Ken Sturrock's SkyX Python Library Functions

October 12, 2019

Note #1: Most parameters passed to these functions should be strings and enclosed in quotes. Some do not. If in doubt: Use the source, Luke.

Note #2: Due to changed socket handling code, you need to ensure that "TCP Response Closes Socket" is set to "True" under Preferences -> Advanced.

Acquisition Oriented Functions

adjAGExposure(origAGExp, origAGDelay, XCoord, YCoord)

This routine analyzes a guide star to determine if it is too bright or too dim for the supplied exposure duration. If the guide star is between 20% and 80% of the camera's range, then it does not suggest another value. Otherwise, it will scale the exposure to bring the guide star within this range. If it shortens the exposure, it will pad the delay with the difference. If it lengthens the exposure, it will not increase the exposure by more than 50%. The procedure will not try to measure a star unless the guider uses autodark or a calibration library.

Regardless of if a new exposure is created or the original is passed, the function will return an exposure and delay separated by a comma.

atFocus2(target, filterNum)

Calls @Focus2 (requires "full auto mode") through the specified filter and then Closed Loop Slews back to the specified target. Rif the target is near the meridian, the routine will slew the mount east or west, away from the meridian, so that @Focus2 won't choose a star on the other side of the meridian, which would require a flip. Results are printed as a note and "Success or "Fail" is returned.

atFocus2Both(host, target, filternum)

Calls @Focus2 on both a local and remote Imaging camera. It's necessary to have this special version because @Focus2 depends on slewing to a focusing star. This routine slews to the star, focuses and then asks the remote focuser to focus before slewing back to target. It ASSumes that you will want to focus both cameras through the same filter number.

You must also take some care to calibrate the second (remote) rig on the same magnitude of focusing star as the primary (local). That way the remote rig can use the

same star as selected by the local rig. You'll need to keep the remote machine's "virtual mount" synched with the local mount.

atFocus3(target, filterNum)

Calls @Focus3 through the specified filter. If a target is specified then it will Closed Loop Slew back to the target about 50% of the time (randomly). The reason why it might be a good idea to slew back to target is to serve as an occasional reset for any dithering pattern and to fix any unexpected centering issues. If you don't want to re-slew back to the target then specify the target as "NoRTZ". Results are printed as a note and "Success or "Fail" is returned.

atFocusRemote(host, whichCam, method filter)

A generalized routine to focus either camera (Imager or Guider) with either @Focus2 or @Focus3. It does not return to target if using @F2 because it assumes that you'll handle that with the main script.

calcImageScale(whichCam)

This function calculates the image scale for the specified camera. If no images are available, it will take an image in order to extract the pixel size value from the FITS header. If there are problems extracting the needed information from the FITS header, it will let you know and default to 1.7 AS/Pix (The DSS standard). The guider settling limit calculator and dither routine use this function. The function will return the image scale in AS/Pixel

calcSettleLimit()

Compares the imaging camera and guiding camera image scales and suggests a percent of the guider pixel that represents an imaging pixel. It has some modifications for my Takahashi Temma mounts or for highly under-sampled guiders. The function will return the settle threshold in guider camera pixels.

camConnect(whichCam)

Connects the specified camera ("Imager" or "Guider"). "Success or "Fail" is returned.

The remote version does this task on a remote system.

camDisconnect(whichCam)

Disconnects the specified camera. “Success or “Fail” is returned.

The remote version does this task on a remote system.

camDisconnectRemote(host, whichCam)

Disconnects a remote camera.

camState(camera)

Tells you the numeric and verbose status of the camera.

cloudWait()

Turns off the sidereal drive and waits five minutes. It will then “wake up” every five minutes to see if it can detect stars with the imager. If no stars are to be found after half an hour, it will hardPark(). This routine used to use the guide camera but I switched to the imaging camera for people without guide cameras.

CLSlew(target, filterNum)

Closed Loop Slews to the specified target using the specified filter. It performs the CLS in two stages: an initial slew, followed by the actual CLS. This was done to cope with very slow moving mounts. Results are printed as a note. “Success or “Fail” is returned.

dither()

Dithers the mount using the “jog” technique (versus the “move guide star” technique). The dither directions and distance are random but adjusted based on image scale and declination. The smallest it will dither is about a pixel distance. The distances and directions are printed.

findAGStar()

This module is a re-packaged Javascript routine that analyzes the current guider image and searches for a guide star based on size, shape, location and brightness. Colin McGill & Kym Haines both provided critical assistance and ideas for the routine. It returns X & Y coordinates and a path to the analyzed file.

If you really want to understand what it does, please read through the diagnostic version in my original Bash-based script set.

flipPath(imgPath)

This is a routine that takes a Windows path with forward slashes (which are also used as a special character on UNIX) and converts it to a path that will work with SkyX. If you are running UNIX, it doesn't really change anything.

getActiveImagePath()

This routine returns the path to the currently active image.

getStats()

This provides some basic pointing, FWHM & focuser position statistics. It uses Image Link, so it is a bit slow. It analyzes the current imaging camera image. "Success or "Fail" is returned.

The remote version does this task on a remote system.

getStatsPath(imgPath)

This routine runs statistics on a specified file (full path) as specified by imgPath. It's a bit of a specialized routine which is useful for analyzing groups of previously shot (or modified) images.

getTemp()

Provides the current selected temperature sensor temperature.

hardPark()

Slews the mount towards the appropriate pole, turns off the sidereal motor, resets the camera defaults and disconnects the cameras. Will call the park routine if it works.`

isDayLight()

Tells you if it is light outside and, if it is evening twilight, will wait for darkness.

isGuiderLost(limit)

Determines if the autoguider is lost. It simply takes the supplied guider pixel error limit and multiplies it by three. Returns a "Yes" or a "No".

linReg(arrayX, arrayY)

Runs a simple least squares linear regression between the arrays.

nameFilters(where)

Provides an ~~array~~ list that contains the names of all of the filters in the wheel. Shortens the simulator's list to eight.

preRun()

Uses the "mysterious" INI file variables to make sure that the user has needed configuration values set. "Success or "Fail" is returned.

remoteImageDone(host, whichCam)

Determines if the image executed on a remote system has completed. This is needed to synchronize remote asynch actions with another machine.

reSynch()

Synchronizes the mount position and the sky chart. Don't use this if you have TPoint because it doesn't add anything and will, at worst, screw up the model's pointing.

settleGuider(limit)

Measures the guider error until it reports less than the supplied limit five times in a row. If the error remains higher than the limit for longer than 30 checks, it will go ahead and continue anyway. If the guider appears lost, it will return the status "Lost" so that your main script can do something intelligent.

Just remember, your guider will guide as good as it can. The only purpose of the limit & settle functions is to allow the process to continue sooner rather than waiting some arbitrary time. Returns "Settled" or "Lost".

slew(target)

Slews to the specified target. It uses the same rules as the "find box" and you can specify names without spaces (e.g. ngc363).

The remote version slews a remote (real or virtual) mount.

softPark()

Pauses the script and gives the user a chance to do something. If nothing is done, it will perform a hardPark().

startGuiding(exposure, delay, XCoord, YCoord)

This function starts guiding with the specified exposure and delay on the star specified by the two coordinates.

stopGuiding()

Stops guiding.

takeImage(whichCam, exposure, delay, filterNum)

Takes an image with the specified camera (“Imager” or “Guider”) for the specified exposure, with the specified delay through the specified filter. Use “NA” if there are no filters or you don’t care. Remember that the first filter in the wheel is filter ZERO and not one. “Success or “Fail” is returned.

The remote version runs this task on a remote host.

targAlt(target)

Returns the specified target’s altitude.

targAz(target)

Returns the specified target’s azimuth.

targExists(target)

Makes sure that the target exists in the SkyX database. Returns “Yes” or “No”.

targFromImage(imgPath)

Reads the OBJECT FITS keyword in an image to try to derive the target name.

targHA(target)

Returns the target’s Hour Angle. This is used primarily for determining if the mount needs to flip.

targRiseSetTimes(target, desiredAltitude)

Returns a text string including the time that the target passes the desired altitude on the way up, when it transits and when it passes that same altitude on the way down.

tcpCheck()

This function checks to make sure that the "TCP responses close socket" option is set. It's a kludge and relies on the ASF being in the normal place.

timeStamp(message)

This is a quick way to print an informational message with a time stamp.

TSXSend(message)

This is the core function. It sends the specified message to SkyX across the network socket and returns the result. Note the "Verbose" variable at the top of the file.

The remote version runs the command on a remote machine with a different IP address & port.

writeNote(message)

Works like timestamp, but without the time.

Calibration Oriented Functions

takeDark(exposure, numFrames)

Takes a sequence of dark frame. The exposure parameter tells it the exposure duration to use and the numFrames tells it how many that you want.

takeFlat(filterNum, numFlats, takeDarks)

This routine takes flat frames. The filterNum parameter tells it which filter to use, the numFlats parameter tells it how many flats to take and takeDark should be "Dark" or something else to tell it to take matching exposure duration dark frames.

There is no exposure control because the routine takes a one-second exposure and then scales the exposure duration to get you to about 40% ADU. The takeDarks option is great on a camera with a real mechanical shutter but is a pain in the ass if you don't have a real shutter. If you don't have a real shutter then you're better off shooting all

your flats, then all your darks with the above function. See the “take_flats.py” example in the examples subdirectory.

Science Oriented Functions

circularAverage(type, PAs)

Calculates a circular average where type is mean or median and PSs is a list of values. Returns the average.

classicIL()

This routine is designed to facilitate the use of classic/traditional image link (not all-sky). It does this by setting the expected image scale by reading FITS keywords and doing the math. If the needed keywords aren’t there, it will set the image scale to 1.7 AS/pixel, the same as DSS uses.

dsCatStats()

Looks up the catalog information for the selected double star and gives you the position angle and separation, if available.

dsMeasure(imgPath, primaryStarX, primaryStarY, targPA, targSep)

Tries to measure the PA and separation angle in the image imgPath with a primary located at primaryStarX, primaryStarY and, if available, using targetPA and targetSep as hints for where to look.

If you don’t have values for targPA and/or targSep then set them as “NA”.

If the hints are provided, the module will look for a star around the hinted location. If the PA hint is available, but no separation is provided, then the module will search for the first light source in a triangular area radiating from the primary star location. If neither is available, the module will look for the closest light source to the primary star location.

dsProcess(imgPath)

This module is a macro function that sets up a “work flow” for measuring a double star. First, it takes the image and searches for a primary star name in the image (OBJECT keyword). It then finds the X,Y coordinates that represent that primary star. Next, it looks up the PA and separation (if available) for the target. It then “snaps” to the nearest light source to compensate for any fuzz or inaccuracies in the coordinate

transformation and tries to find the secondary. Finally, it returns the target name, the found PA and Separation as well as the catalog PA and separation.

dumpStars(imgPath)

This function takes a specified image (or "Active") and creates a comma separated values file (aka spreadsheet) that lists all of the found light sources as well as their known X, Y, RA, Dec, a variety of brightness estimates – including catalog magnitude – and the proper names associated with that location based upon your currently selected stellar catalogs.

findRADec(imgPath, targX, targY)

Reports the equatorial coordinates for the provided X,Y location in the specified image.

findXY(imgPath, target)

Finds the light source that corresponds to the specified target – which can be either a proper name (eg "Vega") or a set of SkyX-findable coordinates (decimal notation separeated by a comma or sexigismal format (12h 14m 31.3s, 43d 15m 19s)

getMag(imgPath)

This dumps a variety of magnitude measures into a two-dimensional array. It is called by dumpStars and doesn't offer much standalone.

namesAt(target, limit)

This routine uses a bit of fuzzy ~~thinking~~ logic to click near the target and generate a list of associated proper names and magnitudes. The output is fun to look at, but the return value is simply the magnitude followed by the proper names with semi-colons between them. Thanks to Rick McAlister for the idea.

HMSToDec(H, M, S)

This routine takes individual hour, minute and second parameters and returns "pretty" (zero padded, suitable for printing) versions of those values as well as a decimal version for easier math.