServer writes images on the disk in FITS format.

PGauge reads the Inficon pressure gauge.

LogTempPressure maintains a log of the temperatures at several points in the instrument and the pressure inside the instrument. The **time step** is how often to save

a sample in the log file. You may choose which temperatures to show on the plot. The VI records the temperature and pressure even if they are invalid.

SpartanServer implements the communication model of the observatory control system (OPEX). SpartanServer receives commands from a client and passes them on to StartanGUI. In addition, it sends status to the client. The text-based user interface, SpartanTUI, is such a client.

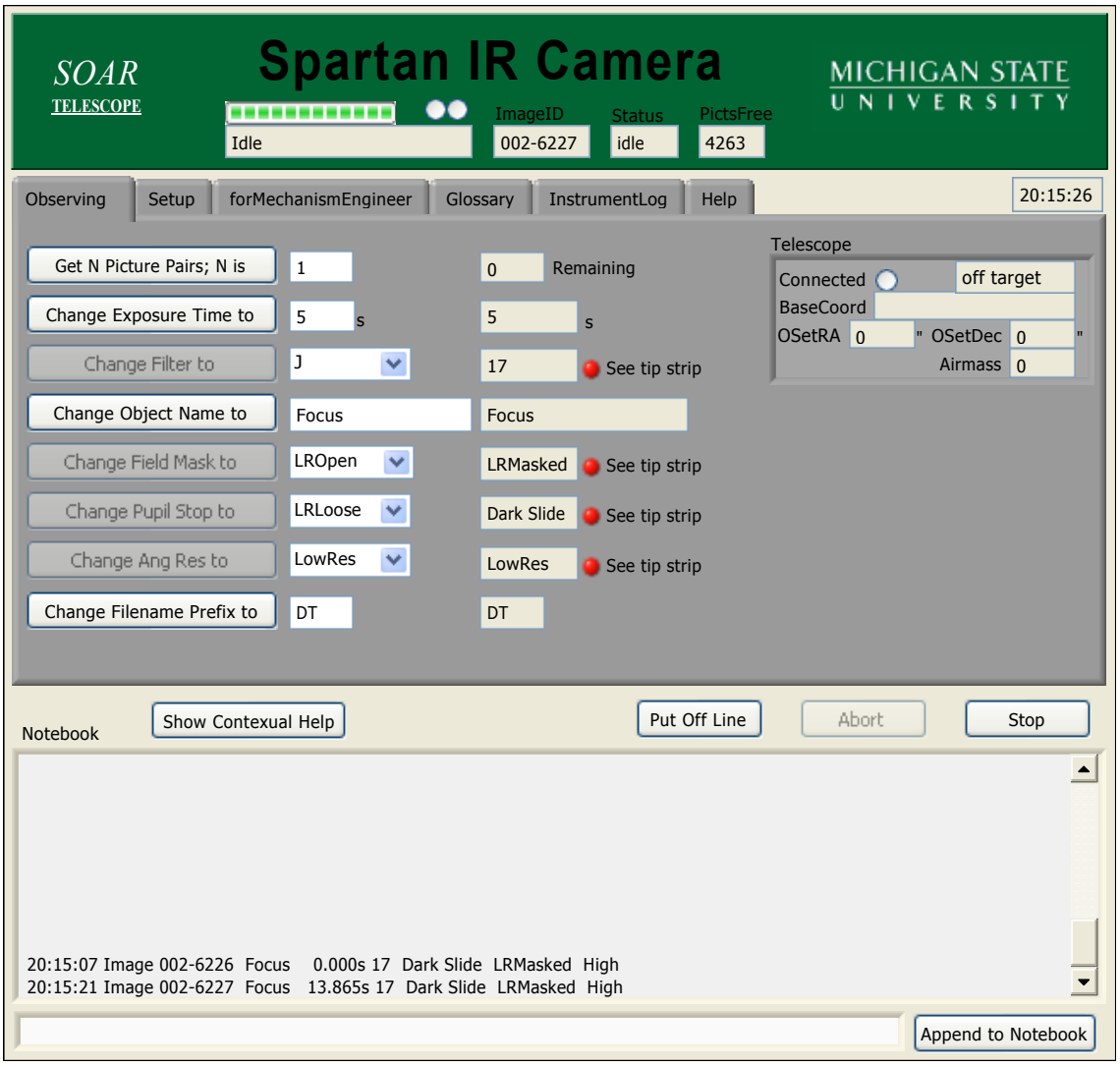

Figure 1: SpartanGUI, the graphical interface of the software, with the observing panel selected.

1 Native Commands

Native commands (Table 1 & Table 2) are defined by the Spartan software itself. The more commonly used native commands can be abbreviated. The shortest possible abbreviations are in the tables.

Commands are not case sensitive.

Many commands require a parameter. For example, to move to a filter, the parameter is the filter name. To move to the J-band filter, type

filter J

To execute a command without a parameter more than once, type

<command> <number of times to execute it>

For example, to get 5 pictures, type

getpicture 5

2 Scripts

You may write scripts, which are combinations of native commands and other scripts. To execute a script, type

<script name> <number of times to execute it>

For example, to execute the script, runHRCollimator.txt, 34 times, type

`runHRCollimator 34`

You may put comments in a script: What follows a semicolon ";" is a comment.

Scripts are text files <scriptName>.txt. The script name must not be the name of a native command or an abbreviation of a native command. To be safe, you may begin the name of the script with the letter "z," which will never be used for native commands.

The software searches for scripts first in **scriptFolders** (on the front panel), in order, starting with folder 0. If not found, then the software searches in **defaultFolder**. If SpartanTUI is running as a stand-alone application, the default folder is `\script` in the folder of SpartanTUI.exe. If Spartan.vi is running, the default folder is `\home Spartan\script`.

Here are two examples of scripts. The script exerciseHRCollimator calls runHRCollimator. The script exerciseHRCollimator.txt:

Figure 2: SpartanTUI, a text-based user interface. The VI accepts primitive commands and scripts, for which the VI searches in "scriptPaths" and if not found there, then in \SpartanHome\Script. You may either run the command (press **Run Command**) or test the command without running it (press **Test Command**).

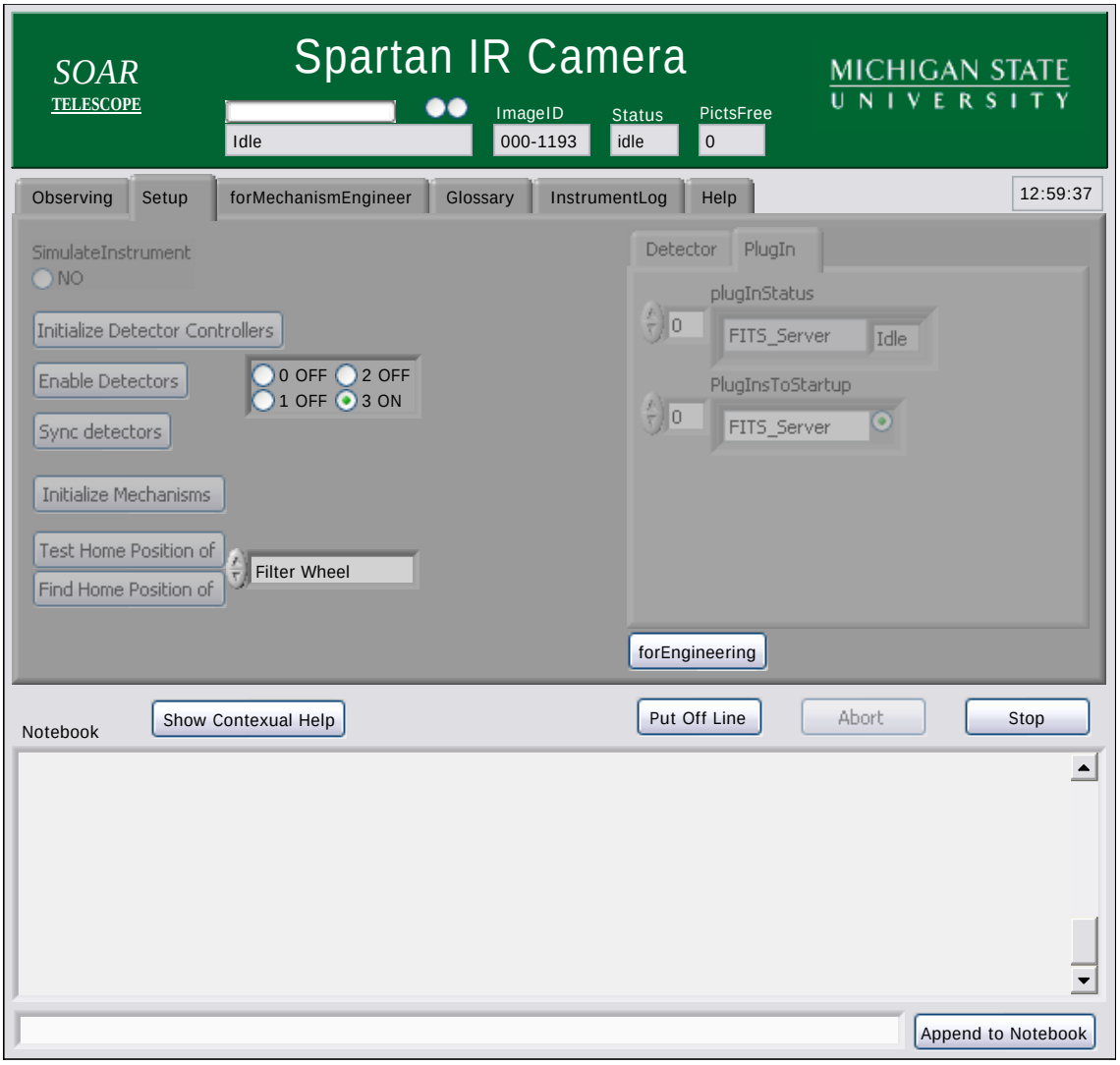

Figure 3: SpartanGUI with the panel **setup** selected.

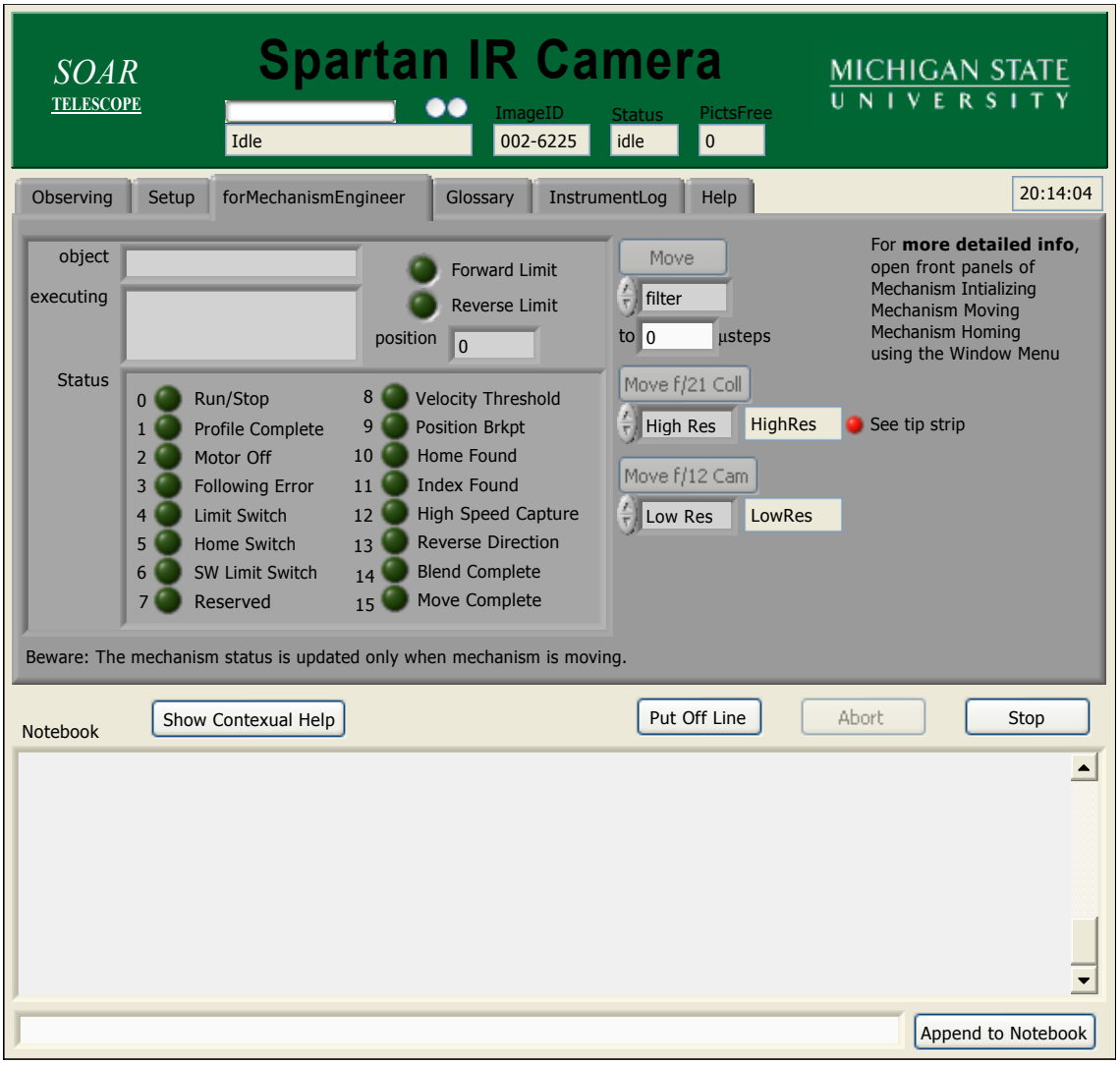

Figure 4: SpartanGUI with the panel **forMechanismEngineer** selected.

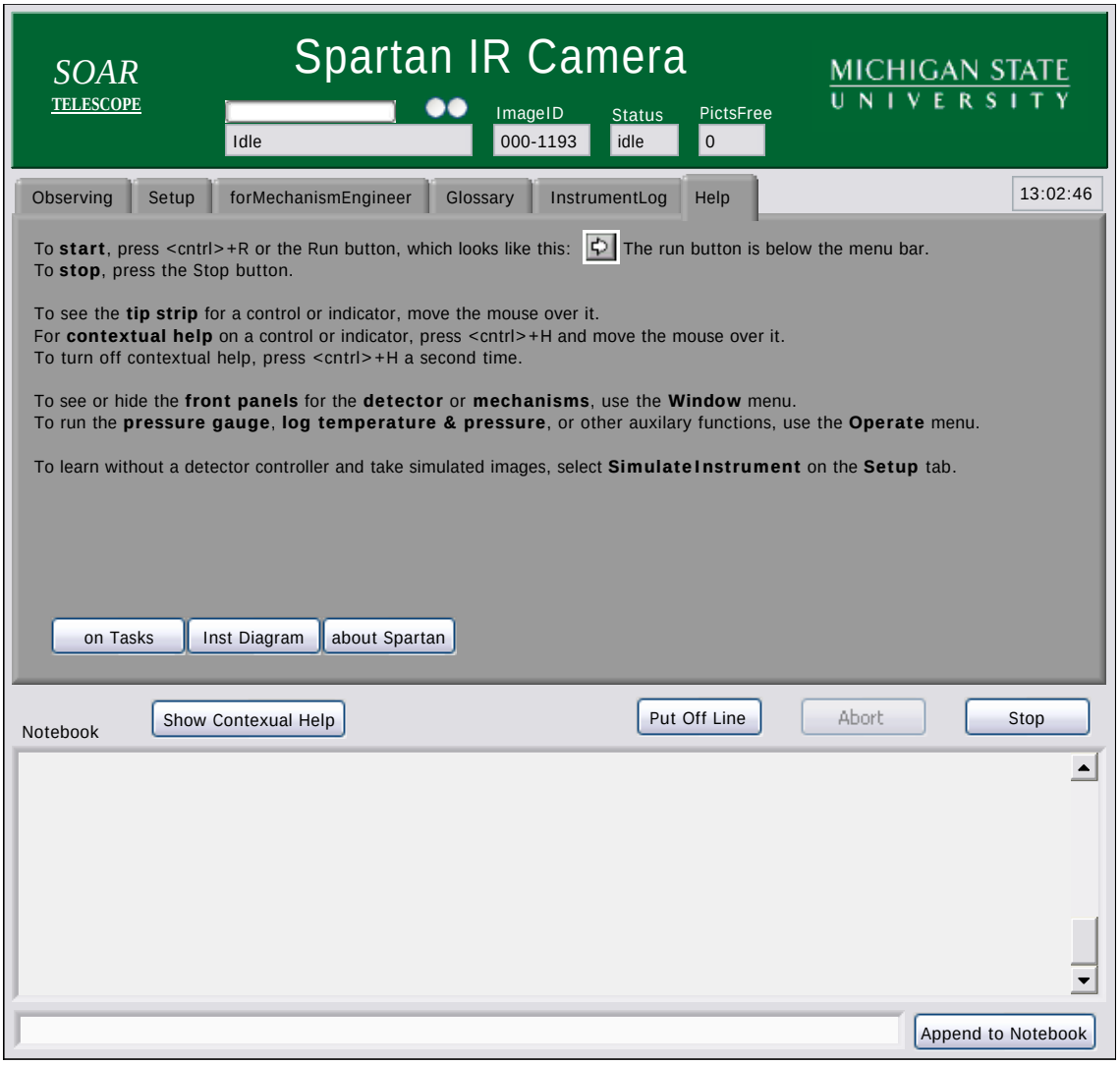

Figure 5: SpartanGUI with the panel **help** selected.

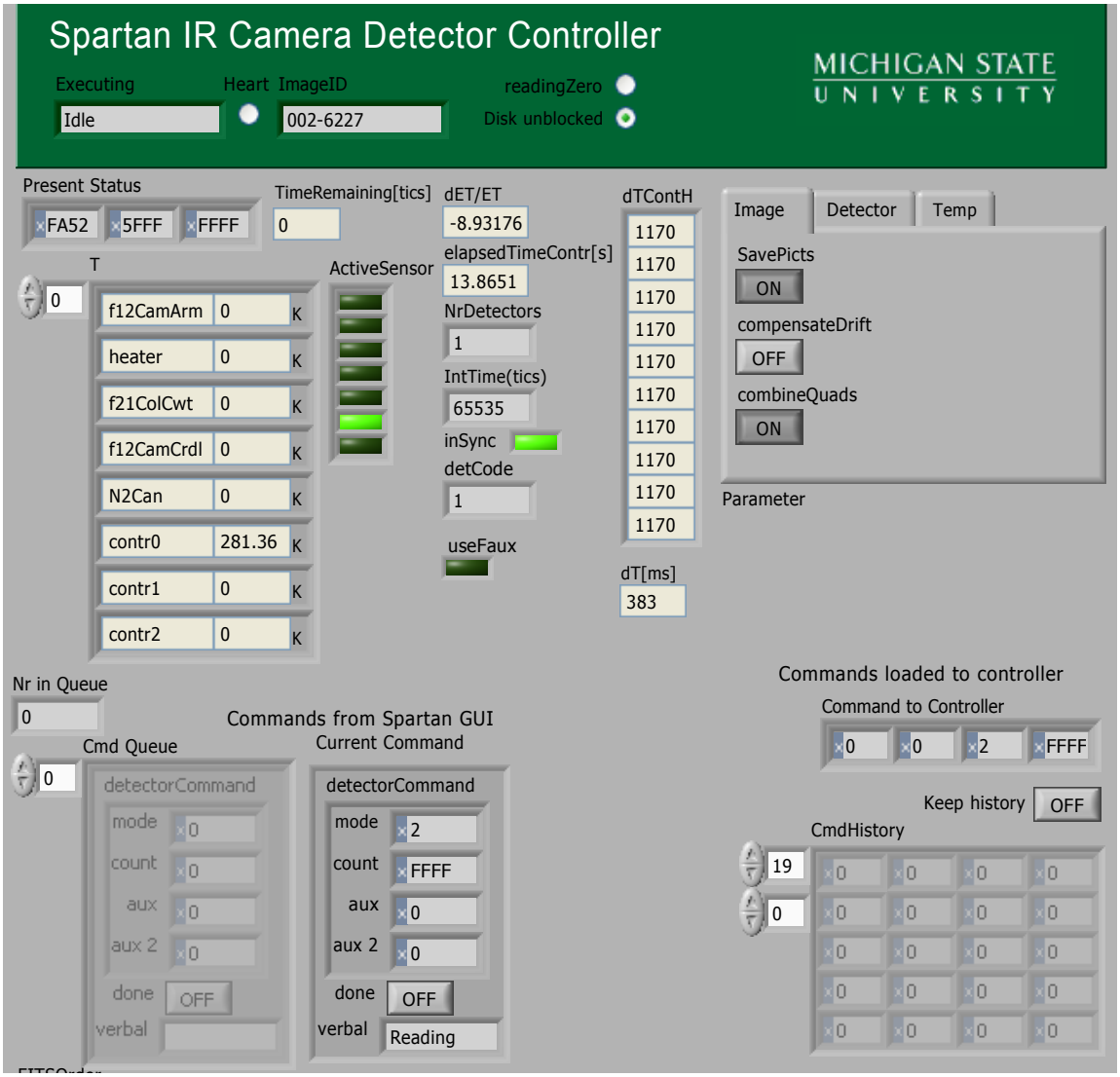

Figure 6: CameraControl, which controls the detectors. Part of the front panel is not shown.

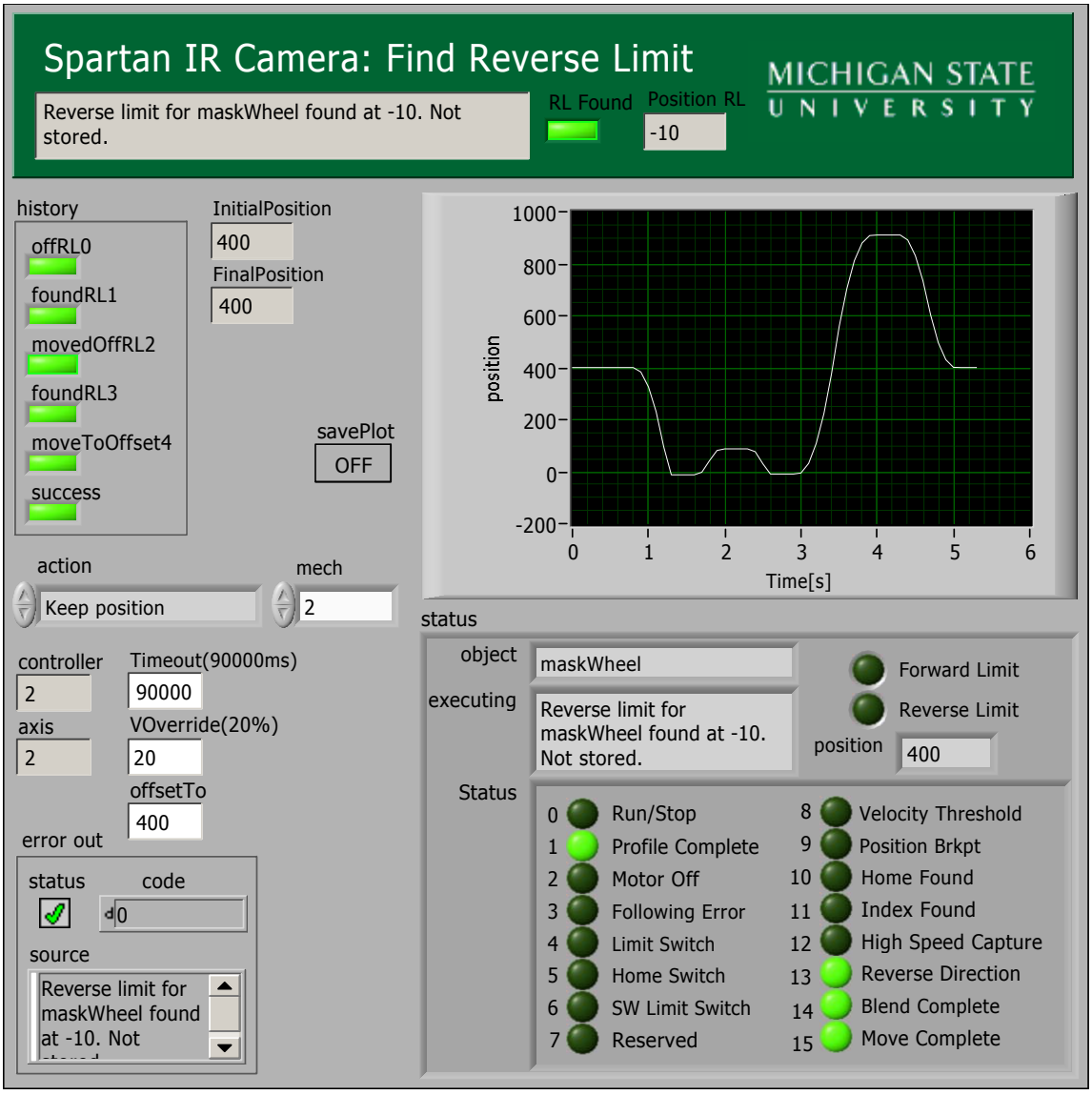

Figure 7: FindRevLimit, which searches for the reverse-limit switch of the mechanisms in order to index the position. The graph shows the position as a function of time. The mechanism moved in the reverse direction until it found the limit switch, moved forward and found the reverse limit a second time, moving more slowly, moved to 400 steps from the reverse limit, backing into the final position to eliminate backlash.

2 Text-based Interface

For running repetitive sequences of tasks, Spartan has a text-based user interface, SpartanTUI, which passes commands to the graphical user interface, SpartanGUI.

2.1 Native Commands

Native commands (Table 1 & Table 1) are defined by the Spartan software itself. The more commonly used native commands can be abbreviated. The shortest possible abbreviations are in the tables.

Commands are not case sensitive.

Many commands require a parameter. For example, to move to a filter, the parameter is the filter name. To move to the J-band filter, type

filter J

To execute a command without a parameter more than once, type

<command> <number of times to execute it>

For example, to get 5 pictures, type

getpicture 5

2.2 Scripts

You may write scripts, which are combinations of native commands and other scripts. To execute a script, type

<script name> <number of times to execute it>

For example, to execute the script, runHRCollimator.txt, 34 times, type

`runHRCollimator 34`

You may put comments in a script: What follows a semicolon ";" is a comment.

Scripts are text files <scriptName>.txt. The script name must not be the name of a native command or an abbreviation of a native command. To be safe, you may begin the name of the script with the letter "z," which will never be used for native commands.

The software searches for scripts first in **scriptFolders** (on the front panel), in order, starting with folder 0. If not found, then the software searches in **defaultFolder**. If SpartanTUI is running as a stand-alone application, the default folder is `\script` in the folder of SpartanTUI.exe. If Spartan.vi is running, the default folder is `\home Spartan\script`.

Here are two examples of scripts. The script exerciseHRCollimator calls runHRCollimator. The script exerciseHRCollimator.txt:

```
;exerciseHRCollimator exercises the high-res collimator
; by testing home before and after 100 movements
testHome 2          ;test home position for motor 2
runHRCollimator 100 ;run script 100 times
testHome 2          ;test home again
```

The script runHRCollimator.txt:

```
;runHRCollimator moves the high-res collimator
; into the optical path and out
HRCollimator 0     ;move into beam
HRCollimator 45000 ;move 90 degrees
wait 60            ;wait 60s to prevent motor from overheating
```

2.3 Testing Scripts

You may test a script for syntax errors before you run it. Type the command for calling the script as you normally do, and then press the button **Test Command**, rather than the button **Run Command**.

Table 1: Commonly used text commands. The shortest possible abbreviation is underlined. Commands are not case sensitive. Some commands set a parameter. To find out the value of the parameter, type the command without a parameter. For example, `time 20` sets the exposure time to 20 s, and `time` asks for the exposure time. The query form of the command exits for commands with a "?" in the second column.

Command	*Q*	*Explanation*
GetPicture		Get picture: Read detector.
Time x	?	Set exposure time to x seconds.
Filter x	?	Move to filter x.
Pupil x	?	Move to pupil stop x.
Mask x	?	Move to field mask x.
Object x	?	Set object name to x.
Prefix x	?	Set filename prefix to x.
Resolution x	?	Change ang-res to x, where x is "high" or "low."
OLog x		Write entry x into the observing log.
ILog x		Write entry x into the instrument log.
Offset o		Offset the telescope by o. The format for the telescope offset is direction (N, S, E, or W), amount, and an optional unit (" or '), where " is the default. Examples: `offset N23.4'E3` offsets the telescope north 23.4 arcmin and east 3 arcsec. `offset S5` offsets the telescope south 5 arcsec.
focus x	?	Move the telescope focus by x μm
defineTargets t	?	Define telescope targets where t is a list, separated by commas, of offsets from the reference. Example: `definetargets N0,N2',S2',E2',W2'` defines a cloverleaf pattern centered on the reference.
defineTargets n t	?	Define telescope targets starting at target n, where t is a list, separated by commas, of offsets from the reference. Numbering of targets starts with zero. If n is negative or larger than the largest existing target number, the new targets are appended to the existing ones.
dither x	?	Set the dithering radius to x arcsec
to n	?	Move the telescope to target n with dithering. Numbering of targets starts with zero.
cleartargets		Clear the target definitions
reference		Define current telescope position as the reference

Table 2: Less commonly used text commands. The shortest possible abbreviation is underlined. Commands are not case sensitive. Some commands set a parameter. To find out the value of the parameter, type the command without a parameter. For example, `time 20` sets the exposure time to 20 s, and `time` asks for the exposure time. The query form of the command exits for commands with a "?" in the second column.

InitDetector		Initialize detector controllers.
InitMechanism		Initialize mechanisms controllers.
Detectors n	?	Enable detectors n, where n is any combination of 1, 2, 3, or 4.
Home n		Move motor n to home, and reset positioning.
TestHome n		Test home position of motor n, but do not reset positioning.
OnLine		Put instrument on line.
OffLine		Put instrument off line.
Status		Query status.
Sync		Synchronize detector controllers.
HRCollimator x		Move high-res collimator to position x.
LRCamera x		Move low-res camera mirror to x.
Wait x		Wait x seconds.

3 Operating the Software with a Simulated Camera

You may operate the software with a simulated camera. For example, you can time exposures and take pictures even if the detector electronics are not installed on the computer. This is useful for learning to use the instrument.

To operate with a simulated camera, open the **Setup** panel in SpartanGUI before you run the software. Make the control **SimulateInstrument** read "Yes." Now run the software. The banner will be "Simulated Spartan Camera," rather than "Spartan IR Camera."

4 Data

4.1 File Tree

The Spartan software must be installed with the file tree in Table 3. The folders for images, observing logs, instrument logs, and volatile data can move; their locations are specified in the configuration file.

4.2 Volatile Data

Volatile data are the mechanism positions and the current image ID. The data are stored in these files, whose path is in the entry `volatileDataPath` in the configuration file `spartan.txt`.

mech0.txt-mech4.txt Before moving mechanism 0, the flag "moving" is written to the file `mech0.txt` to indicate that the mechanism position is changing. After finishing a motion, the flag is erased and the new position is written to the file. Thus the position of the mechanism and whether the position is accurate are both stored on disk to ensure safe recovery from a crash. Each mechanism has its own file.

image ID.txt contains the number of the last image. The image ID is used as part of the name of the image, and it must be unique.

4.3 Software Configuration

These files, which are in `\home Spartan\Configuration`, contain configuration information.

Table 3: File tree for the Spartan software. Folders that may be outside the tree are marked with an asterisk.

- **home Spartan** top directory
  - **app** stand-alone applications
  - **SpartanTUI*** stand-alone SpartanTUI
    - **script** scripts for stand-alone SpartanTUI
  - **common** subVIs
  - **configuration** VIs and .cfg files used for configuration
  - **data*** volatile data
  - **docs** documentation
  - **instLog*** instrument logs
  - **modules** subVIs for packages
    - **cam** subVIs for detector control
    - **cmdlineServer** subVIs for command-line server
    - **FITS** subVIs for the FITS server
    - **mechanisms** subVIs for controlling mechanisms
  - **obsLog*** observing logs
  - **plugIn** plug-ins here are visible to SpartanGUI
    - **FITS** package from SOAR
    - **pressure** pressure gauge
  - **script** scripts
  - **www** html remote panels for web browsers

Table 4: Example configuration. Note: the first slash is translated to a colon; *e. g.*, the entry `C/home Spartan/instLog` translates to the path `C:home Spartan\instLog`.

```
[default]
computerName="unknown"
imagePath="/C/images"
volatileDataPath="/C/data"
observingLogPath="/C/obsLog"
instrumentLogPath="/C/instLog"
[35.10.222.84]
computerName="horolog4"
imagePath="/C/images"
volatileDataPath="/C/home Spartan/data"
observingLogPath="/C/home Spartan/ObsLog"
instrumentLogPath="/C/home Spartan/InstLog"
```

```
[35.9.70.129]
computerName="sextans"
imagePath="/E/images"
volatileDataPath="/C/data"
observingLogPath="/C/obsLog"
instrumentLogPath="/C/instLog"
```

spartan.txt has computer-specific paths. An example is in Table 4. The format is that of Windows configuration files. The line in brackets specifies the IP number of the computer to which the information applies.

soarCommsSpartan.txt contains IP addresses of the text-based server and authorized clients.

defaultOptic.vi (Figure 8) contains the default mechanism positions for the high-res and wide-field observing channels.

detectorSetpoint.vi contains locations and operating voltages of the detectors.

hdr-obs.txt contains the FITS key words that are implemented.

initDatabase.vi contains information about the observatory, telescope, and instrument.

mechanismParameter.vi (Figure 8) contains the operating information for the mechanism, which are the controller and axis number to address the mechanism, the maximum position (counting from 0), the number of steps per position, whether to compensate for backlash, the offset from the reverse limit to the 0-th position, whether the mechanism is a wheel (and therefore the last position is adjacent to the first), and the serial number of the rotation stage. You must access the block diagram to make changes.

SpartanToWheelNames.vi (Figure 8) contains the correspondence between the mechanism positions and names of the optics.

4.4 Queues

Queues are used to signal and to pass information.

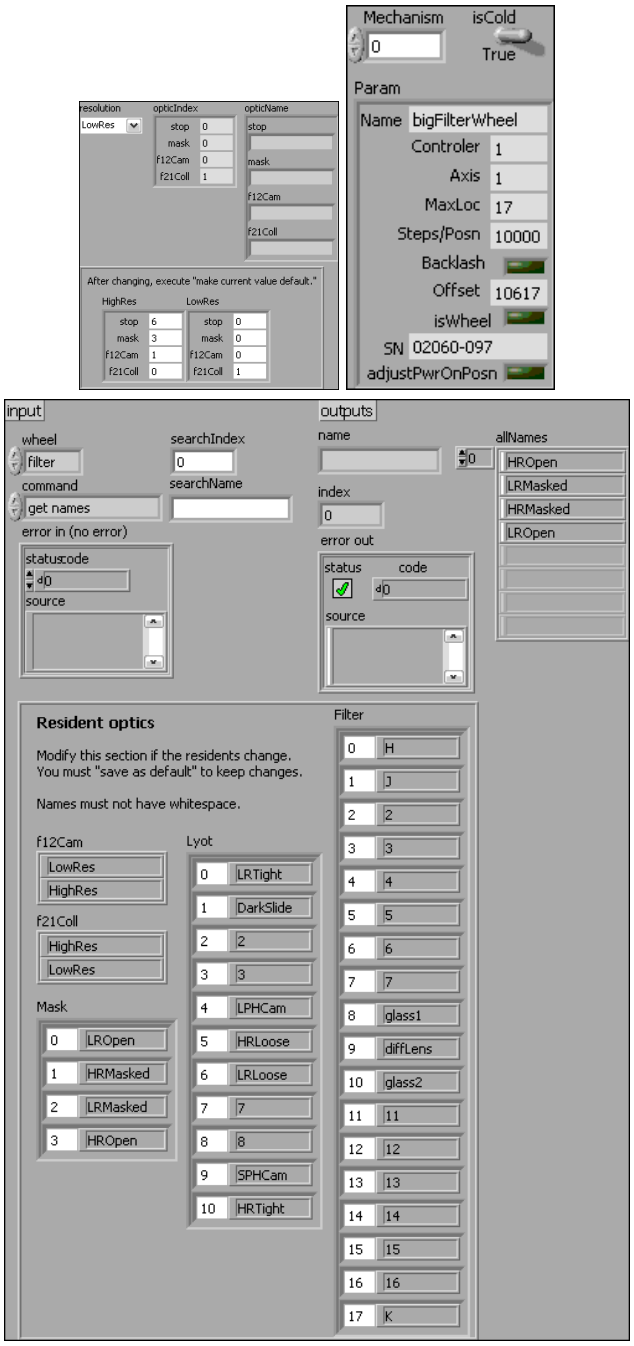

Figure 8: VIs, defaultOptic.vi (top left), mechanismParameter.vi (top right), and SpartanToWheelNames.vi (bottom), for configuring the instrument. The data for mechanismParameter.vi is in the block diagram; an indicator on the front panel is shown.

Table 5: Queues

Name	*Contents*	*Originating VI*	*Terminal VI*
detector	command for detector	SpartanGUI	CameraControl
detectorControl	next operation for CameraControl	CameraControl & SpartanGUI	CameraControl
abortCam	signal to abort operation of CameraControl	SpartanGUI	CameraControl
image	information about image to save	CameraControl	FITSServer
mechanism	command for mechanism	SpartanGUI	MotorControl
abortMech	signal to abort operation of motorControl	SpartanGUI	MotorControl
commandLine	parsed command	SpartanServer	StartanGUI
signal	information for observer	several	StartanGUI
notebook	entry in observer's log	several	StartanGUI
instLog	entry in instrument log	several	StartanGUI
mechStat	detailed information on mechanisms	several	SpartanGUI
lookAtMech	signal to look at status of mechanisms after finishing a move	several	StartanGUI
CloseCommandLine	signal to close SpartanServer	SpartanGUI	SpartanServer

4.5 Instrument Status

The software implements the device model for SOAR communications[2] with one exception, which is described below. The instrument may be in one of four states. The states are these:

Idle The instrument is ready but not performing any operation.

Active The instrument is busy with an operation.

Error An error occurred.

Offline The instrument is not able to accept any commands. This state is used when the operator is adjusting the instrument or fixing a problem.

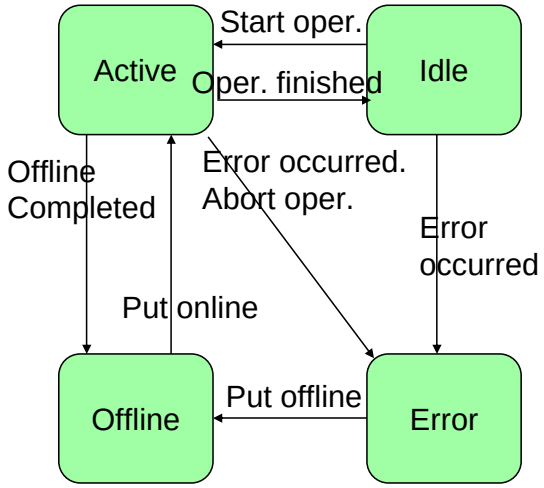

Figure 9: Device model for SOAR communications, modified from Schumacher & Ashe, 2002.

The allowed transitions (Figure 9) between the states are these:

Idle → **active** occurs at the start of an operation.

Idle → **error** occurs with an error.

Active → **idle** occurs at the completion of an operation.

Active → **error** occurs when an operation errs or the operator aborts an operation.

Active → **offline** occurs when the offline operation completes.

Error → **offline** ocurrs at the instignation of the operator.

Offline → **active** occurs at the start of the command to put the instrument online.

With this device model, recovery from error occurs only with the intervention of the operator. The operator must put the instrument offline and then back online to continue. The purpose is to allow the operator the option to fix the cause of the error before continuing. The single exception to the SOAR device model is that the transition from error to active is not allowed.

[2] Schumacher, G., & Ashe, M., 2002, "SOAR TCS: From prototype to implementation," SPIE xxx, xxx.

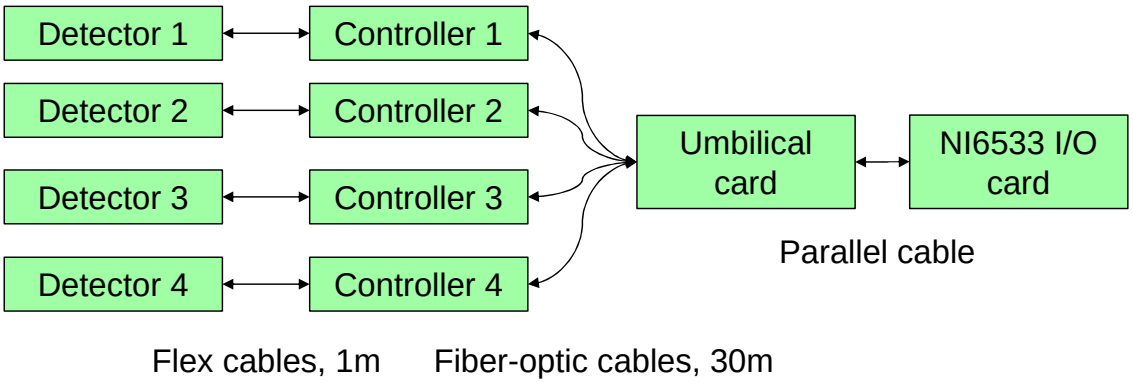

Figure 10: Diagram of the electronics and cabling. The detector cards are inside the instrument. The controller cards are on the outside of the instrument. The umbilical card is near the computer.

5 Detector Controller

The electronics consists of four kinds of cards and cabling to suit the needs. The four detector controllers connect to the computer through the umbilical card and the NI6553 card, a parallel data card from National Instruments. See Figures 10 and 12. The detector controllers are on the outside of the instrument, and they operate at ambient temperature. Each detector controller connects to a detector card inside the instrument through a flexible cable. The flexible cable has a microstrip geometry: traces run over a ground, and the impedance is 61 Ω. Furthermore, it serves as a thermal resistor, since the traces are thin. Four pairs of fiber optics connect the detector controllers and the umbilical card. The umbilical card is near the computer. A parallel cable connects the umbilical cards and the NI6533 card, which is on the PCI bus of the computer.

The umbilical and controller cards are custom. National Instruments provides a driver and a suite of Labview software for its NI6533 card under Windows.

The umbilical card serves two purposes. First, it services four detector controllers with a single port on the computer. This requires interleaving input data and sending duplicate commands to each detector controller. The second purpose is to transport the data to a location remote from the computer. (The requirement here is only 30 m, but the design can handle 2 km.) The umbilical card serializes the data to use a fiber-optic link. In addition it has a first-in-first-out buffer with 4k 2-byte words. At a sample time of 7 μs, the data rate is 2.3 MHz for 4 quadrants on 4 detectors. The FIFO buffers 1.8 ms of data.

The detector controller and computer operate together as shown in the flowchart of Figure 11.

The times at which events in the controller occur are deterministic, but events in the

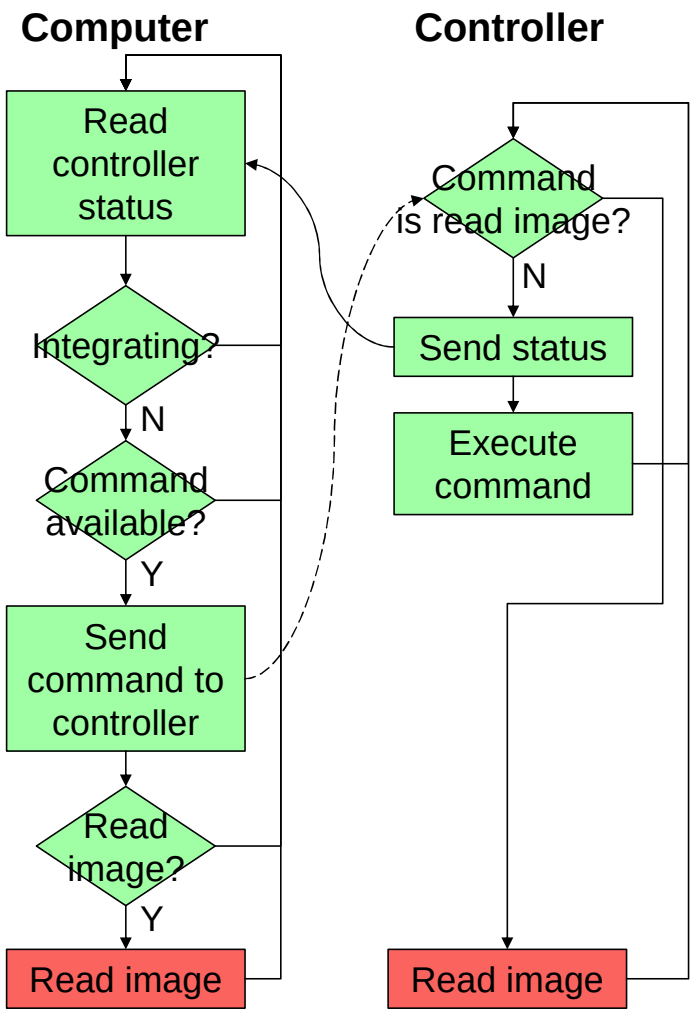

Figure 11: Controller flowchart on the computer side (left) and on the controller side (right). Each cycle through the right flowchart is one frame. Links between events on the controller and computer are shown as curved lines. The dashed curved line is not blocking: if no command is sent, the controller uses a default. The two events “read image” (in red) must occur simultaneously in the sense explained in the text.

computer are not, since the computer performs other tasks besides those in the flowchart. Some tasks are part of Labview, and the priority for these can be controlled. Other tasks such as handling the mouse and running user-initiated processes cannot be controlled by the software.

Because the computer is a fully functioning Windows computer, you must be aware of its limitations: The Spartan software cannot block tasks that are outside LabView. When reading an image, you must not be running other software, such as looking at images, reading your mail, or checking ESPN.com. These tasks can be active, but they must not be using the CPU or the disk.

The detector controller operates in frames. The detector controller executes a new command every frame. If no command if ready, the detector controller executes the command "idle," which makes it do nothing until the next frame. When idle, a frame takes 384 ms. When reading a picture, a frame takes about 7 s, which is the time to read the picture.

The two events "read image" in the controller and "read image" in the computer (Figure 11) must be simultaneous in this manner. The read operation in the computer must start within 1.8 ms of the start of reading the image on the controller side. If not, the FIFO buffer in the umbilical board fills, and data are lost. Once started, the NI6533 card demands time on the PCI bus quickly enough to keep up with the data from the controller card.

The computer and the detector controller must synchronize to a frame. Here we describe the case where data are lost. (1) The detector controller sends the status. (2) The software reads the status. (3) The software sends a command to the detector controller to read an image. (4) The software initiates a read on the NI6533 card. (5) The NI6533 card is ready to accept data. (6) Data arrive. The time between events (1) and (6) is one frame (380 ms). If task (4) is delayed, event (6) can occur before event (5), and data are lost. Event (5) can be delayed if the computer is performing other tasks instead of doing the tasks (2), (3), and (4). We did have problems with synchronization when the disk was faulty and writing an image required CPU usage over a period of 15 second. (Apparently with LabView 7.1 and Windows XP 2002 service pack 2, servicing the disk has higher priority than LabView tasks.)

5.1 Atomic functions

Atomic functions (Table 6) are the simplest functions performed by the detector controller. All actions performed by the detector controller may be broken up into one or more atomic functions.

The detector controller sends status every idle frame. Status is a triplet $(u, t, v +$

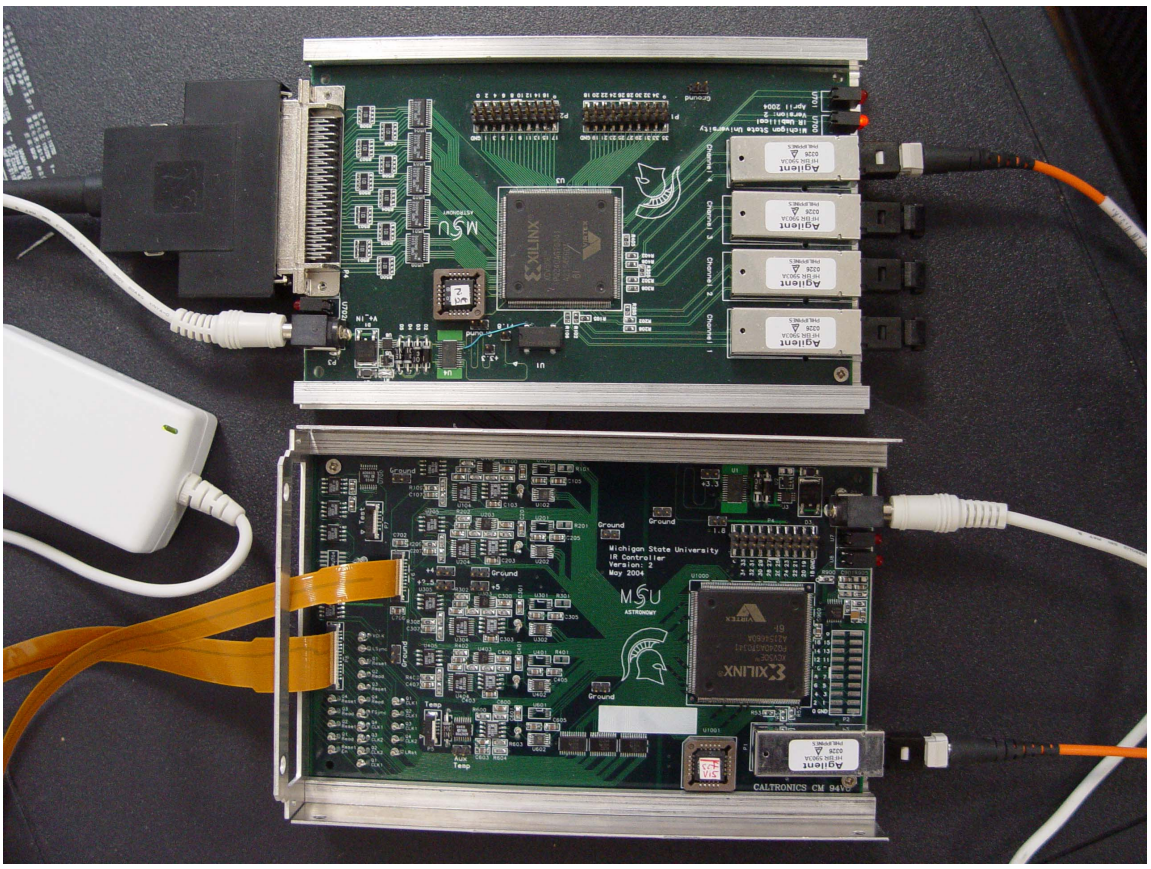

Figure 12: Umbilical card (top) and one detector controller (bottom). The brown flexible cables to the detector run into the bottom left edge of the figure. The black parallel cable to the 6533 card runs out the top left edge of the picture. The fiber-optic link is orange. The cards are 100×160 mm.

$2000_{\mathrm{X}}a)$ of 16-bit words. (The subscript x indicates hexadecimal radix.) In the third word, v is the 13-bit value of temperature sensor a. Whether an exposure is in progress determines the meaning of the u and t. If an exposure is in progress, the 2^7 bit of u indicates whether the exposure completed, $-t$ is the remaining time in 384-ms tics, and $u/100_{\mathrm{X}} + 100_{\mathrm{X}} \mod (u, 4)$ is the time kept by the controller in 327.28-μs tics. If an exposure is not in progress, $t + 10000_{\mathrm{X}}u/8$ is the time kept by the controller in 40-ns tics.

Table 6: Atomic Functions. A command is a set of four 16-bit numbers (m, c, x_0, x_1). A number with the subscript x is hexadecimal; *e. g.*, $100_{\mathrm{X}} = 256$.

Function	*Command*	*Comments*
Read status	Default	
Null	$m = 0$	Do nothing
Flush	$m = 1$	Flush charge
Read picture	$m = 2$, $c = -1$	Read all detectors that are enabled.
Flush & read	$m = 42_{\mathrm{X}}$, $c = -1$	For each row of the picture, flush charge and then read.
Reset	$m = 3$	Reset controller by clearing all counters. This is used to synchronize all detector controllers.
Time exposure	$m = 100_{\mathrm{X}}$, $c = -t$	Time t is in frames of 384 ms. $0 \le t < 16383$.
Load voltage	$m = 800_{\mathrm{X}}$, $c = n + 100_{\mathrm{X}}a$	n is the 8-bit value for address a. $a = 0$ for vReset, 1–4 for vOffset for quadrants 1–4, and 5 for biasGate. For vReset, the voltage $v = 2.500n/256$ V; for the others, $v = 4.096n/256$ V.
Enable detectors	$m = 8000_{\mathrm{X}} + d$	$d = \sum_{i=0}^{3} e_i 2^i$, where $e_i = 1$ if the i-th detector is enabled and $e_i = 0$ otherwise.

5.2 Multiple Detectors

The camera may have one to four detectors, and this section addresses the issue of running multiple detectors.

Each detector controller is connected to the umbilical board by a fiber-optic cable, and communications to that detector is either open or closed. The umbilical board passes commands from the computer to the controller only if the channel is open.

To change the state of the channels, the computer sends the private command `Enable detectors`, which the umbilical board does not pass to the controller boards. A “1” in the most significant bit indicates a private command. The least significant 4 bits code whether the channels are open or closed (Table 6).

The state of the channels remains until the next `Enable detectors` command.

When the umbilical board turns on, channel 3 is open, and the others are closed.

The data from the open channels are interleaved in the order of channel numbering. For example, if channels 0 and 3 are open, the data arrive in this order: first datum from channel 0, first datum from channel 3, second datum from channel 0, second datum from channel 3, etc.

If an open channel malfunctions, the data stream stops.

5.3 Umbilical and NI6533 Input/Output Cards

The link uses “burst mode,” which transfers data on the rising edge of a clock when both the NI6533 and the camera controller are ready. Table 7 lists the signals. The signals req1 and ack1 are active high, which is the default. Only group 1 signals are used. Data on ports A and B are used to form a 16-bit word. The line pclk2, controlled by the computer, indicates the direction of data transfer. The umbilical card drives pclk1 for transfers in both directions, whereas the default is that the NI6533 card drives the clock for transfers to the computer and the umbilical card drives the clock for transfers from the computer. The software controls the direction of the clock.

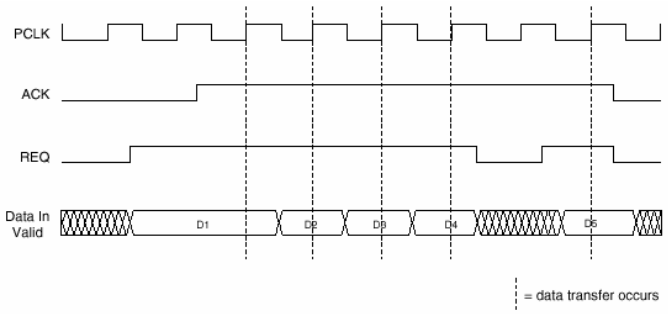

Figure 13: Timing diagram for burst mode from 653X User Manual, National Instruments, Jan 2001, p. 3-6. Data transfer occurs on the rising edge of pckl when both req1 and ack1 are true.

These are the steps that the software takes to read or write data.

Read status or image (1) Set pclk2 to 0 to set the direction of transfer to input. Set direction of the clock to “reverse.” (2) Read data. (3) Set pclk2 to 1 to set the direction of transfer to output and to clear data in the first-in-first-out (FIFO) buffer in the umbilical card. Set the direction of the clock to normal.

Send a command Write the command, which is a cluster of four 16-bit words.

5.4 CameraControl.vi

CameraControl.vi controls the detector. It sends commands to the detector controller cards and receives status and images from them. See Figure 6 for the front panel.

Table 7: Signals on the NI6533 input/output card

Name	*Direction*	*Function*
req1	input	umbilical is ready to transfer data
ack1	output	NI6533 is ready to transfer data
pclk1	input	clock for data transfer
pclk2	output	controls direction of data transfer
dioa	both	least significant byte of data
diob	both	most significant byte of data

The block diagrams in Figures 14–19 show the decisions and branch points. Refer to the descriptions of the VIs in the Appendix. The block diagrams are readable if magnified. For each case structure, one case is shown with the structure. You can find the other cases by searching for boxes of the same size. For some case structures, the trivial cases are not shown. For example, if the error case is to do nothing, it is not shown.

CameraControl.vi has two main parts. An event structure monitors the front panel. A state machine controls the detector. Normally a state determines the next state by queueing the next state in the queue detectorControl. At the completion of a state, execution stalls until the next state is dequeued from detectorControl. StartanGUI.vi can control execution by queueing a state.

These are the actions of the state machine:

Initialize the VI. See Figure 14.

Read Status reads the status, the queue abortCam, and the queue detector, and decides on the next state, either Read Status or Send Command. See Figure 15.

Read Picture reads a picture by calling readPictStep1.vi and readPictStep2.vi and then gathering information for the FITS header. If the picture is to be saved, it puts an entry in the image queue. See the block diagram of this state in Figure 16.

ReadPictStep1.vi, which runs at time-critical priority, performs time-critical tasks to start reading a picture. (See the block diagram of readPictStep1.vi in the top panel of Figure 17.) ReadPictStep1.vi must have set up the NI6533 card for input, allocated the input buffer, and started the data transfer before the buffer on the

umbilical card overflows, which takes 18 ms. For this reason, as much of the initiation as possible is put into this VI. When ReadPictStep1 is finished, data transfer is under control of the NI6533 card, which can control the PCI bus, and the latency between the arrival of data on the umbilical card and transfer into the computer is less than a μs.

ReadPictStep2.vi (bottom panel of Figure 17) transfers data to separate buffers for each quadrant and detector. Because there are no time-critical tasks, it runs at normal priority.

Send Command If the command is not "read image," then the command is sent to the detector controllers, and the next state is Read Status. Otherwise, the next state is Read Picture. The flag "isTiming" is set if the command is to time an exposure. See Figure 18.

Quit releases several queues and signals the front panel loop to quit. See Figure 19.

No Action does nothing.

5.4.1 Memory Usage

LabView is a flow-control language, and it makes a new copy of a data set when it changes, is accessed, or reformatted. In a text-based language, the programmer reuses the memory because it is explicitly allocated. Since the programmer has little control over memory allocation in LabView, the memory allocation can easily grow beyond the size of the computer memory. With our first attempt at writing CameraControl.vi, which uses the most straightforward way of saving the image, 8 copies of the image were made.

Handling the data in "chunks" saves memory space.[3] Rather than handle the entire image, the image is saved in a buffer in the VI GLV_I2Buffer, and chunks of it are retrieved or stored in the buffer.

Reading the image, separating the data for quadrants and detectors from the data stream, and saving the image on disk (with FITSServer)—all use GLV_I2Buffer.vi to store and retrieve the data in chunks from a single copy of the image.

Two copies of the image are needed to implement correlated-double sampling. The picture with no light is saved in memory and the difference between it and a picture with light is computed and then saved on disk.

[3] The note "Managing Large Data Sets in LabVIEW" in the Developer Zone on www.ni.com has a library GigaLabView.llb. We use GLV_WaveformBuffer.vi from this library modified to handle 2-byte data and renamed GLV_I2Buffer.vi.

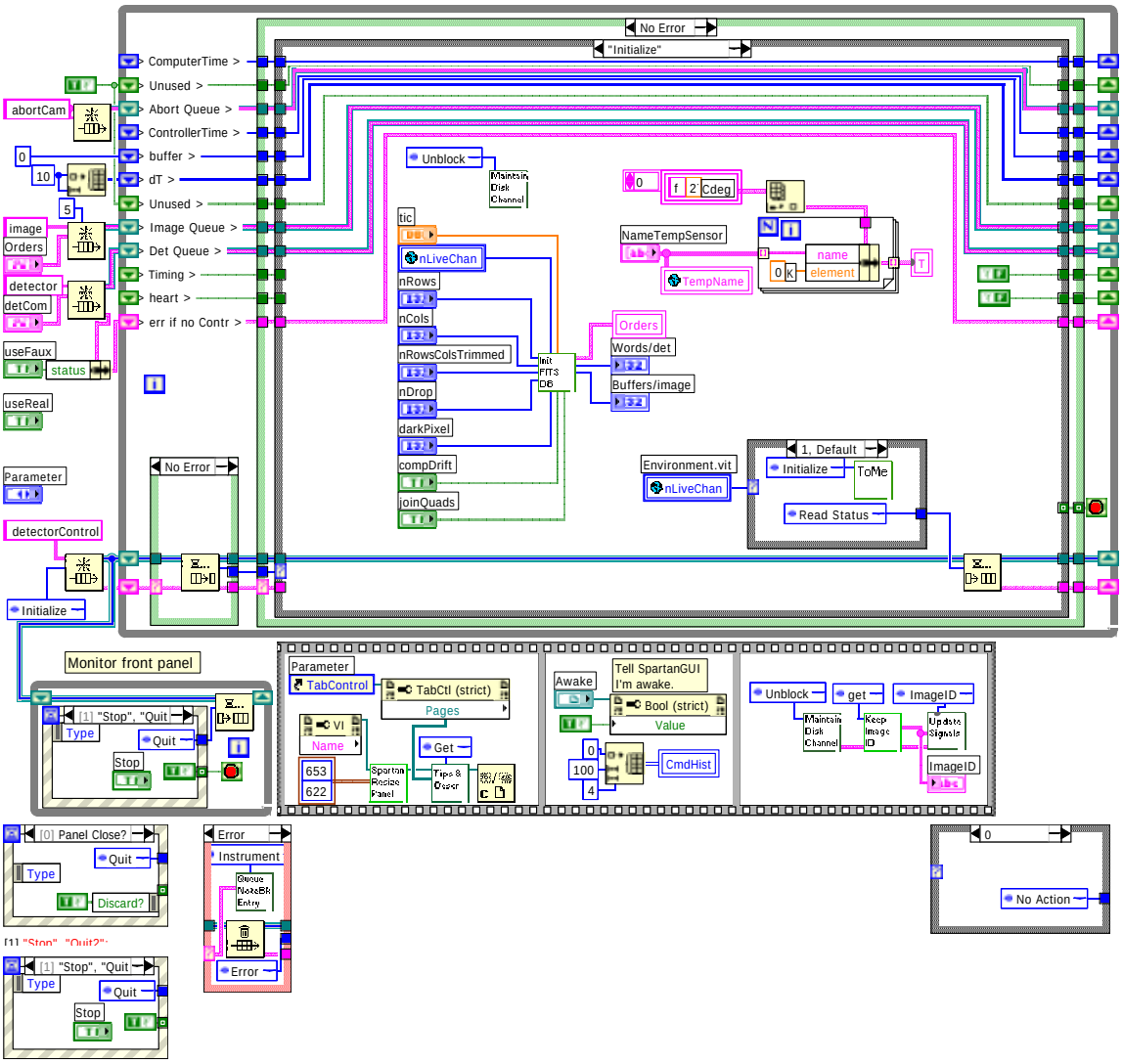

Figure 14: Block diagram in CameraControl.vi for the state Initialize. The VI unblocks disk access, loads the database with information about the image and how it is to be stored, loads labels for the temperature sensors. If there are any channels with live detector controllers, then the VI initializes the NI6533 data port and executes the state Read Status. If no channels are live, the VI executes the state No Action. The event structure for monitoring the front panel is the box with striped borders at the lower left. The other events in the structure are the other boxes with striped borders.

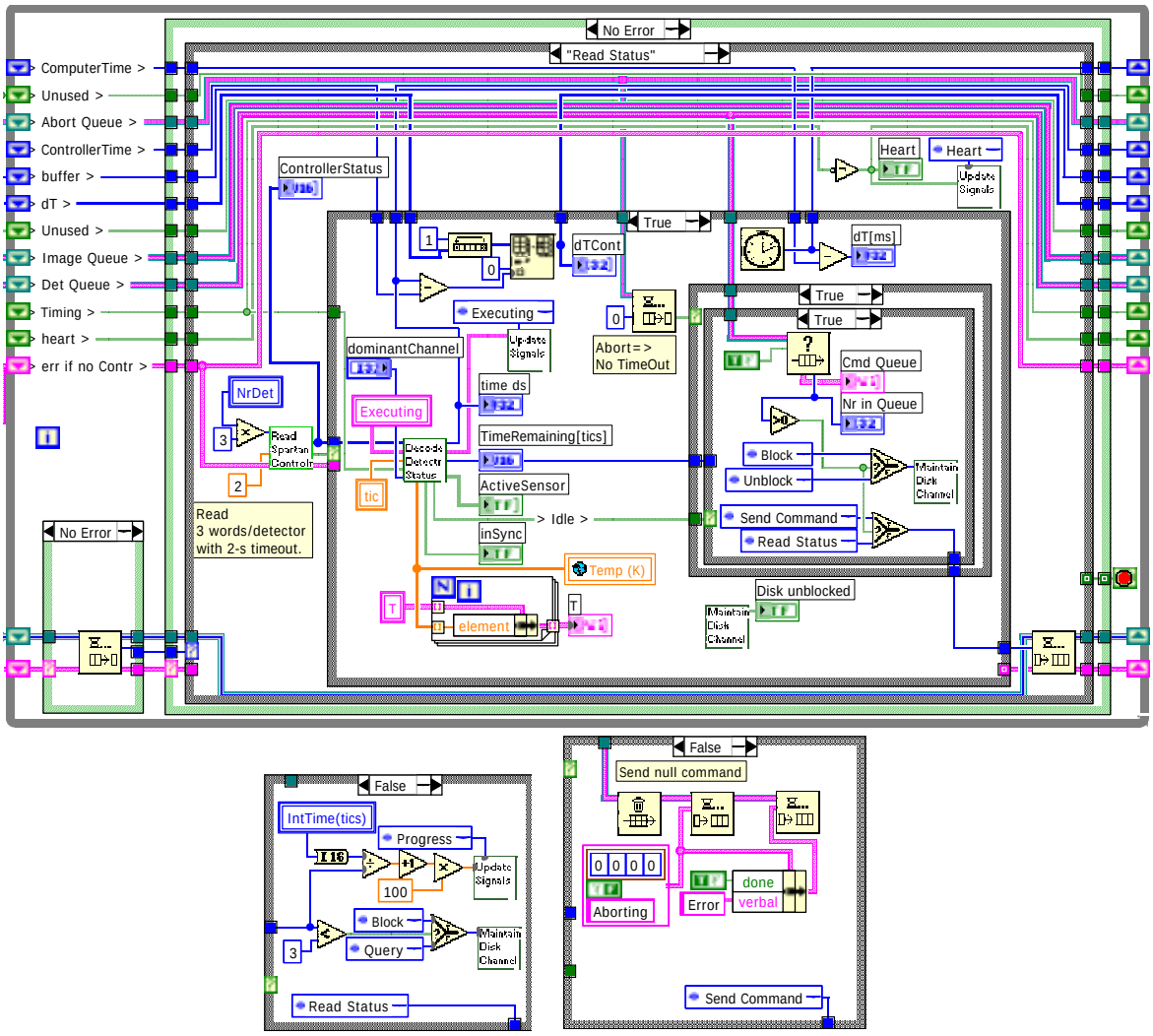

Figure 15: Block diagram in CameraControl.vi for the state Read Status. If dequeuing an element from the queue abortCam does not time out, which indicates the user wants to abort the current command, (lower right pane), the remaining elements on the queue named detector are flushed, a null command is queued, and an error is queued. If there is no request to abort and the controller is timing an exposure (lower left pane), the VI blocks disk access if the exposure is almost complete. If there is no request to abort and the controller is idle (pane inside the main part of the block diagram), the VI reads status again and unblocks the disk if no commands are in the detector queue. If commands remain in the queue detector, then the VI blocks the disk and executes the state Send Command next.

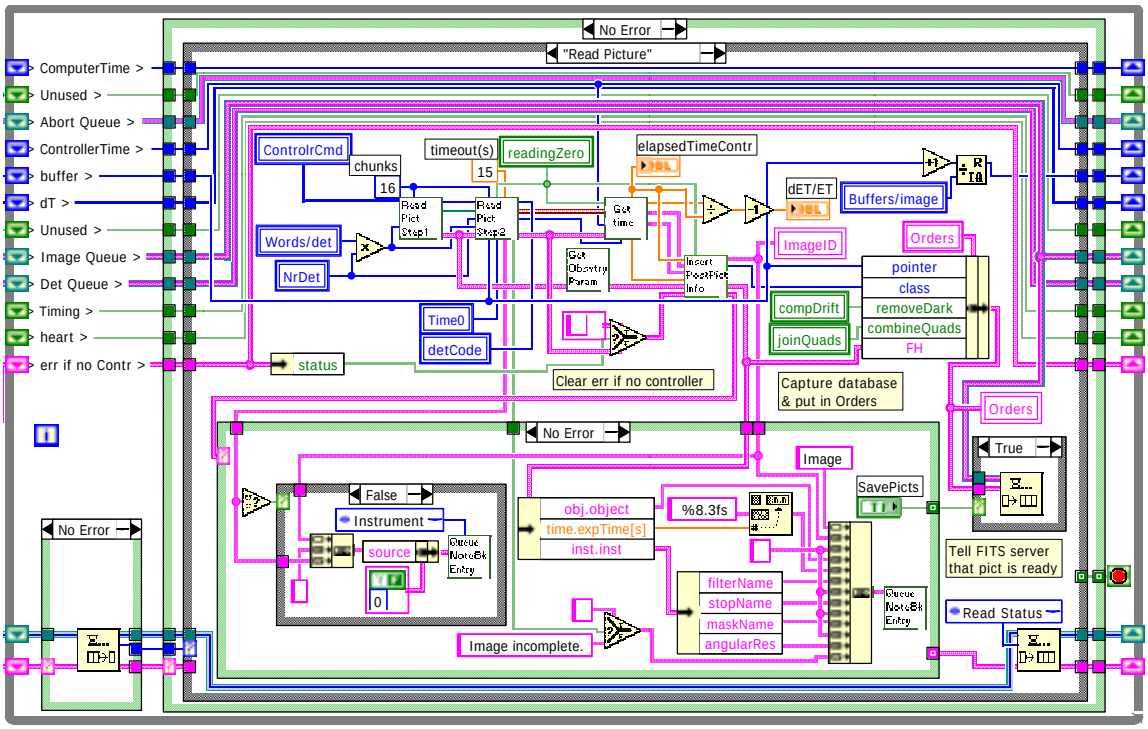

Figure 16: Block diagram in CameraControl.vi for the state Read Picture. ReadPictStep1.vi and readPictStep2.vi transfer data into buffers. After the time-critical readPictStep1.vi completes, GetObservatoryParameters.vi queries the observatory for telescope and weather information to put into the Spartan database. After readPictStep2.vi completes, GetTime.vi produces the time elapsed between the start of reading of the zero-light and nonzero-light images and converts the time of the start of reading into local time and UTC. All of the information is captured and put into the queue of orders for the FITS Server. The next state is Read Status.

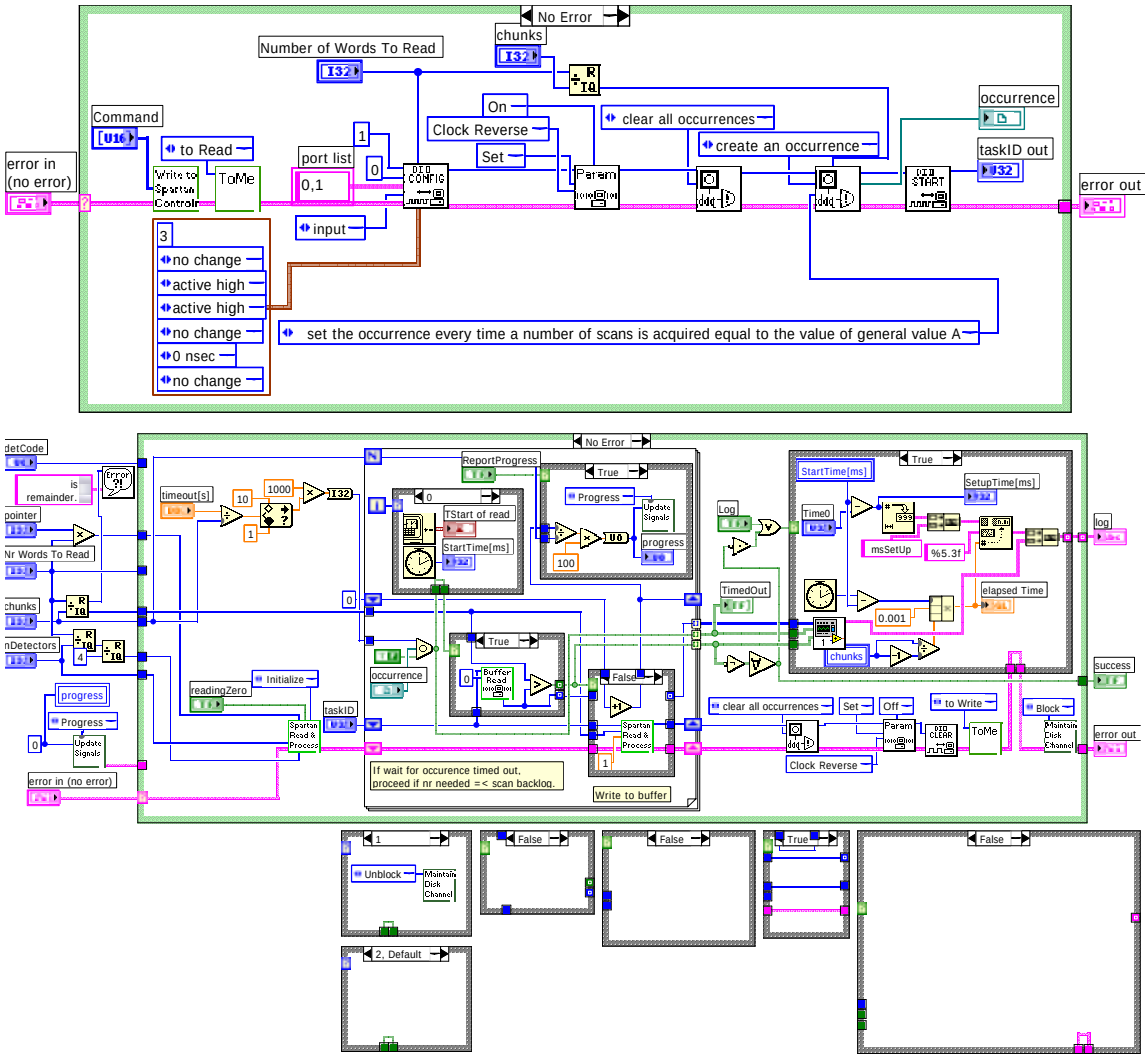

Figure 17: Block diagrams of readPictStep1 (upper) and readPictStep2 (lower). ReadPictStep1 sends a read command to the controller, configures the input/output port and the clock between the NI6533 card and the umbilical card, creates an “occurance,” which synchronizes the software and completion of reading a chunk, which is 1/32nd of the image, and finally starts the transfer of data into the computer. ReadPictStep2 waits for the occurance that a chunk has been transferred. If the occurance did not time out, ReadPictStep2 calls ReadNProcessChunk to save the data stream into separate buffers for each quadrant of each detector. When the first chunk has been transferred, the computer time is saved as the starting time of reading. When the second chunk has been transferred, the disk is unlocked, which allows the FITS Server to use the disk.

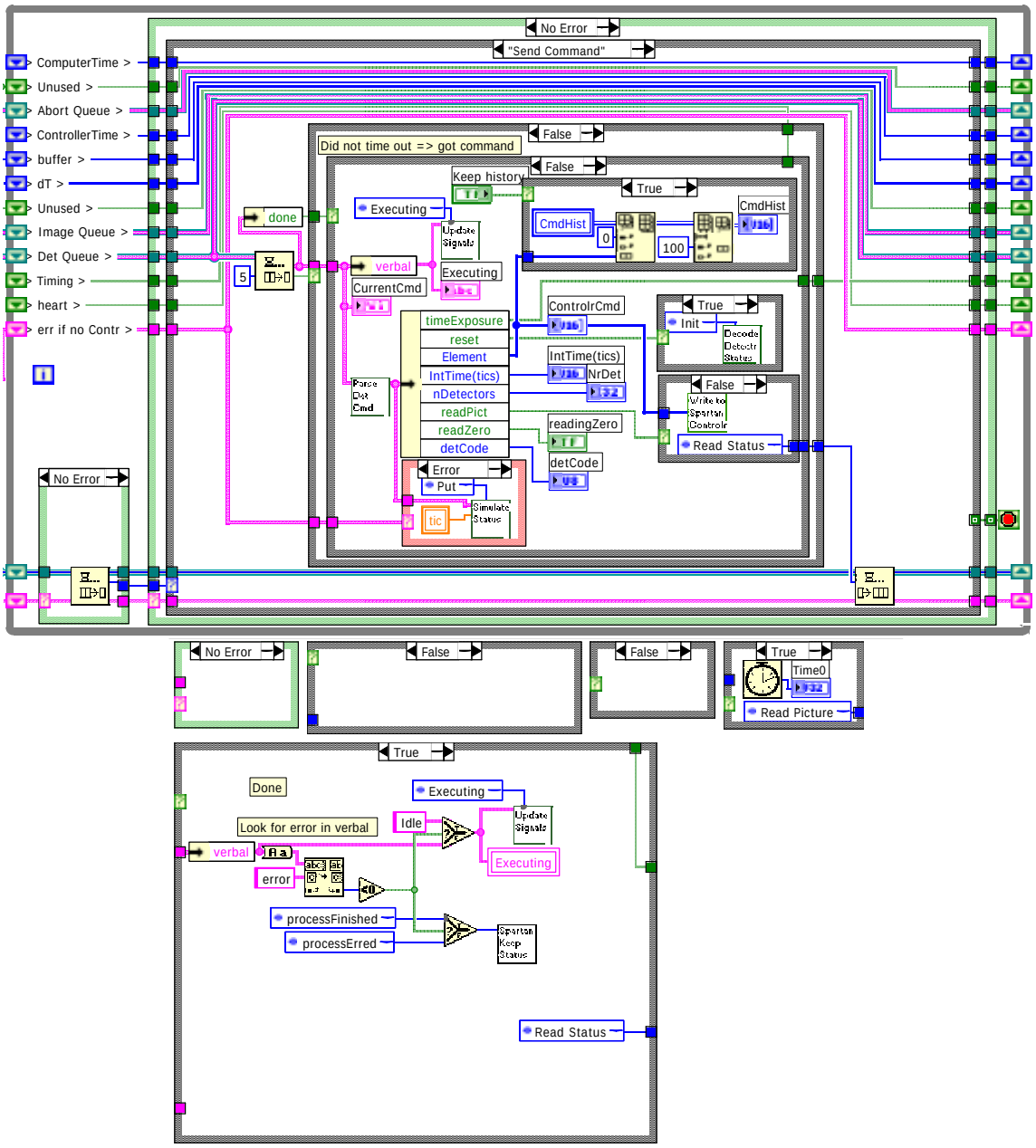

Figure 18: Block diagram in CameraControl.vi for the state Send Command. If the command is "done," the VI passes either "processFinished" or "processErred" to keepStatus.vi, and the next state is Read Status. If the command is not "read picture," the VI passes control (middle right pane) to the state Read Picture. Otherwise the VI writes the command to the detector controller (in the main panel) and then passes control to the state Read Status.

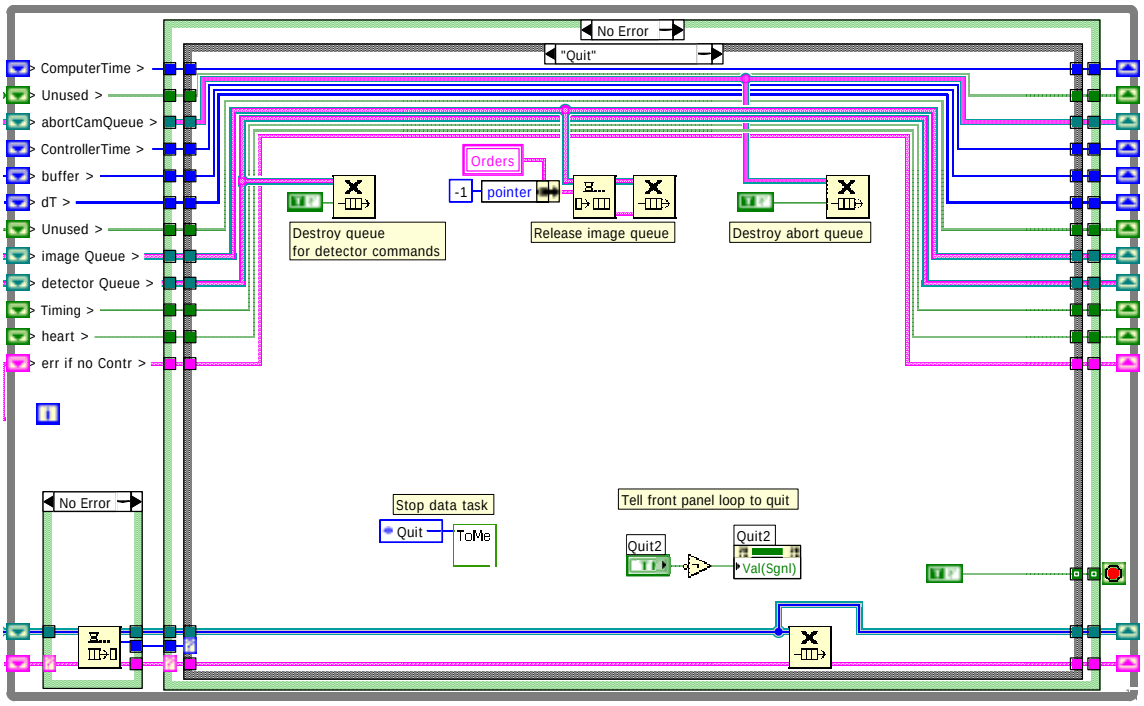

Figure 19: Block diagram in CameraControl.vi for the state Quit. The VI destroys the queues detector and abortCam. It queues a quit message to the image queue, which causes FITS Server to stop, and releases the queue. The VI stops the NI6533 data channel, and stops the state machine and event structure.

5.4.2 Controlling CPU Usage

LabView controls execution of its tasks according to priority, and it maintains several threads of tasks to make switching between tasks efficient[4].

We found that these two devices, execution priority and execution threads, are not sufficient to insure the images are read without losing data. See the beginning of the section for a discussion about synchronizing the detector controller and software. There are two cases where data are lost.

Writing the images on disk uses the C package CFITSIO library[5] and the LabView interface[6] to that package. Calling a C subroutine takes execution outside the control of LabView. Therefore, writing an image to disk completes even though its priority may be low.

We control this problem by blocking the FITSServer at critical times. We use the idea of a channel that is either open or closed. The FITSServer writes data using the channel. CameraControl closes the channel at the time when reading an image starts, namely when the integration has fewer than three tics remaining, and opens the channel after the first chunk is read. Starting the read requires the software to execute, but after starting, it proceeds on the PCI bus, which does not require software intervention.

The second problem is that LabView has no control over tasks that are outside of it. The operator can start a web browser or view an image. Those tasks operate under Windows, and Windows has no mechanism to assign priority according to a directive from LabView. We have no way of solving this other than to advise the operator not to run other tasks when an image is about to be read, which a short remaining integration time indicates.

Checking whether input completed using the VI DigitalBufferRead from the NI data acquisition library uses all of the CPU and locks out other tasks. For example, starting Windows Explorer takes minutes. Instead we use an “occurence,” a LabView concept. The occurence triggers when a certain amount of data (of order 256k pixels) is written in the input buffer. When reading data, the VI readPictStep2 is made to wait on the occurence, which has the effect of lowering its priority and not demanding CPU time. When the occurence triggers, DigitalBufferRead.vi reads the buffer, which takes a short time, because the data are available. Then readPictStep2 is made to wait on the next occurence for the next block of data.

[4] Dorst, N., 2000, “Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability, Application note 114, www.ni.com

[5] http://heasarc.nasa.gov/fitsio/fitsio.html

[6] SOAR 2002, FITS module version 1.14, private communication, G. Schumacher

6 Graphical User Interface

SpartanGUI.vi is the interface for the observer and the technician. See § 1 for a brief description of its function and Figure 1 for a picture of the observing panel.

SpartanGUI.vi has several major sections, which operate in parallel.

Main event structure monitors several types of events:

Button pressed by the observer or by the handler for the command line interface. The observer interacts with SpartanGUI by pressing buttons. Examples of these are take picture, change exposure time, and change filter. There are about 25 buttons.

Panel close or exit, which means the user pressed the close button, <alt>+F4, or the exit button on the menu. This stops the software.

Menu selected The software supports three menus.

Operate The selected plug-in starts.

Browse The front panel of the selected VI changes between visible and not visible.

Help A type of help opens.

Timeout maintains a clock on the front panel. The clock updates every timeout, which occurs once a second.

Stop pressed closes down CameraControl.vi, FITSServer.vi, SpartanServer.vi, and stops SpartanGUI.vi.

Handler for command line interface attends the queue commandLine. When a command appears, it presses a button as if the observer did.

Notebook handler attends the queue notebook. When an entry appears, it writes the entry on disk and presents it in a list box on the front panel.

Handler for Instrument Log attends the queue instLog. When an entry appears, it writes the entry on disk and presents it in a list box on the panel InstrumentLog.

Update signals attend the queue signal. When a signal appears, it writes the signal on the front panel. Signals are the execution message, the progress bar, the heartbeat of the detector controller, the image ID, and status.

Mechanism monitor presents status on the panel forMechanismEngineer while the mechanism is moving. It attends the queue mechStat.

Advise on mechanisms reads the mechanism status and presents advice. It attends the queue lookAtMech.

7 Communicating with the Observatory

The interface to the SOAR Telescope is the Observation Planning and Execution software (OPEX). OPEX moves the telescope and sequences observations. Communication with OPEX uses the SOAR Communications Language.

OPEX is a client, and SpartanServer is a server. OPEX sends a command to the server, and SpartanServer responds promptly. If SpartanServer does not respond, OPEX assumes the communications link is not functioning. SpartanServer does not initiate communications with OPEX.

The commands (Table 8) fall in three classes: (1) a request to perform an action, (2) a query about the status of the instrument, and (3) a query about an instrument parameter.

For each state of the instrument, only certain commands are allowed. For example, moving the filter wheel is not allowed if the instrument is taking a picture. (See the column labelled “Legal States” in Table 8.)

SpartanServer handles commands from OPEX in this way for each class of command:

Perform an action If the command is acceptable, then the SpartanServer (1) responds “Busy” to the OPEX, (2) changes the state to “active,” and passes the command to CameraGUI. If the command is erroneous or the command is not compatible with the state of the instrument, then the server sends the response “Error: unable to . . . while . . . ” to OPEX.

Query status The possible responses are “done,” “busy,” “offline,” or “error.”

Query an instrument parameter The response is “done xxx,” where xxx is the instrument parameter. This may be done when the instrument is performing an action. It may not be done when the instrument is offline.

Commands may be abreviated (Table 8). Commands are not case sensitive.

Table 8: Commands for use with OPEX. The command may be abbreviated to the underlined part of the command. If the state of the camera is not one of the "legal states," the command is rejected. States are active (a), error (e), idle (i), and offline (o).

Command	*Meaning*	*Legal State*	*Responses*
GetPicture	Get picture: Read detector.	i	Busy, Error
Time x	Set exposure time to x seconds.	i	Busy, Error
Time	Query the exposure time.	aei	Done <x>, Error
Filter x	Move to filter x.	i	Busy, Error
Filter	Query filter name.	aei	Done <x>, Error
Pupil x	Move to pupil stop x.	i	Busy, Error
Pupil	Query pupil stop name.	aei	Done <x>, Error
Mask x	Move to field mask x.	i	Busy, Error
Mask	Query field mask.	aei	Done <x>, Error
Object x	Set object name to x.	i	Busy, Error
Object	Query object name.	aei	Done <x>, Error
Prefix x	Set filename prefix to x.	i	Busy, Error
Prefix	Query filename prefix.	aei	Done <x>, Error
Resolution x	Change ang-res to x, where x is "high" or "low."	i	Busy, Error
Resolution	Query angular resolution.	aei	Done <x>, Error
OLog x	Write entry x into the observing log.	i	Busy, Error
ILog x	Write entry x into the instrument log.	i	Busy, Error
InitDetector	Initialize detector controllers.	i	Busy, Error
InitMechanism	Initialize mechanisms controllers.	i	Busy, Error
Detectors n	Enable detectors n, where n is any combination of 1, 2, 3, or 4.	i	Busy, Error
Detectors	Query which detectors are enabled.	aei	Done <n>, Error
Home n	Move motor n to home, and reset positioning.	i	Busy, Error
TestHome n	Test home position of motor n, but do not reset positioning.	i	Busy, Error
OnLine	Put instrument on line.	o	Busy, Error
OffLine	Put instrument off line.	ei	Busy, Error
Status	Query status.	aeio	Done, Busy, Offline, Error
Sync	Synchronize detector controllers.	i	Busy, Error
HRCollimator x	Move high-res collimator to position x.	i	Busy, Error
LRCamera x	Move low-res camera mirror to x.	i	Busy, Error
Wait x	Wait x seconds.	i	Busy, Error

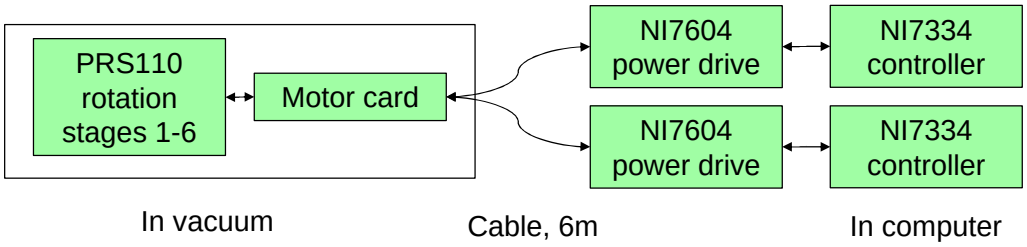

Figure 20: Block diagram of the electronics for the rotation stages. The motor card is inside the instrument. The controller cards are in the computer.

8 Mechanisms

The mechanisms are driven by Phytron/Micos PRS110 rotation stages. The electronics are two National Instruments NI7334 motion controllers, one NI7604 motor drive, one Prismatics MDM2200 motor drive, and a custom motor card (Figure 20). The motor card acts as a thermal resistor between the rotation stages at 77 K and the vacuum bulkhead at ambient temperature, because the traces on it are thin. National Instruments provides a driver and a suite of Labview software for the motor drive and controller under Windows.

The motor software is straightforward, since the data rate is very low.

Power to the rotation stages is shut off when they are not moving in order to reduce the heat load, which is 6 W were all 6 motors in the reduced current state.

When power is shut off, the rotation stage moves to a detent position, which is a multiple of 10 microsteps. When power is turned back on, the NI7604 card powers the motor to a multiple of 40 microsteps, regardless of the position of the motor at the time the power is shut off. One full cycle of the motor is 40 microsteps.

With the NI 7604 motor driver, the motor phase is always the same when the power is turned on, regardless of the phase when the power is turned off. This means that the position is a multiple of 40 when the power is turned on. There is a range of positions n, where $15 < \mod(n, 40) < 25$, that causes the motor to shift unpredictably at power up. An example illustrates the problem. Suppose the motor is shut off at $n = 18$. The motor moves to the detent position 20. (This is predictable.) At power up, the power is turned on to a multiple of 40. The motor may move to 0 or to 40. We have found that one motor moves in one direction at some positions and in the other direction at other positions.

To account for this behavior at power up, the motor is moved to the position n nearest the intended position, where n is a multiple of 10 and $\mod(n, 40) \neq 20$ before power is turned off.

With the Prismatics MD2200 motor driver, the phase is preserved when the motor is

powered down.

The rotation stage is at most 10 microsteps from the intended position, which translates to 0.35 mrad or 1.2 arcmin.

9 Logging Temperature and Pressure

The VI LogTempPressure logs the temperature and pressure periodically. The format of the output file is tab-separated, which Excel can read.

The VI PGauge monitors the the Inficon BPG400 pressure gauge periodically. The data from the gauge are read as a long stream. Synchronization is done by searching for the start pattern (7,5). The data have low outliers. Each output reading is the 75-th percentile of 8 samples.

10 Health of the instrument

For the telescope operator to monitor the health of the instrument, information about the health of the instrument is published for the telescope operator's console.

10.1 Requirements

- The DataSocket Transport Protocol (dstp) will be used for VIs to transfer data about the health of the instrument. The instrument software publishes the data, and a VI for the telescope operator subscribes to the data.
- At a minimum, the data should indicate whether the instrument is operating safely. The data must be interpreted without expert knowledge. (An indicator that turns red when parameters are ourside the safe range is an example.) Other information may be provided.
- Each instrument must publish its health data. The block diagram that publishes the data need not be made public.
- Each instrument must provide a VI that subscribes to the data. SOAR programmers may use this VI as a template for writing tailored VIs.
- Each instrument must provide the telescope operators instructions on getting more detailed information on the health of the instrument. (For Spartan, this is the URL for the remote panel for LogTemperaturePressure.vi, which shows the history of the temperature and pressure.)

10.2 SpartanHealth

The monitor for Spartan checks these conditions: (1) The Spartan software is running. (2) The pressure sensor is operating. (3) The pressure is not too high. (4) The temperature sensor for the liquid nitrogen reservoir is operating. (5) The temperature of the liquid nitrogen reservoir is not too hot.

The data on health is a standard LabVIEW error cluster, which consists of status, code, and location. The status is true if the instrument is unhealthy. The code is used to check whether the instrument is running. When Spartan is running, the code increases once a minute. The location is a description of the error, which is usually the error reported by the instrument. If, however, a problem exists with the connection to the Datasocket server, the DataSocket error is placed in the description.

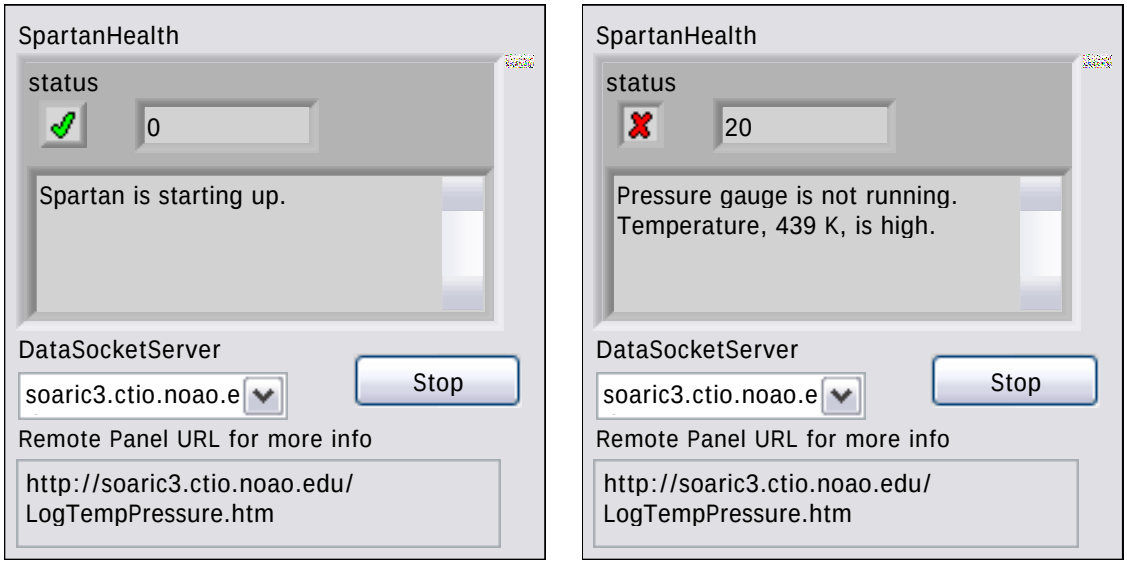

Figure 21: The front panel of SpartanHealth.vi for a healthy condition (left) and an unhealthy condition (right)

Figure 21 shows the front panel of SpartanHealth.vi, which subscribes to the data on health.

11 Troubleshooting

See the troubleshooting section of *Maintenance & Operating Manual, Spartan IR Camera for the SOAR Telescope.*

12 Operating Model and Security

The operating model provides access from the observer's computer and preserves security. The software is meant to be run using "remote panels," a method for controlling the software on one computer from another computer. Three computers are involved: (1) The **Spartan computer** controls the hardware. An observatory person logs into this computer with administrator's privledges, since controlling the hardware requires such privledges. A server for remote panels runs on the Spartan computer. The Spartan computer must be close to the instrument because of cable length. (2) The **gateway computer** runs the software on the Spartan computer through remote panels. Using remote panels, a user on the gateway computer is able to see and to control the front panels on the Spartan computer. The gateway computer is at the observatory. (3) The **astronomer's computer**, which may be anywhere, is connected to the gateway computer through VNC.

The server for remote panels should be set up to allow access only from the gateway computer.

The observatory is responsible for controlling VNC access to the gateway computer. The astronomers who are observing need access; others should not.

The directories for images, observing logs, and instrument logs on the Spartan computer are accessible from the gateway computer. Their default names are `c:\images`, `c:\obsLog`, and `c:\instLog`, and they may be changed by modifying the configuration file `spartan.txt`.

This operating model provides security. Users on the gateway computer and the astronomer's computer may perform a limited set of tasks. Users may run the Spartan software, of course, but remote panels restricts other Labview operations. For example, users may not change the software, because the block diagram is invisible, and they may not save the vi.

13 Installation

You must have already installed LabView (The software uses LabView version 7.1.) and the components Traditional NI-DAQ, NI-Motion, and NI-VISA. If you are uncertain whether

the components are installed, open NI Measurement and Automation Explorer (NI-MAX)[7] and look at the Software tab.

The top folder of the software is `\home Spartan`, and the software requires the directory tree in Table 3 relative to `\home Spartan`. As noted in the table, some of the folders may move outside of the tree.

The steps of the installation follow. For definition of the computers involved, see §12

Configuration The paths of the images, volatile data, observing log, and instrument log are in the file `\home Spartan\Configuration\Spartan.txt`. See §4.3 and Table 4. Make certain that the folders do exist. Make certain that the motor position files `mech0.txt` through `mech4.txt` and `image ID.txt`, do exist in the folder of volatile data. If not, copy them from `\home Startan\Data`.

Image ID The imade ID is a serial number unique to each image. If you moved `image ID.txt`, make certain that the image ID is updated. If not, the images will have duplicate names.

Exporting SpartanTUI SpartanTUI, a text-based interface for the camera, may run on a separate computer. In the distribution are two versions of it, a normal vi, `SpartanTUI.vi`, and a stand-alone application, `SpartanTUI.exe`. To export the stand-alone application, copy the complete contents of the folder `\home Spartan\app\SpartanTUI`. The software, scripts, and configuration file `soar_commsnew.txt` are in that folder. You must update the IP addresses in `soar_commsnew.txt` before running the application. IPClient is the IP address of the computer on which SpartanTUI is running. IPServer is the IP address of the computer that is controlling the Spartan Camera.

Remote panels is a means to control the camera from remote computers. You must set up the web server. In LabView, select Tools>Options. On the panel "Web Server: Configuration," enable the web server. On the panel "Web Server: Visible VIs," allow access to all VIs. On the panel "Web Server: Browser Access," allow "Viewing and Controlling" for the gateway computer. Enter its IP address. Security is implemented by restricting the IP address that may access the software. Three levels of access are possible: Any user from a given IP address may either (1) control the remote panel, (2) only view the panel, or (3) not have any access.

[7] To start NI-MAX, go to Start > All Programs > National Instruments > Measurement and Automation.

14 Cold Start

On the Spartan computer (1) Log on. You must have administrator's privledges. (2) Load SpartanGUI.vi.

On the gateway computer (1) Establish a connection to the Spartan computer via remote panels. Start any VI. Select the menu item Operate>Connect to Remote Panel. You must specify the IP address of the Spartan computer. The name of the VI is "SpartanGUI.vi" (2) Request control of SpartanGUI. Right click on SpartanGUI and select "Request control of VI." (2) Start SpartanGUI by pressing the arrow below the Edit menu or by selecting the menu item Operate>Run.

A Brief Description of All VIs

Included here are brief descriptions of all VIs except those in the SOAR Communications Library[8], the FITS package[9] distributed by SOAR that were not modified, and the NI libraries. The descriptions are taken from the "description" property of the VIs. These are accessible using context-based help and moving the mouse pointer over the icon of the VI.

A.1 VIs Specific to SpartanGUI.vi

SpartanGUI.vi is the user interface for the detector control and motor control.

AssembleMechCmds.vi assembles commands for the mechanism queue.

defaultOptic.vi defines the default optics for LowRes and HighRes modes.

detectorSetPoint.vi returns the set points for a detector.

InitializeDetectors.vi generates primitive command to initialize detectors.

InitializeMenuForSpartanGUI.vi initializes the menus for the top-level VI. If forObserver is true, items useful for programming are removed from the menu.

keepButtons.vi enables or disables a set of buttons.

[8] SOAR 2003, distribution SCLN-1.1.zip

[9] SOAR 2002. The last change was "2002-09-05 CLT 18:07:05 <German Schumacher> FITS module version 1.14."

lookAtMech.vi handles the queue lookAtMech.

Notebook.vi handles instrument & observing notebooks. The notebook name is yyyy-mm-dd.txt. If the notebook is opened for the first time, the date is written first. The VI writes an entry with the universal time.

openHTML.vi opens HTML help for a topic.

SpartanCreatePluginList.vi Reads the plugins directory. Any VIs which are contained in the directory are opened according to the type specifier on the front panel of this VI. If the VI is successfully opened the title of the VI's window is placed in a list, which will be used to create a selection list.

SpartanPlugIn.vi handles plug-ins that are in the directory \home Spartan\PlugIn.

A.2 VIs Specific to CameraControl.vi

CameraControl.vi controls the detectors.

Digital Buffer Read Modified.vi returns digital input data from the internal data buffer.

GLV_I2Buffer.vi reads and writes chunks of data to a buffer to prevent creating new full copies of the data. Modified from GLV.

hoot.vi formats information on occurances.

readNew.vi reads camera data into a single buffer.

readNProcessChunk.vi reads a chunk, splits the data into detectors and quadrants, and stores the data.

readPictStep1.vi prepares to read an image.

readPictStep2.vi reads an image.

ToMeNew.vi controls the To_Me line, which controls the direction of data transfer on the NI6533 card.

writeOldMod.vi writes 4 words to the camera controller.

A.3 VIs Specific to MotorControl.vi

MotorControl.vi gets commands from mechanism queue and calls the appropriate VI.

findReverseLimit.vi finds the address of the reverse limit of a rotation stage. The steps are

1. Turn power on.
2. Call findReverseLimitPrimitive.
3. Turn off power.

findReverseLimitPrimitive.vi finds the address of the reverse limit of a rotation stage. This VI, based on the NI findReference.flx, delays when the limit switch engages in order to recover from bouncing. (Bouncing caused findReference.flx to fail.) The steps are

1. Move off the reverse limit, if it is engaged.
2. Back up until the reverse limit engages.
3. Move forward slowly until the reverse limit releases.
4. Back up slowly until the reverse limit engages again.
5. Move forward 900 steps from the reverse limit.
6. Back up 500 steps.
7. Reset the address of reverse limit if required.

fixposition1.vi (1) rounds the address of a rotation stage to the nearest multiple of 10 to move to a detent position and (2) avoids the address n where $\mod(n, 40) = 20$ in order not to power up at an ambiguous position.

InitializeMotorControllers.vi initializes two NI7334 motor controllers and NI7604 motor drivers. The steps are:

1. Initialize Controller using settings in Controller1 and Controller2
2. Sets “Limit Input Polarity” to noninverting.
3. Disables the “Home Inputs.”
4. Configures the “Inhibit Outputs” to link them with motor on and to set the polarity.
5. Disables limit switches for the wheels.

6. Shuts the power off.
7. Reads the limit switches to determine whether rotation stages are installed.
8. Reads status and positions from disk and loads positions into the controller.

keepMechanismStatus.vi maintains the position and "moving" flag of the mechanisms.

LimitSwitchEnabled.vi returns a cluster LimitSwitchedEnabled, which is true for each axis for which the limit switch stops motion. The limit switches are enabled for mechanisms that are not wheels.

moveRSNew.vi moves mechanism either to an optic or to a target position. The steps are these:

1. Turn power on.
2. Set the flag "moving" and store on disk. If "moving" is set while initializing, then the position is lost because the motion did not finish properly.
3. Move the rotation stage. If anti-backlash compensation is needed and the move is in the forward direction, then add 520 steps to overshoot.
4. If the move is successful and anti-backlash compensation is needed, a second move is made to approach the position in the reverse direction.
5. Put the address in the range [0,180,000) to make it single-valued.
6. Clear the flag "moving." Store it and the position on disk to preserve it for restarting the software.
7. Turn the power off.

moveTo.vi moves to target position and reports position at completion.

NIQuotientRemainder.vi computes the quotient rounded to the nearest integer and the remainder.

OptimizeMove.vi finds the target address that allows for the shortest move. This is used for wheels where addresses that differ by a multiple of 180,000 steps refer to the same position.

powerAxis.vi turns power to an axis on or off.

PowerOnPosition.vi find the address of a rotation stage when power turns on. Address is the nearest multiple of 40.

PutPositionInRange.vi puts the position in the range [0,180000) and stores the position in the motor controller.

SaveImage.vi saves image into obsLog directory. The universal time is coded into the file name as <prefix>yyyy-mm-ddThhmmss.png. Example: If the prefix is xxx, the date is 1 Nov 2006, and the time is 23:15:13, then the file name is xxx2006-11-01T231513.png.

senseLimits.vi senses the forward and reverse limits for an axis.

A.4 VIs Specific to the Command-line Interface

SpartanServer.vi implements a command-line interface to Spartan. Responses to the client are Done, Busy, Offline, or Error. Done means Spartan is free to accept a new command that intiates action. Busy means Spartan is performing an action. Error means the command is faulty or Spartan is in error. Offline means Spartan is offline. Put it online. Done xxx is used to pass information to the client in response to a query.

cmdHistory handles the queue for the command history.

CommandBase.vi converts text command into one that SpartanGUI can interpret.

commandExplanation.vi converts a command into a brief.

getCommand.vi gets commands either from the input or from a script.

parseCmdLine.vi parses command line into a command and modifiers, which delimiters (space, tab, carriage return, and new line) separate.

parseCmdmodifiers.vi parses a line into a command and modifiers separated by whitespace.

readScript.vi reads a script

ScriptHandler.vi handles scripts. Choices of tasks are: 1) get command, 2) open script, 3) abort, 4) initialize.

SpartanQueueCommandLine.vi queues commands in the queue commandLine for processing by SpartanGUI.

SpartanCmdLineHandler.vi handles queries and commands for command-line server. VI parses the command line, creates a reply, and queues the command to SpartanGUI.

A.5 VIs Common to Several Groups

CheckDiskSpace.vi checks disk space.

findOpticFromPosition.vi finds the optic from the position.

getPaths.vi gets the paths for images, the imageID, and positional information for mechanisms.

keepImageID.vi maintains the image ID.

keepInternalMechanismStat.vi handles the section of the global status that applies to mechanisms.

keepStatus.vi handles the status for the Spartan camera. Actions are

startProcess to start a process and change the status to “busy.”

markStart to change the status to “busy.”

processFinished to change the status to “done.”

processErred to the status to “in error.”

shutdownFinished to change status to “shutdown.”

MaintainChannel.vi maintains a data channel, which is used to keep processes from interfering with reading a picture. The caller can block, unblock, or seek permission to use the channel.

mechanismParameter.vi contains parameters of mechanisms. Parameters are name, controller, axis, maximum address, steps/optic, whether backlash compensation is needed, address of 0-th optic, whether mechanism is a wheel, and serial number of rotation stage.

QueueNBEntry.vi queues entries to either the observing notebook or the instrument log.

resizePanel.vi resizes panel and locates it to upper left corner.

SpartanEncodeInstrumentParameters.vi encodes instrument parameters

SpartanHandleParameters.vi keeps instrument parameters. Actions are get, put, put time, and put object. To change parameters, get all parameters, modify them, and put them back.

SpartanQueue.vi handles the queues detector and mechanism.

SpartanToWheelNames.vi translates between indices and names for the mask, filter and Lyot wheels.

status_decodeNew.vi decodes the detector status.

testCommand.vi tests whether command is compatible with the status.

tipsDescr manages tips and decriptions for the controls and indicators on a VI front panel and tab control. The files are in the directory \home Spartan\docs. The possible tasks are: 1) Do nothing. (Used when the tips and descriptions have not changed.) 2) Get tips and descriptions from the files VINameTipStrip.txt and VINameDescription.txt. 3) Put tips and descriptions in the files VINameTipStripOut.txt and VINameDescriptionsOut.txt. (The idea for the VI is from John Brohan jbrohan@TradersMicro.com.)

UniversalTime.vi produces elapsed time in seconds and UTC as a formatted string. Default format is %Y-%m-%dT%H:%M:%S%3u which produces 2004-09-23T06:19:00.000.

UpdateSignals.vi queues signals for the front panel of SpartanGUI.

UTC Get Offset2.vi determines the offset between local time and UTC. (Freeware from Moore Good Ideas, www.mooregoodideas.com.)

A.6 Other VIs

DecodeBPG400.vi decodes the data from the Inficon BPG400 pressure gauge.

FITSHandlerStep2 with GLV.vi collects information on an image and activates the FITS Server.

FITSHandler with GLV.vi calls FITSHandlerStep2 for each detector and quadrant.

FITS_Server.vi writes images in the FITS format. It is the SOAR FITS_Server[10] modified to use less memory and less time. There are two sections, which are queue handlers. Handler #1 dequeues a command from the image queue and runs toFITS.vi to queue the FITS header and image information on the queue COMMANDSQ. Handler #2 executes commands from the queue COMMANDSQ. It is the original FITSServer from SOAR with two changes. (1) The original FITSServer required

[10] SOAR 2002, FITS-1.14.zip

three copies of the image, one copy on the queue, one dequeued, and another reformatted. Now, a pointer to the image is put in the queue, rather than the entire image. This requires only one copy of the image, when the image is passed to FITSWriteSingleImageNoROI.vi. (2) Checking for disk space is now done before reading the image. This moves this time-consuming operation to a less critical time and reduces the number of times it is needed.

LogTempPressure.vi logs temperature & pressure.

PGauge.vi reads the Inficon BPG400 pressure gauge periodically. It monitors the queue PGauge for a quit message. The pressure and status are put in global variables in Environment.vit It uses the 75-th percentile to avoid low outliers.

B Other Documentation

Spartan Documentation

- **Spartan maintenance and installation manual** Loh, E., 2006.

Vendor Documentation

- **Inficon pressure gauge** Inficon, 2002, Operating Manual, BPG400 Bayard-Alpert Pirani Gauge, www.inficon.com.
- **NI digital input/output card** National Instruments, 2001, 653X User Manual, www.ni.com.
- **NI motor driver** National Instruments, 2001, MID-7604/7602 Power Drive, www.ni.com.
- **NI motor controller** National Instruments, 2001, 7344/7334 Hardware User Manual, www.ni.com.
- **Prismatics motor driver** Prismatics, 2005, MDM2200 Reference & Maintenance Manual, www.prismatics.com