CHIRON tip-tilt guider

A. Tokovinin, A. Szymkowiak

Version 1.1. April 18, 2013
file: prj/bme/chiron/guider/chiron-guider.tex

1 Introduction

This document presents the tip-tilt guider for CHIRON. This device compensates tracking errors of the telescope by a rocking plate installed in front of the focus. The plate is tilted in two directions by stepper motors, displacing the star image in the focal plane. The star is thus kept centered on the fiber. The position of the star is measured by the acquisition camera GC-650 (650x499 pixels, pixel scale $0.42''$). The guider corrects star position twice a second.

The rocking plate is a coated window of diameter 40 mm, thickness 6 mm, AR-coated. The range of image displacement provided by the plate is $\pm 9.5''$ or $l = \pm 5.2$ mm for a tilt range of $\pm 14^\circ$[1] The full range corresponds to ± 1500 motor steps, with a scale of 6.4 mas/step.

2 Mechanical design

2.1 The structure module

The new structure replaces existing cylindrical interface between the FEM and the GAM door to which it is attached. Now the FEM is fixed to the door by a cassette-like piece (square 12-cm) with two 60° dovetail corners. On the opposite side, it is pressed by another 60° clamp which rotates on a 1/4" screw. To detach FEM, we loosen the screw and turn the clamp up (counter-clockwise), see Fig. 1.

The cassette is permanently joined with the square box that interfaces to the existing FEM body. The rocking-plate assembly is fixed inside the box, with some of its elements (two motors and connector) protruding outside the box through a suitably large hole. The center of the 10x10-cm box is displaced by 25mm in each coordinate from the optical axis, thus providing space for the guider elements.

All elements of the guider are attached to a triangular plate which is fixed inside the box by two M3 screws. For tuning and service, this plate can be removed easily. The round connector and motors protrude outside through holes in the box.

[1] Image displacement by a tilted plate of thickness d and refractive index n is, to first order, $l = \alpha d(n-1)/n \approx \alpha d/3$ (for $n = 1.5$).

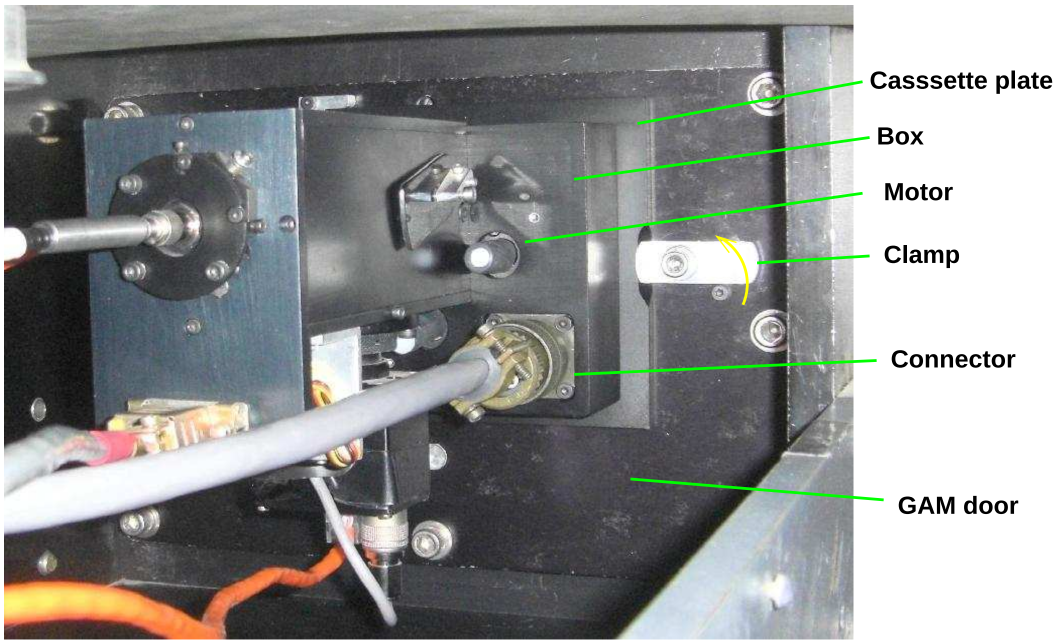


Figure 1: View of the Front-End Module (FEM) with the guider.

2.2 Kinematics

The angle of the rocking plate to the optical axis is controlled within ±14° in both coordinates by two miniature stepper motors. Each motor drives the 25-mm lead screw (M3x0.5) and moves the nut by ±6 mm from its central position. Linear motion of the nut is translated into tilts by the kinematic arrangement illustrated in Fig. 2.

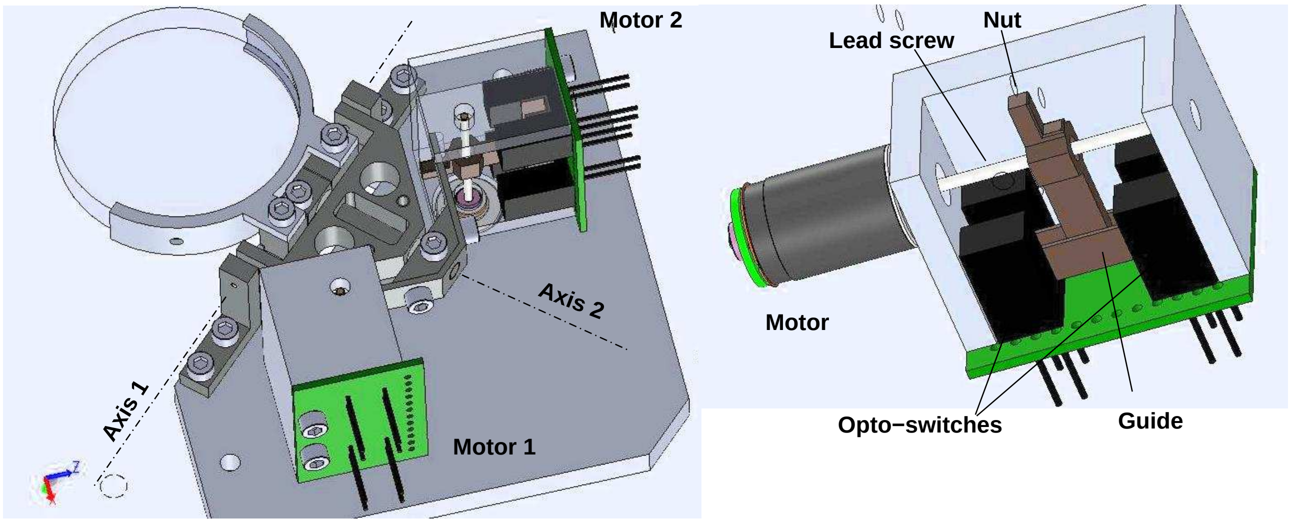


Figure 2: Kinematic principle of the guider.

The glass plate is attached by means of a gimbals-like mount tilting around axis-1 and axis-2. The two orthogonal axes cross at a point which is the center (pivot) of the tip-tilt motion. The axis-2 goes through a “fork” which tilts around axis-1. The axis-2 is solidly connected to the plate mount on one end and to the two levers at roughly 45° angle to it, “anchor”, at the other end. The fork is spring-loaded, providing contact between the levers and the nuts. When the motor 1 is actuated, the pivot point and the contact point of the lever 2 remain fixed and form a virtual axis of rotation. The motor 2 acts in the same way (the kinematics is symmetric). The displacement range of ± 6 mm and the lever length of 21 mm provide maximum tilt of $\pm 16°$ in each direction; the actual range is slightly less.

The right side of Fig. 2 shows the motor module in detail. Rotation of the nut is prevented by a “finger” which slides in a guide parallel to the lead screw. The two home and limit opto-switches QVA11134 define the motion range. They are activated when the nut border crosses the center of each switch, cutting its beam. The lower (left in the Figure) switch acts as a home sensor and defines the nut position corresponding to the step zero. Each motor together with its lead screw, nut, guide, and opto-switches is assembled in a self-contained module which can be connected and tested independently of the rest. Wires that connect to the opto-switches go through 1-mm holes on the protruding border of the PCB, to prevent mechanical stress at the solder points. The wires of the motor connection pass close to the motor body. Figure 3 shows the actual hardware.

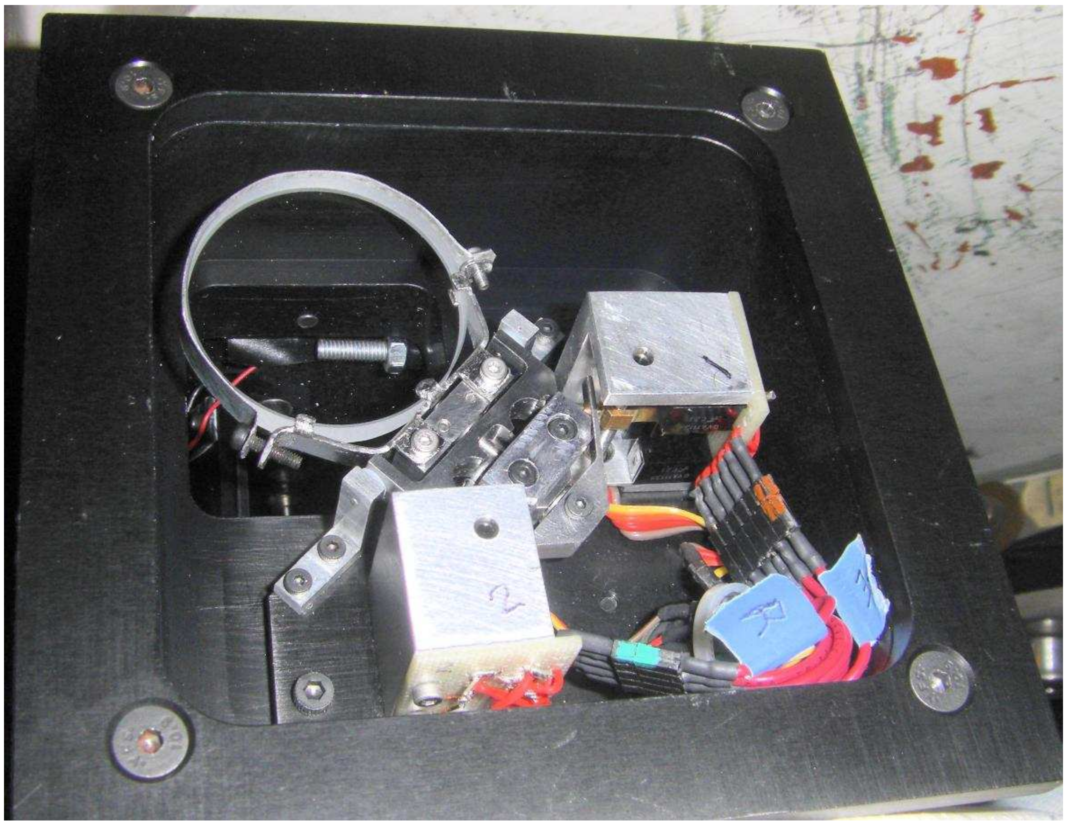


Figure 3: View of the guider module from the front side.

3 Electronics

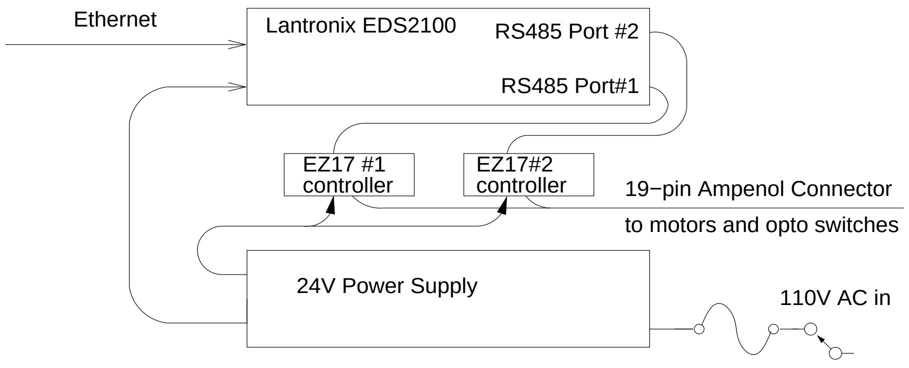


Figure 4: Block-diagram of the guider electronics.

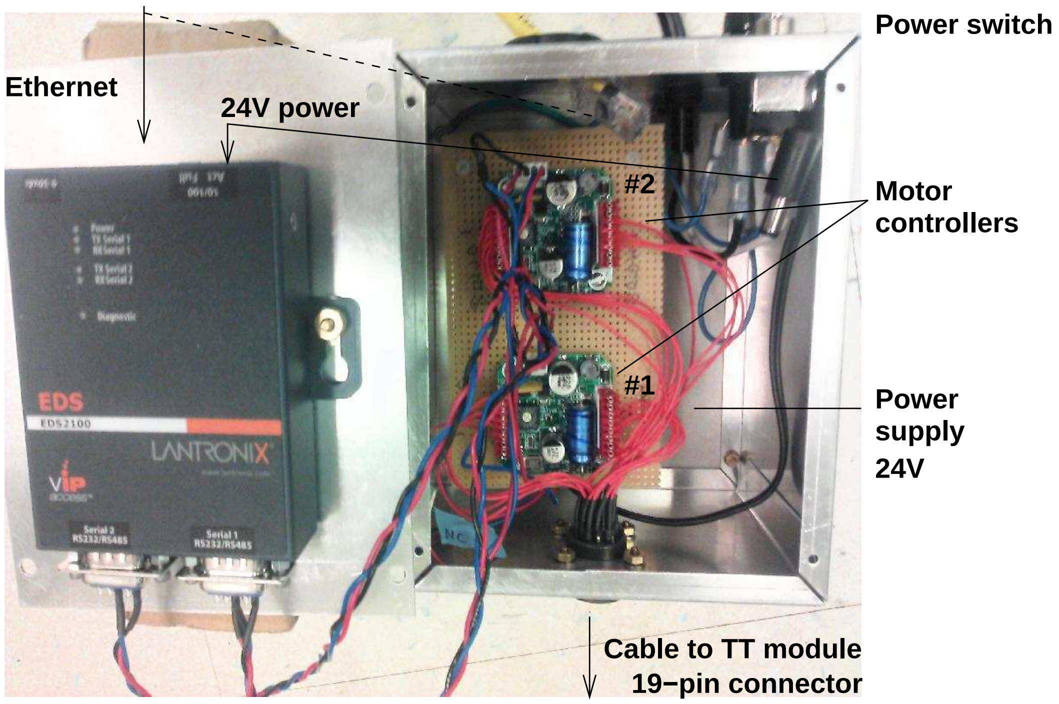


Figure 5: Electronic module

The tip-tilt electronics module (Figs. 4,5) is assembled in a small box and attached to the telescope. It contains the 24V power supply, two stepper-motor controllers EZ17 from `allmotion.com`. These controllers take serial commands on an RS-485 interface. In the tip-tilt electronics box is a Lantronix EDS2100 interface, which is used in tunnel mode to deliver the serial streams to and from the two RS-485 ports. We chose to connect each device to its own port (although RS-485 would have allowed us to daisy-chain both controllers on one port, but at the added expense of a communication bottleneck forbidding simultaneous operations).

The controllers are connected to the guider by a cable that transmits motor currents (4 lines per motor) and opto-switch signals (5 lines per motor). The round connector has 19 pins, with a straight-through cable.

The 110 volt cord goes through a DPST switch and a fuse before going to the inputs of the power supply. Outputs from the PS goes to coax power connector for Lantronix EDS2100 and to the power pins of the two AllMotion EZ17 stepper controllers. The EZ17s are labelled as 1 & 2 on the board they are attached to; Serial output 1 of the Lantronix is wired to comm. pins of EZ17#1 Lantronix tunnel for 1st serial connection is at port 10001 (and 2nd on port 10002).

Conn	EZ17#	EZ17 pin	function
A	1	Mot 3	winding 1 +
B	1	Mot 4	winding 1 -
C	1	Mot 5	winding 2 +
D	1	Mot 6	winding 2-
E	1	Opto 1	ground
F	1	Opto 2	opto#1 sensor
G	1	Opto 3	opto#1 emitter
H	1	Opto 4	ground
J	1	Opto 5	opto#2 sensor
K	1	Opto 6	opto#2 emitter
L	2	Mot 3	winding 1 +
M	2	Mot 4	winding 1 -
N	2	Mot 5	winding 2 +
P	2	Mot 6	winding 2-
R	2	Opto 1	ground
S	2	Opto 2	opto#1 sensor
T	2	Opto 3	opto#1 emitter
U	2	Opto 5	opto#2 sensor
V	2	Opto 6	opto#2 emitter

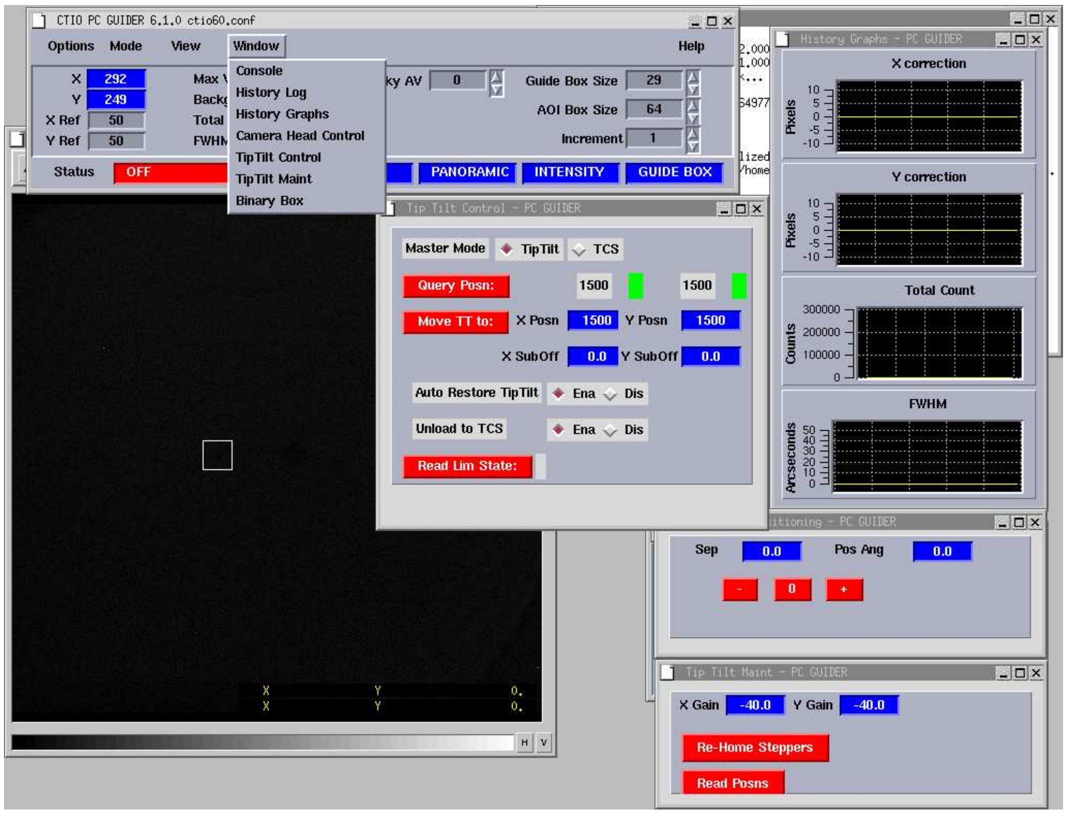
The motors are wired as follow

ez17 pin:	ribbon color:	motor pin:
3	yellow	1A
4	orange	2A
5	red	3B

With the above assignment, negative direction is down towards the plate so EZ17 opto 1 (for homing) is the lower opto (nearer the plate) for both axes (and opto 2 is therefore used for the "upper" limit) For both motors: ~3200 eighth steps covers full range, so 1500 is approximately mid-range (with some margin at top to avoid encountering upper limit).

4 Software

The software that controls the tip-tilt module is an adaptation of the existing PCGuider program done by A. Szymkowiak. It consists of the core module written in C and the graphic user interface (GUI) written in Tcl/Tk. The new version of PCGuider is called 6.1.0. It can be evoked from the X-windows menu (mouse right-click in the background) as `PCguideTT`. The old version which does not use the tip-tilt module is also available in this menu as `PCguide`.



X 292
Y 249
X Ref 50
Y Ref 50
Status OFF
Sky AV 0
Guide Box Size 29
AOI Box Size 64
Increment 1
Master Mode [✓ TipTilt] TCS
Query Posn: 1500 🟩 1500 🟩
Move TT to: X Posn 1500 Y Posn 1500
X SubOff 0.0 Y SubOff 0.0
Auto Restore TipTilt [✓ Ena] Dis
Unload to TCS [✓ Ena] Dis
Sep 0.0 Pos Ang 0.0
X Gain -40.0 Y Gain -40.0

Figure 6: Graphic use interface of the PC Guider program 6.1.0 with the new menu options.

4.1 User interface

Three new options were added to the `"Windows"` menu of the main guider GUI (Fig. 6).

The main one is labelled `"TipTilt Control"`. At the top are two radio buttons, to select between using the tip-tilt mechanism, or to fallback to the old TCS only mode. The next line starts with the `"Query Position"` button. The line next to the button will be populated by information indicating the current position of the stepper motors. This line is automatically updated at a rate programmed into the GUI (currently 5 seconds). Each position has a colored field to the right which indicates if the corresponding stepper is in range (green), out of range (red), or has entered a warning area (yellow). The yellow range is used to trigger the automatic TCS unloading.

The next line starts with the `"Move"` button, followed by two number entry fields. This is the position to which the steppers will move if the "auto restore" feature is enabled, when the guiding is stopped (or if the button is pressed).

The next line allows for the entry of offsets which are added to the errors to account for guiding to the sub-pixel level.

Next are two buttons for controlling the "Auto Restore" feature. This is enabled by default. It causes the mechanisms to move to the position currently entered next to the move button whenever the guider goes from `on` to `off` (or `calculate`) mode.

After that, are two buttons for controlling the use of the "Unload to TCS" feature, which is by default enabled.

At the bottom of this panel, is a button which will fetch the state of the 4 limit opto detectors (as a list of 4 bits). This is intended only to be used to document the state in the case that trouble arises, and should not be used in normal operations.

The third new GUI is invoked by the `"Binary Box"` item. It has two number entry fields labeled "Sep" (for "Separation") and "PA" (for "Position Angle"). Under the number entry boxes are three buttons labelled "-", "0", and "+". When the "+" button is pressed, the guiding box is moved according to the offset numbers. (Pressing it multiple times will continue to step away in that direction, which is not really useful for guiding on a pair of binary stars). The "-" button takes steps in the opposite direction. Pressing the "0" returns the guiding box to the nominal position (292,249) which is hard-coded in the GUI.

Implementation notes: The nominal position is programmed into the Tcl/TK code, as is the scale for converting the separation value into guider box offset amounts. The position angle argument is implemented as counter clockwise in degrees from horizontal (which is not the usual astronomical convention.

The other new GUI is invoked as the `"Tiptilt Maint"` item. This is not intended to be used in normal operations. There are two number entry boxes for changing the x and y gains. There is a button labeled "re-home" which will start the stepper controller hunting for the lower limit opto positions. This is only to be invoked (with guiding off) if one thinks that the positioners have missed some steps. There is another button, labelled "Reload posns", which reads the current positions from the stepper controllers, and reloads them into the software. Again, this should never be required in normal operations, and is only intended for use during problem investigations.

4.2 Software design

The software must control the positions of the two stepping motors. The motors are each connected to an AllMotion EZ17 controller module. These controllers take serial commands on an RS-485 interface. In the tip-tilt electronics box is a Lantronix EDS2100 interface, which is used in tunnel mode to deliver the serial streams to and from the two RS-485 ports. We chose to connect each device to its own port (although RS-485 would have allowed us to daisy-chain both controllers on one port, but at the added expense of a communication bottleneck forbidding simultaneous operations).

An existing subroutine library was used to open sockets to the two ports in the Lantronix that are used to tunnel the traffic to and from the serial ports. Once these sockets have been established, the traffic can be sent and received using standard Unix reads and write.

A small collection of subroutines was written just for operations with the steppers, named `ez17.c`. Two are used most commonly in normal operations; `ez17_send()` which sends a string and waits for the returned status, and `ez17_wait_for_move()` which repeatedly queries the current position of the selected axis, until two successive readings show that the axis is no longer moving. After opening up the communications during program initialization, `ez17_clear()` is used to remove any artifacts from the streams that sometimes are present after turn-on. There is a routine intended for internal use only in the library named `ez17_get_return`, which parses the returned string for the standard header bytes and prints out any error conditions in the returning messages. (Since there can be noise when an RS-485 bus changes direction, this routine discards any bytes seen before the standard header sequence.)

We decided to integrate the tip-tilt control into the framework of the existing CTIO `PCGuider` program, and to keep as much of the existing look-and-feel. The PCGuider functions are actually handled by a series of tasks written in C, which communicates through Unix RPC calls to the GUI components, which are implemented in Tcl/Tk. We started with the code as present in the `/usr/local/pcguider6.0` directory in use on the CHIRON guiding computer. The existing routines took frames from the guiding camera, accumulated the leaky average, passed the data from within the guiding box to one of the guiding algorithms which calculates the x and y error signals from the image. In the previous version, these error were sent through a queue to the `error_task`, which then accumulated a series of corrections, and periodically sent a correction request through a socket to the TCS. The main change we made was to establish a new queue, and have the errors instead sent to a new task, which was established to generate commands to the tip-tilt system.

(Most) all of the new code was created in a new module name `ttLib.c` (In most places, the tip-tilt variables, keywords, functions and queues are prepended with an initial "tt"). During startup, the `tt_init` routine is called to read in parameters, to open the socket connections and to initialize the steppers. We appended the new TT parameters to the existing configuration file `config/ctio60.conf`, and added the routines to fetch the values to the existing `paramsLib.c` routines. A new file is also read from the config directory, named `tt_setup_file`, which contains commands which will be sent to both stepper controllers when the program starts. These are used to set various parameters (modes, velocities, and currents) in the controllers. Then the two controllers are told to perform a "home" operation, which causes them to turn in the negative direction until the lower opto switch detects the entrance of the flag on the rear of the nut that rides on the lead screw. The motors are then commanded to position the nuts at the "restore" position, which is approximately in the middle of their full range.

Once guiding is turned on, the series of error determinations are placed into the `ttQueue`, where the `tt_task` starts to process them. It takes the average of all the corrections found in the ttQueue when awakened, multiplies the average corrections by the loop gains, and adds the requested delta to the current position. If the new position will remain in the "green" zone, the command to request those moves are generated and sent to the controllers. If either of the new positions will take the stepper into the "red" zone, the new command is truncated to stop at one step past the red zone boundary. If the "auto TCS unload" option is enabled, and the new position falls within the "yellow" zone, the stepper motion is skipped, and instead a TCS move is requested by placing the unload step amount into the (old) `errorQueue`, so the previously existing program infrastructure is used to perform the TCS commands.

Three new Tcl/Tk panels were made to control and view the TT operations. These new panels were added to the `"Windows"` pull-down menu of the main guider control panel. They are described elsewhere, but here we will mention some of the aspects of the software written to support them. A new item was added to the existing `parseLib.c` routine that dispatches commands received on the C programs RPC channel. Commands stating with the string `"tiptilt"` are then dispatched to a new routine in `ttLib.c` named `parse_tt_service`. Many of the new commands receive and set values from the Tcl interfaces. There are a few routines to read back (usually parameter values) from the C to the Tcl worlds, or to request updates to current positions, etc. Two of the new commands actually cause actions to occur - one moves the steppers back to the "restore" position (imagined to usually be the center), and the other, not intended to be invoked in normal operations, can cause the stepper controllers to perform another "home" initialization.

4.3 Configuration files

Here are some notes about new parameters added to the `ctio60.conf` file for version 6.1.0 to support operations of the tiptilt stage.

```
TT_IP_ADDRS: lantrxa.ctio.noao.edu
```

The address of the Lantronix EDS2100 in the TT electronics box.

```
 TT_X_IS_1: 0
```

This is a Boolean which indicates whether motor 1 is associated with the X or Y axis.

```
 TT_X_GAIN: -40.0
 TT_Y_GAIN: -40.0
```

The gains applied to convert calculated error amounts to TT stepper motor deltas. Could be negative if either movement is "backwards" from what was assumed when written.

These next eight entries are for setting the "lower and "upper" "red" and "yellow" limits. If the guiding attempts to move the steppers beyond the red limits, they will stop at one microstep beyond the limit (and turn the background of the status field in the GUI to red). Similarly, if the guiding commands the steppers into any of the region between the yellow and red limits, the status field will turn from green to yellow, and the terminal bell will start to sound. If the auto TCS unloading is enabled, the TCS will take a step to move the star so that the stepper should move away from the red limit as the tiptilt follows the movement of the object after the TCS movement. (The full range of the steppers is from 0 to approximately 3200; please keep the red limits slightly less than the full range.)

```
TT_X_LR: 20
TT_Y_LR: 20

TT_X_LY: 950
TT_Y_LY: 950

TT_X_UY: 2050
TT_Y_UY: 2050

TT_X_UR: 3100
TT_Y_UR: 3100

TT_X_RES_POS: 1500
TT_Y_RES_POS: 1500
```

These are the stepper settings at which the mechanisms will start, and to where it will move back when guiding is turned off. These populate the positions in the GUI next to the move button. If the GUI entries are changed, they become the new restore position. For now I am starting these near the middle of the TT range. If there is a buildup of error in one direction (say, RA) during long exposures, may want to start with a different offset.

```
TT\_X\_UNL\_STEP: -1.0
TT\_Y\_UNL\_STEP: -1.0
```

These are the size of the steps the TCS will take if the steppers enter the yellow region. Could be negative, if necessary.

`TT_RATE: 2.0` This is the nominal rate at which TT corrections will be made.

There is also a new file in the config directory, named `tt_setup_file`. This contains a list of stepper controller commands which are sent to both controllers during the program start up sequence. These set various parameters, such as velocities and currents. (The newline at the end of each command from this file is changed into the return that the controllers require).

4.4 Specifications of hardware components

Table 1: Hardware components

Device	Vendor	Part No.	Qty	Cost, $
Rocking plate	TBD	TBD	3	
Stepper motor	www.faulhaber.com	AM1020-2R-V-12-250-23	3	270
Lead screw	www.faulhaber.com	M3x0.5x25	3	incl.
Controller	www.allmotion.com	EZ17	3	175
Opto-sensor	www.digikey.com	QVA11134	10	2.12
Connector (cable)	www.amphenol.com	PT06E-14-19S(SR)	2	
Connector (panel)	www.amphenol.com	PT06SE-14-19P	2	
24V/2.1A power supply	www.digikey.com	SLPower GLC50-24G #271-2306	1	