SLA1 Camera Characterization

PSF Candidates in 30s Darks

On May 8, 2024 (UTC) we took various dark exposures with the QHY42 Pro camera.

As they are read in, the darks are scaled to undo zero padding (divided by 16) and the effect of gain (multiplied by 1.39).

They are then combined into a master dark. The master dark is subtracted from the individual darks.

Then we find pixels that (a) exceed a threshold and (b) are brighter than their four nearest neighbors. These are the “hot pixel leaders.&rdquo

Then we cull the hot pixel leaders whose neighbors fall off too sharply using the following quick criterion for non-PSF-shaped regions: any hot pixel leader that has a neighbor <30% of the peak is not a PSF candidate.

Finally, the region around these candidates is displayed.

Notes

The pixels are 1.5x1.5 arcsec. In seeing of FWHM=3 arcsec, the PSF will have a FWHM=2 pixels.

A better criterion to consider implementing later:

Compute the probability that the 5x5 pixels (with >200 e-) could have been drawn from the PSF shape plus Poisson noise. Essentially chi-squared. Normalize perhaps by the peak value set to 1.0, or normalized by total area.

In [1]:
```
# THIS COMMENT IS THE LONGEST A LINE CAN BE AND STILL RENDER COMPLETELY WHEN PRINTING IN LANDSCAPE MODE.
```

```
import os, sys
import numpy as np
from astropy import units as u
from astropy.nddata import CCDData
from astropy.io import fits
from ccdproc import ImageFileCollection, combine, subtract_dark, flat_correct # Combiner
import astroalign as aa
import matplotlib.pyplot as plt
%matplotlib inline
from math import log10, floor

home_directory = os.path.expanduser('~')

# soft link to directory containing raw images
sessions_directory = os.path.join(home_directory, '2024 SLA Sessions')

uv_project_directory = os.path.join(home_directory, 'Projects', 'uv-transients')
analysis_directory = os.path.join(uv_project_directory, 'analyses', '30s_darks')

# The path to the first dark on SLA1 is D:/Raw/2024-05-08/03_38_48/Dark30s/00001.fits
# The files to be processed need to be mirrored on the local machine
# at ~/2024 SLA Sessions/ using the same subdirectory structure.
capture_date = '2024-05-08'
capture_time = '03_38_48'
object_name = 'Dark30s'

# Amount to scale the image data (typically to undo 0 padding of 12-bit to 16-bit values)

scale_due_to_padding = 2**4  # This is division by 16

scale_due_to_gain = 1.39  # from QHYCCD manual for gain of 5

scale = scale_due_to_gain / scale_due_to_padding

# threshold for flagging hot pixels

threshold = 200

# discontinuity limit
```

```
ratio = 0.3

# subdirectory for the 30-second darks (following SharpCap Pro capture directory conventions)
dark_directory = os.path.join(
    sessions_directory,
    capture_date,
    capture_time,
    object_name
)

# exposure duration

dark_exposure = 30.0
dark_exposure_with_ccdproc_units = dark_exposure * u.second

# FITS header confirmation

def confirm_fits_header(image, dimensions, exposure_time, filter):
    header = image.header
    assert header['NAXIS1'] == dimensions[0]
    assert header['NAXIS2'] == dimensions[1]
    assert header['EXPTIME'] == exposure_time
    if filter:
        assert header['FILTER'].rstrip() == filter

# Reader with optional parameter to scale (divide) the ADU readings

def scaled_image_reader(file, scale=1):
    img = CCDData.read(file, unit=u.adu)
    scaled_data = img.data * scale
    img.data = scaled_data
    return img

# After all the preliminaries, we read in and combine the dark files

dark_files = ImageFileCollection(dark_directory).files_filtered(include_path='True')

darks = [scaled_image_reader(file, scale=scale) for file in dark_files]
```

```python
for dark in darks:
    confirm_fits_header(dark, (2048, 2048), dark_exposure, None)

combination_method = 'median'  # alternatively, the method can be 'average'

master_dark = combine(darks, method=combination_method)
```

```
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.151953 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.151953 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.152301 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.152301 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.152648 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.152648 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.152995 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.152995 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.153342 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.153342 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.153689 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.153689 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.154037 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.154037 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.154384 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
```

```
Set MJD-END to 60438.154384 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.154731 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.154731 from DATE-END'.
WARNING: FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.155078 from DATE-END'. [astropy.wcs.wcs]
WARNING:astropy:FITSFixedWarning: 'datfix' made the change 'Set MJD-OBS to -678575.000000 from DATE-OBS.
Set MJD-END to 60438.155078 from DATE-END'.
```

Inspect the Data of the Master Dark and a Representative Dark

At this point, the darks and the master_dark are observed to have values of order 1000 ADU, with some outliers far outside that range.

In [2]:
```
# np.set_printoptions(threshold=sys.maxsize) # Uncommenting this line will cause serious I/O strain

master_dark.data
```

Out[2]:
```
array([[5.56000000e+02, 5.34368125e+03, 3.04062500e-01, ...,
        1.29834687e+03, 1.30108344e+03, 1.25013125e+03],
       [4.98749375e+02, 4.98749375e+02, 1.52947781e+03, ...,
        1.25460531e+03, 1.45228937e+03, 1.23327750e+03],
       [1.49537937e+03, 1.55445437e+03, 1.55723438e+03, ...,
        1.25186875e+03, 1.30108344e+03, 1.41849500e+03],
       ...,
       [6.95955625e+02, 7.73491562e+02, 7.24885000e+02, ...,
        1.34695344e+03, 1.31633000e+03, 1.37132187e+03],
       [7.66975937e+02, 7.93081875e+02, 8.24313437e+02, ...,
        1.61109687e+03, 1.66717469e+03, 1.55719094e+03],
       [9.09190312e+02, 1.59715344e+03, 9.83555312e+02, ...,
        2.41773125e+03, 2.18690438e+03, 2.13873219e+03]])
```

In [3]:
```
darks[5].data
```

Out[3]:
```
array([[ 556.      , 5354.80125 ,    0.      , ..., 1300.17125 ,
        1300.17125 , 1241.27    ],
       [ 498.749375,  498.749375, 1518.40125 , ..., 1318.0675  ,
        1419.36375 , 1216.42375 ],
       [1516.576875, 1583.21    , 1518.40125 , ..., 1239.5325  ,
        1397.558125, 1410.32875 ],
       ...,
       [ 694.391875,  788.13    ,  712.0275  , ..., 1314.505625,
        1383.136875, 1363.155625],
       [ 763.718125,  794.7325  ,  825.920625, ..., 1629.775   ,
        1672.778125, 1522.05    ],
       [ 958.144375, 1596.24125 , 1004.014375, ..., 2470.03    ,
        2114.19    , 2175.17625 ]])
```

Subtract Master Dark from Darks

In [4]:
```python
subtracted_darks = [
    subtract_dark(
        dark,
        master_dark,
        data_exposure=dark_exposure_with_ccdproc_units,
        dark_exposure=dark_exposure_with_ccdproc_units,
        scale=False
    )
    for dark in darks
]
```

Inspect the Data of a Representative Subtracted Dark

The subtracted darks are observed to have values ranging from something like -50 to +50 ADU.

In [5]:
```python
# np.set_printoptions(threshold=sys.maxsize) # Uncommenting this line will cause serious I/O strain

representative_dark_data = subtracted_darks[5].data
representative_dark_data
```

Out[5]:
```
array([[  0.       ,  11.12     ,  -0.3040625, ...,   1.824375 ,
         -0.9121875,  -8.86125  ],
       [  0.       ,   0.       , -11.0765625, ...,  63.4621875,
        -32.925625 , -16.85375  ],
       [ 21.1975   ,  28.755625 , -38.833125 , ..., -12.33625  ,
         96.4746875,  -8.16625  ],
       ...,
       [ -1.56375  ,  14.6384375, -12.8575   , ..., -32.4478125,
         66.806875 ,  -8.16625  ],
       [ -3.2578125,   1.650625 ,   1.6071875, ...,  18.678125 ,
          5.6034375, -35.1409375],
       [ 48.9540625,  -0.9121875,  20.4590625, ...,  52.29875  ,
        -72.714375 ,  36.4440625]])
```

The Routines for Locating Hot Pixel Leaders

As a first cut, we will search for all pixels that exceed some threshold. These are the “hot pixels.”

Then each hot pixel is examined to see if it is the brightest relative to its eight nearest neighbors. If it is, it is added to the list of leaders. (A small bit of tie-breaking code is incorporated.)

In [6]:
```
from collections import namedtuple

Pixel = namedtuple('Pixel', 'x y value')
PSFCandidate = namedtuple('PSFCandidate', 'center neighbors')

def is_winner_or_tied(candidate_leader, i, j, data):
    return candidate_leader.value >= data[j, i]

def is_leader(candidate_leader, data):
    data_height, data_width = data.shape
    for offset_y, offset_x in [(-1, 0), (0, 1), (1, 0), (0, -1)]:
        j = floor(candidate_leader.y + offset_y)
        i = floor(candidate_leader.x + offset_x)
        if j < 0 or j >= data_height or i < 0 or i >= data_width:
            continue
        if not is_winner_or_tied(candidate_leader, i, j, data):
            return False
    return True

def find_hot_pixel_leaders(data, threshold):
    # first we simply find all the hot pixels
    data_height, data_width = data.shape
    exceedances = data > threshold  # an array of true-false values
    values_of_exceedances = data[exceedances]
    exceedance_indices = np.nonzero(exceedances)  # a crafty way of getting the indices of the exceedance
    # all of the hot pixels are candidate leaders
    candidate_leaders = np.transpose([exceedance_indices[1], exceedance_indices[0], values_of_exceedances
    leaders = []
    for i in range(candidate_leaders.shape[0]):
        row = candidate_leaders[i]
        candidate_leader = Pixel(row[0], row[1], row[2])
        if is_leader(candidate_leader, data):
            leaders.append(candidate_leader)
    return leaders
```

Find the Hot Pixel Leaders

Now we classify all the pixels whose values exceed the threshold as hot pixels. From among these, only the ones which are brighter than their four nearest neighbors are declared to be "leaders."

In [7]:
```
hot_pixel_leaders = find_hot_pixel_leaders(representative_dark_data, threshold)
```

The Routines for Finding PSF Candidates

In [8]:
```
def is_too_discontinuous(candidate_psf, i, j, data, ratio):
    return data[j, i] < ratio * candidate_psf.value

def is_candidate_psf(candidate_psf, data, ratio):
    data_height, data_width = data.shape
    for offset_y, offset_x in [(-1, 0), (0, 1), (1, 0), (0, -1)]:
        j = floor(candidate_psf.y + offset_y)
        i = floor(candidate_psf.x + offset_x)
        if j < 0 or j >= data_height or i < 0 or i >= data_width:
            continue
        if is_too_discontinuous(candidate_psf, i, j, data, ratio):
            return False
    return True

def find_psf_candidates(leaders, data, ratio):
    candidates = [leader for leader in leaders if is_candidate_psf(leader, data, ratio)]
    return candidates
```

Find the PSF Candidates

In [9]:
```
candidates = find_psf_candidates(hot_pixel_leaders, representative_dark_data, ratio)
```

Display the Candidates

In [10]:
```
# TODO: Fix the hard-coding

THIS_SHOULDNT_BE_HARD_CODED_X = 2048
THIS_SHOULDNT_BE_HARD_CODED_Y = 2048

def display_candidate(candidate, data):

    lower_x = floor(candidate.x - 2)
    upper_x = floor(lower_x + 5)
    slice_x = slice(lower_x, upper_x)
    lower_y = floor(candidate.y - 2)
    upper_y = floor(lower_y + 5)
    slice_y = slice(lower_y, upper_y)

    fig_size_x = 4
    fig_size_y = 4

    # a bit of fussy code for dealing with display near the edges

    if (lower_x < 0):
        lower_x = 0
        fig_size_x *= upper_x / 5
    elif (upper_x > THIS_SHOULDNT_BE_HARD_CODED_X):
        upper_x = THIS_SHOULDNT_BE_HARD_CODED_X
        fig_size_x *= (THIS_SHOULDNT_BE_HARD_CODED_X - lower_x) / 5


    if (lower_y < 0):
        lower_y = 0
        fig_size_y *= upper_y / 5
    elif (upper_y > THIS_SHOULDNT_BE_HARD_CODED_Y):
        upper_y = THIS_SHOULDNT_BE_HARD_CODED_Y
        fig_size_y *= (THIS_SHOULDNT_BE_HARD_CODED_Y - lower_y) / 5

    fig, axes = plt.subplots(1, 1, figsize=(fig_size_x, fig_size_y))
```

```
    title = "x={}:{}, y={}:{} around {}".format(lower_x, upper_x - 1, lower_y, upper_y - 1, candidate)

    subframe = data[lower_y:upper_y, lower_x:upper_x]

    axes.imshow(subframe, cmap='gray')
    axes.set_title(title, fontsize=12)
    plt.tight_layout()
    plt.show()


for candidate in candidates:
    display_candidate(candidate, representative_dark_data)
```

Inject a Candidate

Since the routine finds no candidates, lets inject one at x=100, y=250, to be sure that it is working.

In [11]:

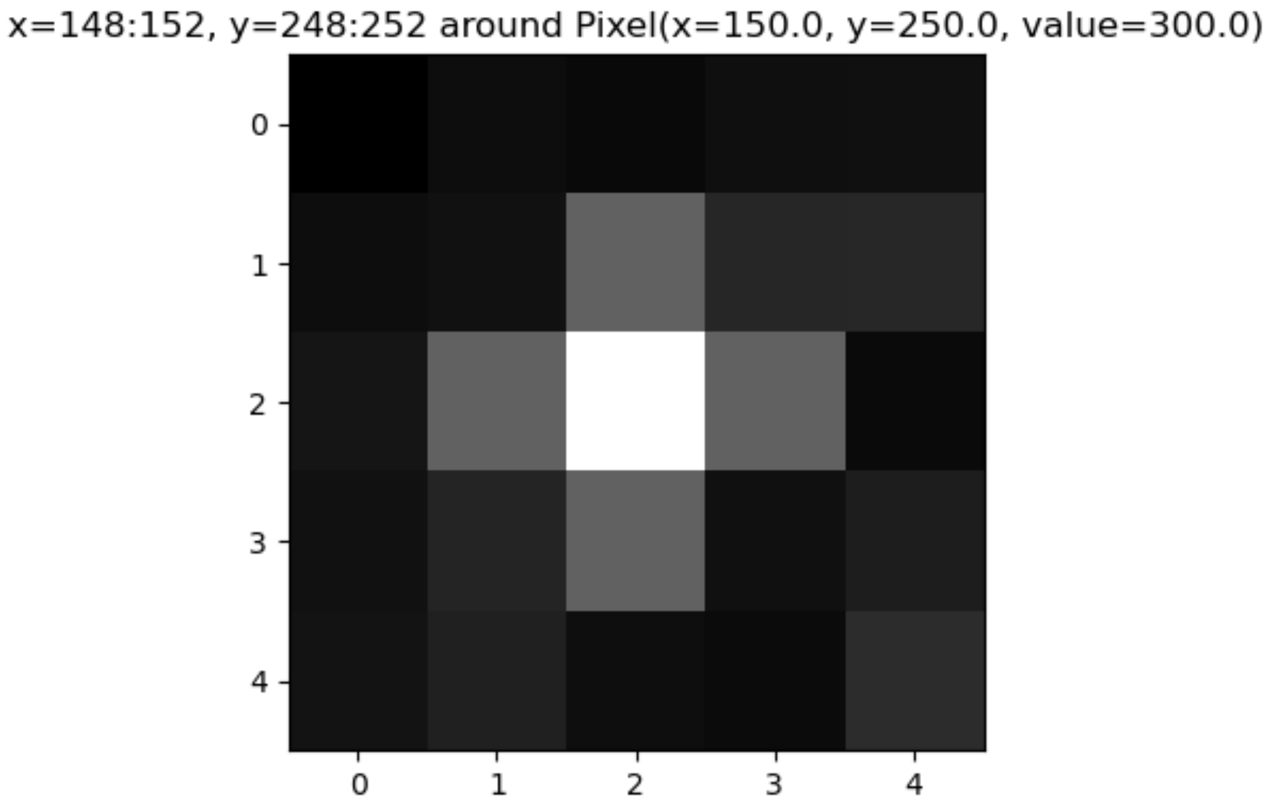
```
injection = [
    (150, 250, 300), # the center
    (149, 250, 100), # neighbors
    (151, 250, 100),
    (150, 249, 100),
    (150, 251, 100)
]

for row in injection:
    representative_dark_data[row[1], row[0]] = row[2]

hot_pixel_leaders = find_hot_pixel_leaders(representative_dark_data, threshold)

candidates = find_psf_candidates(hot_pixel_leaders, representative_dark_data, ratio)

for candidate in candidates:
    display_candidate(candidate, representative_dark_data)
```


x=148:152, y=248:252 around Pixel(x=150.0, y=250.0, value=300.0)
0
1
2
3
4
0
1
2
3
4