PRETTY PICTURES IN HARD REAL-TIME

P. N. Daly, C. F. Claver, D. Dryden, R. Gomez and B. Abareshi
National Optical Astronomy Observatories
950 N. Cherry Avenue, P. O. Box 26732, Tucson AZ 85726-6732, USA
pnd@noao.edu

Abstract

The WIYN Tip-Tilt Module (WTTM) was developed as an adaptive optics solution to image degradation due to atmospheric instability and dynamics. The WTTM was successfully commissioned onto the WIYN 3.5m telescope in February 2002 and is currently undergoing shared risks commissioning. In this paper we review the software design of the WTTM and present some results from the commissioning data.

1 Introduction

The WIYN Tip-Tilt Module (WTTM) has been described in previous workshop proceedings and elsewhere [1, 2, 3]. An engineering schematic of the hardware, as attached to the instrument adapter stage, is shown in figure 1. The software required for WTTM operation is [4, 5, 6, 7]:

```
system:  Red Hat Linux 7.3.
kernel:  kernel 2.4.18.
variant: RTAI rthal5g.
pcisc5:  d16ct and dio316 in-house drivers.
gui:     LabVIEW 6.0.2.
lvrtl:   LabVIEW interface to RTAI.
mbuff:   shared memory devices 0.7.1.
```

A graphical overview of the software is given in figure 2.

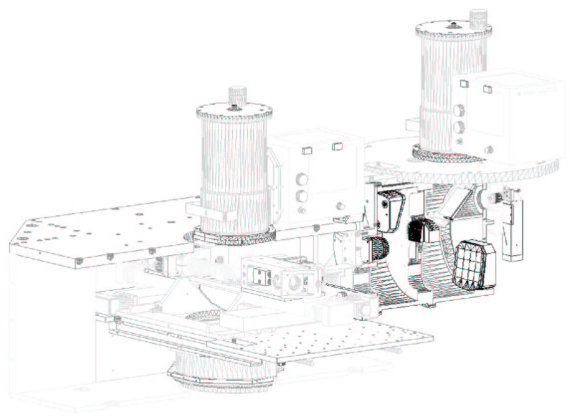

FIGURE 1: *WIYN Tip-Tilt Module.*

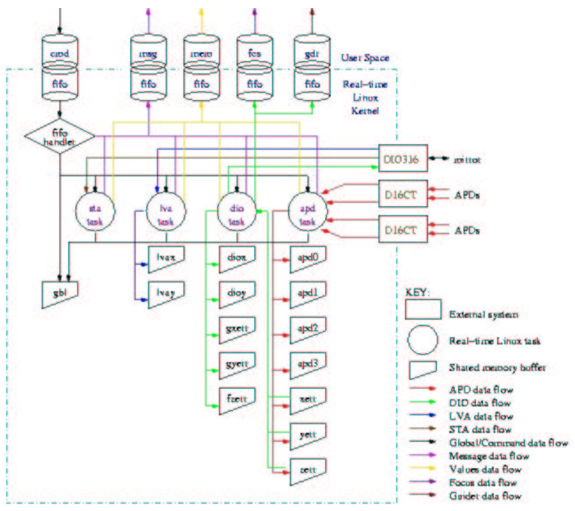

FIGURE 2: *WTTM Core Architecture.*

2 Real-Time Core

As can be seen from figure 2, the real-time core communicates with user space using both real-time fifos (for modest amounts of character based data) and shared memory segments (for bulk data). There is also a global shared memory section.

2.1 Real-Time Fifos

There are five real-time fifos used in the software as follows:

/dev/rtf63 The *cmd* fifo accepts incoming commands from user space and acts upon them within the fifo handler.

/dev/rtf62 The *msg* fifo returns fixed-length messages from any task to user space to avoid well known problems with using *printk* from a real-time kernel. Such messages can be disabled by setting *rtDebug* to 0 (off).

/dev/rtf60 The *mem* fifo returns an array of 25 items associated with some aspects of the system.

/dev/rtf59 The *gdr* fifo returns guider demands which are massaged by a user space program before transmission to the telescope control system.

/dev/rtf58 The *fcs* fifo returns focus demands which are massaged by a user space program before transmission to the secondary control system.

The fifo handler simply waits in a tight loop for data to arrive on the command fifo /dev/rtf63. This data is structured in a simple, but elegant, way:

```
#define INT_SIZE sizeof(int)
#define CHR_SIZE sizeof(char)
#define SHR_SIZE sizeof(short)
typedef struct {
 union {
  char cval[INT_SIZE/CHR_SIZE];
  short sval[INT_SIZE/SHR_SIZE];
  int ival;
 } data;
} TASK_MSG;
```

Typically, the value cval[0] represents an incoming command. If so, the value cval[1] gives the task number to apply the command to. The value sval[1] usually contains the value of some variable. There is no error checking on the communications path. So, for example, we can set the frequency of the APD task to, say, 2500 Hz using:

```
TASK_MSG cmd = {{{0}}};
cmd.data.cval[0] = TASK_FREQUENCY;
cmd.data.cval[1] = APD_TASK;
cmd.data.sval[1] = 2500;
```

This data structure can then be passed along the fifo in the usual way.

Directive	Command	Parameter(s)
FIELD_ROTATION	wsfr	−r
FOCUS_INTERVAL	wsfi	−i
GUIDE_INTERVAL	wsgi	−i
GUI_FREQUENCY	wsgf	−f
HW_CLOCK	wsc	−c0 −c1 −c2 −c3
HW_CLOSE	whc	−c
HW_DATUM	whc	−d
HW_DATUM_XVAL	wsd	−x
HW_DATUM_YVAL	wsd	−y
HW_DATUM_ZVAL	wsd	−z
HW_DATUM_DYNAMIC	wsd	−d
HW_OPEN	whc	−o
HW_RESET	whc	−r
HW_START	whc	−s
HW_STOP	whc	−x
HW_TCOUNT	wst	−t0 −t1 −t2 −t3
LOOP_X_GAIN	wsg	−x
LOOP_Y_GAIN	wsg	−y
LOOP_Z_GAIN	wsg	−z
LOOP_X_INTEGRATOR	wsi	−x
LOOP_Y_INTEGRATOR	wsi	−y
LOOP_HISTORY	wsi	−n
MEM_INIT	wmc	−i
MEM_RESET	wmc	−r
MOVE_X	wsv	−x
MOVE_Y	wsv	−y
RT_DEBUG	wsdb	−d
TASK_BEGIN	wtst	−a −d −l −s
TASK_END	wtsp	−a −d −l −s
TASK_REPORT	wgr	
TASK_UPDATE	wstu	−a −d −l −s
TASK_ALGORITHM	wsa	−f
TASK_CPUS	wtsc	−a −d −l −s
TASK_FREQUENCY	wstf	−a −d −l −s
TASK_SIMULATE	wsts	−a −d −l −s

TABLE 1: *Real-Time Core Commands*

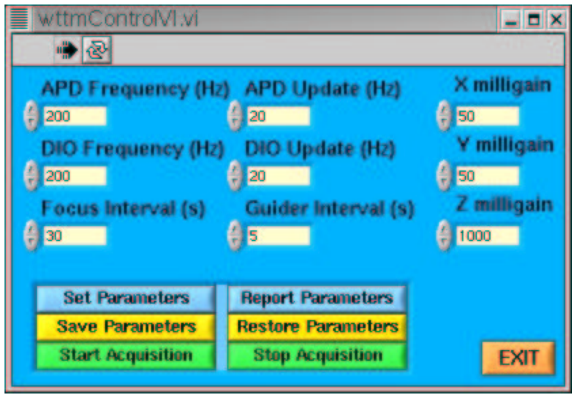

FIGURE 3: *LabVIEW Command Interface.*

There are 3 types of commands: those that can only be used in open loop mode, those that can be used in closed loop mode and system commands. All these commands are shown in table 1. Not all commands are required by the end user so a simple LabVIEW command interface has been built to set the most common parameters via the GUI shown in figure 3.

2.2 Shared Memory Sections

There are 15 shared memory sections created at kernel module initialization time. All but one are 1 Mb in size, the exception being the global shared memory segment. These buffers are re-initialized after every pause–resume operation. The memory sections are:

APD0: raw values from APD 0.

APD1: raw values from APD 1.

APD2: raw values from APD 2.

APD3: raw values from APD 3.

XERR: derived errors for the x-axis.

YERR: derived errors for the y-axis.

ZERR: derived errors for the z-axis.

DIOX: calculated mirror demands for the x-axis.

DIOY: calculated mirror demands for the y-axis.

GXERR: derived values for the x-axis guiding.

GYERR: derived values for the y-axis guiding.

FZERR: derived values for the z-axis focus.

LVAX: feedback mirror positions for the x-axis.

LVAY: feedback mirror positions for the y-axis.

GBLDATA: global data that is shared with user-space.

These memory segments can be dumped out at any time and the resulting data run through a plotting utility such as *xmgrace* to determine system operations. This technique has been used, very successfully, to explore the parameter space of the control loop.

2.3 Real-Time Tasks

All the real-time tasks follow the same generic structure which includes a simulation mode. Each task is assigned a unique number which is also its priority in the real-time system. All tasks use the 'sleep on exit' mode (rather than 'sleep on entry'). Each task has an associated frequency (the rate at which it runs in the real-time core) and an update rate (which is the rate at which it sends data along the *mem* fifo to awaiting user space applications).

2.3.1 The APD Task, Number 0, Priority 0 (Highest)

The function of this task is to read each APD channel, store values and calculate error corrections for all 3 axes. As such it has the highest priority in the system and can run at speeds up to several kHz.

When run, it checks to see if this is the first time it has been run after a *'% wcli resume'* operation (or similar) and if so, reads the APDs and remembers the values. It then goes to sleep until the next scheduling point. When it wakes up, it checks to see if a memory wrap is required (since we can only hold 1 Mb of raw data) and re-reads all APD channels. It then calculates the difference with the last value stored (and does the correct terminal count wrap) and calculates the e_x, e_y and e_z errors from:

$$e_x = [(apd_1 + apd_2) - (apd_0 + apd_3)] / \sum_{i=0}^{3} apd_i \quad (1)$$

$$e_y = [(apd_0 + apd_1) - (apd_2 + apd_3)] / \sum_{i=0}^{3} apd_i \quad (2)$$

$$e_z = [(apd_0 + apd_2) - (apd_1 + apd_3)] / \sum_{i=0}^{3} apd_i \quad (3)$$

These derived values are then scaled by the field rotation factor prior to being written to shared memory. The field rotation is required to co-align the error sensor axes with the tip-tilt mirror axes. It also calculates a running average and variance.

If the task update rate is non-zero, data is sent along the *mem* fifo in accordance with the values specified in wttmTaskData.h. It is important to know that *raw* values are just that: snapshot data at the instant the data is required. But the system also returns accumulated data which is the present value of the running average and variance. So, if we run the task as, say, 500 Hz and specify an update rate of, say, 20 Hz the raw data is a snapshot as though the system was reading the APDs every $\frac{1}{20}$-second. The accumulated data, in this case, is the running average and variance of 500 / 20 = 25 data points.

If the *mem* fifo is active, the GUI shown in figure 4 becomes the main system monitor showing raw APD counts (bottom) and derived error signals (top).

Typically, this task runs in about 10 μs.

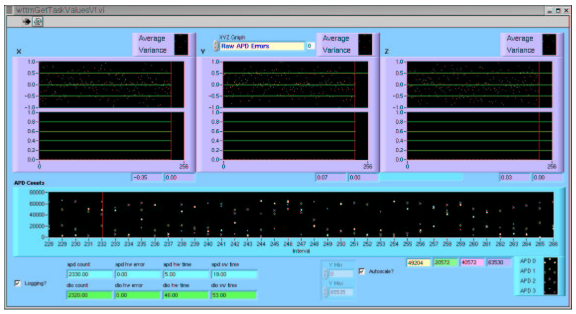

FIGURE 4: *LabVIEW System Monitor.*

2.3.2 The DIO Task, Number 1, Priority 1

The purpose of this task is to move the mirror using derived error signals. As such it is the second most important task in the system and can run at speeds matching the APD task (although it should not exceed it). This task, too, has two loops depending on whether or not it is the first time through the function after a *'% wcli resume'*.

If it is the first time through the loop, it simply re-initializes the memory pointers to keep in step with the APD task. In subsequent iterations, it checks for memory wraps and computes the running average of the e_x, e_y and e_z errors. After having obtained these values, new mirror demand positions are calculated and sent to the mirror. Note that these demand positions are usually used as dynamic datum values the next time around.

The DIO task also maintains a separate running average for focus and guiding and passes the data along the appropriate fifo after the appropriate interval has elapsed. These raw focus and guide values are massaged by user space applications to convert them to reasonable demands for the telescope control system and secondary control system. Thus the tip-tilt mirror is moved at high speed (up to several kHz) but any cumulative error is taken out much more slowly (up to several milliHertz) via the telescope control system.

Typically, this task runs in about 90 μs because, to move the mirror, demands are sent by dynamically reprogramming the on-board chip for each axis.

2.3.3 The LVA Task, Number 2, Priority 2

This task reads the LVDTs on the PI chassis and writes the data directly to shared memory segments "lvax" and "lvay". Typically runs at < 100 Hz. In practice, this task is not enabled and the LVDT read back is not performed.

2.3.4 The STA Task, Number 3, Priority 3 (Lowest)

This task reads the status of the PI chassis and reports back the health of the system. Runs the slowest, with lowest priority, and provides a heart beat at 1 Hz. In practice, this task is usually disabled since it is obvious from the system behavior that the instrument or working or not working.

3 Control Loop Algorithms

To date two algorithms have been implemented:

FIXED FREQUENCY In this scheme, the tasks run at fixed frequencies. This is the most useful mode in stable conditions with optimal seeing.

CONSTANT S/N In this scheme, the APD task counts a certain number of photons and then activates the DIO task directly. Any drop in count rate, therefore, manifests itself in a dynamic slowing of the mirror update rate. In effect, this creates a constant signal/noise. This scheme makes the observation less prone to obscuration by thin cirrus or variable seeing.

The control loop in both cases is a Proportional-Integral loop of the form:

$$M_x = [G_x \times \overline{e_x}] + [I_x \times \sum_{x-w}^{x} e_x] \tag{4}$$

where M_x is the mirror position on the X axis, G_x is the proportional gain, I_x is the integrator gain, $\overline{e_x}$ is the average error, e_x is the instantaneous error and w is the history window size. There is a similar equation that covers the Y axis. Note that no derivative term has been determined for this system as it appears to be sufficiently damped.

4 Results

After a long delay, the WTTM was brought to the telescope during the T&E run of 26–28 February 2002. During this time the WTTM was successfully installed on the telescope and brought on-line. Basic functionality of the WTTM was verified during the first two nights. On 28 February 2002 the team successfully closed the loop while observing the galaxy NGC 2841. During this time multiple exposure images were obtained alternating between open-loop and closed-loop images. In most cases the closed-loop point spread functions (PSFs) showed significant improvement in quality relative to the open-loop PSFs.

In figure 5 we show radial profiles of the same star in open-loop (pluses) and closed-loop (open squares) exposures taken back-to-back. In addition to improvement in the FWHM there was ~50% increase in peak intensity in the closed-loop images.

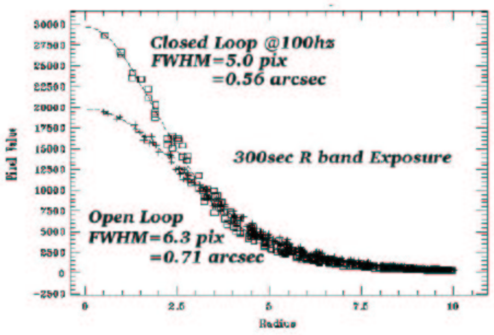

FIGURE 5: *NGC 2841 Radial Stellar Profiles.*

The WTTM is currently using an engineering grade EEV $2k \times 4k$ $13\frac{1}{2}$ μm pixel device for commissioning (on loan from GMOS). Even though this is an engineering grade device not suitable for scientific imaging, observations obtained were capable of showing the marked improvement in image quality capable with the WTTM.

In figure 6 we show uncompensated (left) and compensated images (at 100Hz) in 30 second exposures in the R-band. In this figure the compensated PSF has a FWHM=1.00 arcseconds and the uncompensated image has FWHM=1.29 arcseconds.

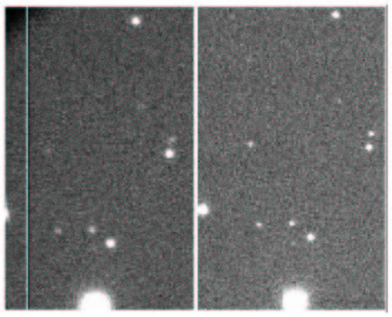

FIGURE 6: *100Hz, R-band 30 second Images.*

In figure 7 we show 300 second exposures of NGC 2841 to show detailed image improvement in the compensated case (right) versus the uncompensated case (left). It was this image from which the stellar profiles in figure 5 were extracted.

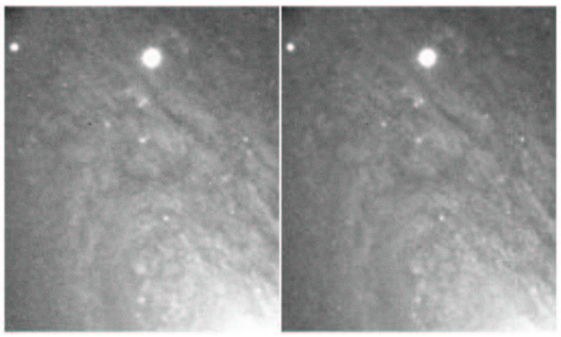

FIGURE 7: *NGC 2841, 300 second exposure.*

After the initial run of success, the instrument was returned to the laboratory for further work. During the lab work, the team discovered a 45 ms phase delay in the error sensor position determination and the application of the correction on the tip-tilt mirror. This, effectively, limited the speed of the tip-tilt corrections to ~20Hz!

Careful analysis of the real-time core showed that memory initialization within the real-time core was the culprit so this was moved to user space. The phase delay dropped to ~80 μs! Subsequent tests in the lab showed that the system is now capable of performing closed-loop operation at 4000 Hz without any limitations (less the availability of photons at such high rates). The WTTM was returned to the telescope for the 23 March 2002 T&E run and the performance gains from the phase delay fix were immediately apparent: typical improvements in FWHM were 0.15 arcseconds or better in closed-loop. The closed-loop rate was measured at 400 Hz for a V=15.5 magnitude star or nearly double the rate required in the original specification.

Lastly, during the T&E run in April 2002, the WTTM delivered its best image quality to date where a focus frame in the Gunn Z filter delivered 0.28 arcseconds and a 300 second integration delivered 0.33 arcseconds rivaling the Hubble Space Telescope (HST) and setting a new record for the WIYN telescope!

In figure 8 we show an image of the western edge if globular cluster M92 in Gunn Z (0.31 arcseconds) and Johnson R (0.33 arcseconds) depicting the improved image quality over a large area of sky. In figures 9 and 10 we show the Ring nebula, M57, from the WTTM and from the HST. To the untrained observer these images are virtually identical.

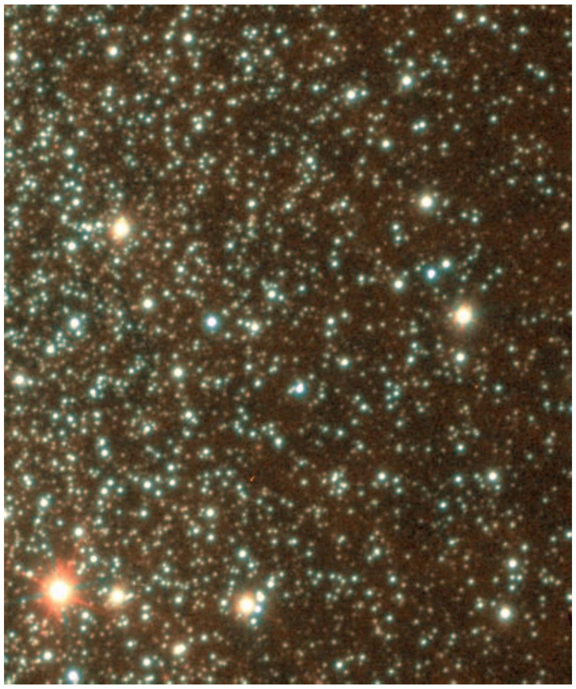

FIGURE 8: *M92 in Gunn Z + Johnson R.*

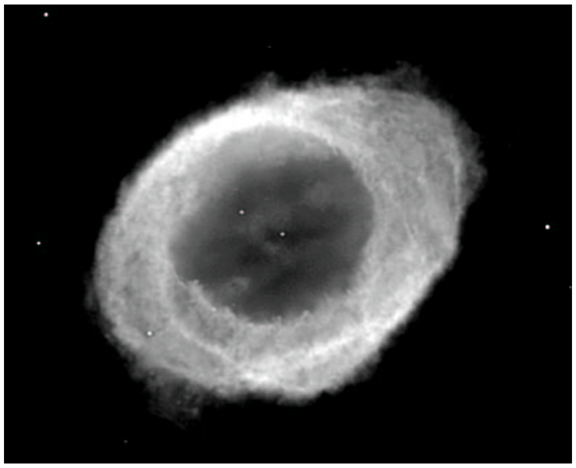

FIGURE 9: *M57 in Hα + [NII] at WIYN.*

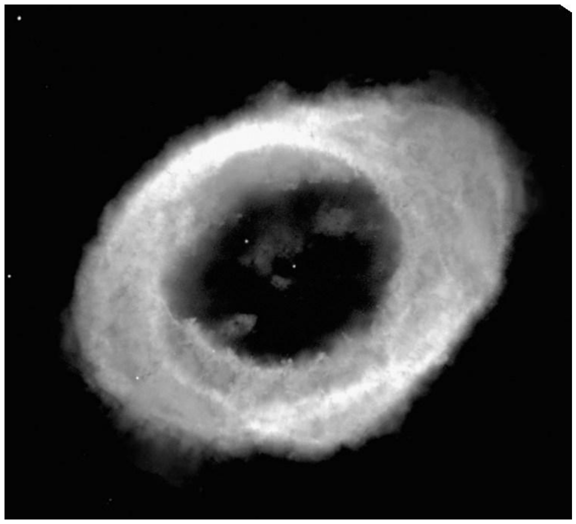

FIGURE 10: *M57 in [NII] from HST.*

The WTTM is now being offered to the US astronomical community for shared risks observing and science papers are already beginning to flow [8]. In figure 11 we show a colour composite image of the Ring nebula ([OIII]=Green, Hα+[NII]=Red).

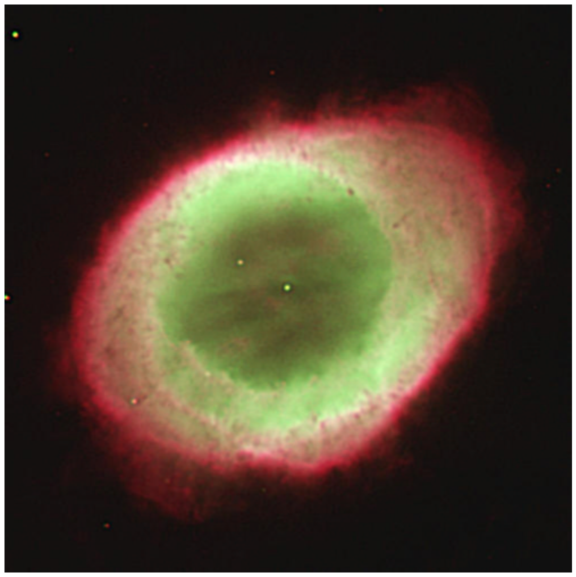

FIGURE 11: *M57 in [OIII], Hα and [NII].*

.

5 Acknowledgments

Linux is a registered trade mark of Linus Torvalds. *RTAI* is copyright Paulo Mantegazza *et al*, DIAPM, Italy. *mbuff* is copyright Tomasz Motylewski. NOAO is the national center for ground-based nighttime astronomy in the United States and is operated by the Association of Universities for Research in Astronomy (AURA), Inc. under cooperative agreement with the National Science Foundation. The WIYN Observatory is owned and operated by the WIYN Consortium, which consists of the University of Wisconsin, Indiana University, Yale University and NOAO.

References

[1] Daly, P. N. and Claver, C. F. 2000, *Real-Time Linux and LabVIEW as a Control Environment for the WIYN Tip-Tilt Module*, in Advanced Telescope and Instrumentation Control Software, Proceedings of SPIE Vol. 4009, H. Lewis, editor.

[2] Daly, P. N., Schumacher, G., Mills, D. and Ashe, M. 1999, *Real-Time Linux at the NOAO*, Proceedings of the Real-Time Linux Workshop I, Vienna.

[3] Daly, P. N. 2000, *Real Time Linux and the WTTM Project*, in Astronomical Data Analysis Software and Systems IX, ASP Conference Series Vol. 216, N. Manset, C. Veillet and D. Crabtree, editors.

[4] Daly, P. N. 2000, *The DUAL16CT Linux Device Driver*, WIYN Tip-Tilt Module Project, Tucson AZ 85719, USA.

[5] Daly, P. N., 2002, *The DIO316 Linux Device Driver*, WIYN Tip-Tilt Module Project, Tucson AZ 85719, USA.

[6] Daly, P. N., 2002, *The PCISC5 Linux Device Driver*, WIYN Tip-Tilt Module Project, Tucson AZ 85719, USA.

[7] Daly, P. N. 2000, *Interfacing Real-Time Linux and LabVIEW*, Proceedings of the Real-Time Linux Workshop II, Orlando.

[8] Jacoby, G. H., Feldmeier, J., Claver, C. F., Garnavich, P. M., Noriega-Crespo, A and Quinn, J. 2002, *Confirmation of SBS 1150+599A as an Extremely Metal-Poor Planetary Nebula*, Astronomical Journal (in press)