ZTF24aahgqwk in NGC 3443

Calibration Notebook

This notebook takes the approximately 18 *2* 60 = 2160 images taken of ZTF24aahgqwk between March 20, 2024 and May 2, 2024 and stacks them into 36 images.

The factor of 60 and the factor of 2 is present because typically a session has 60 30-second exposures per filter and two filters were used, Sloan r' and Sloan g'. There were 18 sessions.

Some sessions have more or fewer than 60 images per filter. For additional detail on any given session, see ZTF24aahgwk Observation Log.

The notebook calibrates the images using calibration data taken April 12.

In [ ]:

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

import os

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

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

supernova_project_directory = os.path.join(home_directory, 'Projects', 'supernova-observation')
stacked_directory = os.path.join(supernova_project_directory, 'analyses', 'ZTF24aahgqwk', 'stacked')

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

# filters

filters = ['r', 'g']
filter_full_names = ["Sloan r'", "Sloan g'"]

SLOAN_R_FILTER = 0
SLOAN_G_FILTER = 1

# exposure durations

light_exposure = 30 * u.second
flat_exposure = 0.1 * u.second
dark_exposure = light_exposure  # our method presumes this equality
bias_exposure = flat_exposure  # our method presumes this equality

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

# Trimmed image reader utility (because the 3x3 binned images have a final row of zeros)

def delete_last_rows_and_columns(arr, rows_to_delete, columns_to_delete):
    row_count = np.shape(arr)[0]
    arr = np.delete(arr, slice(row_count - rows_to_delete, row_count), 0)
    column_count = np.shape(arr)[1]
    arr = np.delete(arr, slice(column_count - columns_to_delete, column_count), 1)
    return arr
```

```python
def trimmed_image_reader(file):
    img = CCDData.read(file, unit=u.adu)
    data = img.data
    trimmed_data = delete_last_rows_and_columns(data, 1, 0)
    img.data = trimmed_data
    return img

def observation_directory_for_date(observation_date):
    return os.path.join(os.path.expanduser('~'), '2024 Sessions', observation_date)

def light_directory_for_filter(observation_date, filter):
    observation_directory = observation_directory_for_date(observation_date)
    return os.path.join(observation_directory, filter)

def calibrated_directory_for_filter(observation_date, filter):
    observation_directory = observation_directory_for_date(observation_date)
    return os.path.join(observation_directory, filter, 'calibrated')

def aligned_directory_for_filter(observation_date, filter):
    observation_directory = observation_directory_for_date(observation_date)
    return os.path.join(observation_directory, filter, 'aligned')
```

Combine the Calibration Images into Masters

Calibration Images

The calibration images are in ~/2024 Sessions/2024-04-12/. In turn, ~/2024 Sessions is actually a soft link to /Volumes/Astronomy Data/2024 Sessions/2024 Sessions.

In [ ]:

```python
# the date on which the calibration images were taken

calibration_date = '2024-04-12'

# calibration directory

calibration_directory = os.path.join(observations_directory, calibration_date)
```

```python
# subdirectory for the 30-second darks

dark_directory = os.path.join(calibration_directory, 'dark')

# subdirectories for the 0.1-second g and r flats

flat_directories_by_filter = {filter:os.path.join(calibration_directory, 'flat', filter)
                              for filter in filters}

# subdirectory for the biases (TheSky Professional Edition may indicate that these are 0.1-second darks)

bias_directory = os.path.join(calibration_directory, 'bias')

# Trimmed image reader utility (because the 3x3 binned images have a final row of zeros)

def delete_last_rows_and_columns(arr, rows_to_delete, columns_to_delete):
    row_count = np.shape(arr)[0]
    arr = np.delete(arr, slice(row_count - rows_to_delete, row_count), 0)
    column_count = np.shape(arr)[1]
    arr = np.delete(arr, slice(column_count - columns_to_delete, column_count), 1)
    return arr

def trimmed_image_reader(file):
    img = CCDData.read(file, unit=u.adu)
    data = img.data
    trimmed_data = delete_last_rows_and_columns(data, 1, 0)
    img.data = trimmed_data
    return img

# darks

dark_files = ImageFileCollection(dark_directory).files_filtered(include_path='True')
darks = [trimmed_image_reader(file) for file in dark_files]

for dark in darks:
    confirm_fits_header(dark, (1381, 940), 30.0, 3, 0.0, 'dark')

# flats by filter
```

```
flat_files_by_filter = {filter:ImageFileCollection(flat_directory).files_filtered(include_path='True')
                        for filter, flat_directory in flat_directories_by_filter.items()}
flats_by_filter = {filter:[trimmed_image_reader(file) for file in flat_files]
                   for filter, flat_files in flat_files_by_filter.items()}

for filter, flats in flats_by_filter.items():
    for flat in flats:
        confirm_fits_header(flat, (1381, 940), 0.1, 3, 0.0, filter)

# biases

bias_files = ImageFileCollection(bias_directory).files_filtered(include_path='True')
biases = [trimmed_image_reader(file) for file in bias_files]

for bias in biases:
    confirm_fits_header(bias, (1381, 940), 0.1, 3, 0.0, 'dark')

# Combine darks, flats, and biases

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

master_dark = combine(darks, method=calibration_combination_method)
master_flats_by_filter = {filter:combine(flats, method=calibration_combination_method)
                         for filter, flats in flats_by_filter.items()}
master_bias = combine(biases, method=calibration_combination_method)

# Perform dark subtraction of the master flats

master_flats_subtracted_by_filter = {filter:subtract_dark(master_flat,
                                                          master_bias,
                                                          data_exposure=flat_exposure,
                                                          dark_exposure=bias_exposure,
                                                          scale=False)
                                     for filter, master_flat in master_flats_by_filter.items()}
```

Load, Calibrate, Align, and Stack Lights

What follows is a giant for loop, done once for each observation date.

In [ ]:

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

aa.PIXEL_TOL = 3  # raised this from the default of 2 due to sometimes poor seeing or wind shake
aa.MIN_MATCHES_FRACTION = 0.2  # lowered this from the default of 0.8
aa.NUM_NEAREST_NEIGHBORS = 7  # raised this from the default of 5
detection_sigma = 1.4 # lowered this from the default of 3.0 to align soft images, especially 2024-03-23

alignment_reference_date = '2024-04-04'
alignment_reference_filter = 'r'
alignment_reference_filename = '00006442.NGC 3443.fit'
alignment_reference_filepath = os.path.join(
    light_directory_for_filter(alignment_reference_date, alignment_reference_filter),
    alignment_reference_filename
)
alignment_reference_image = trimmed_image_reader(alignment_reference_filepath)

# The following detection_sigmas worked -- when aligning a single night/filter data with itself.
# The defaults had to be further adjusted to align all sessions/filters to one reference session/filter.

# observation_dates = [
#     SUCCESS W/ 2.5 '2024-03-20',
#     SUCCESS W/ 2.0 '2024-03-21',
#     SUCCESS W/ 2.0 '2024-03-23',
#     SUCCESS W/ 3.0 '2024-03-27',
#     SUCCESS W/ 3.0 '2024-04-02',
#     SUCCESS W/ 3.0 '2024-04-03',
#     SUCCESS W/ 3.0 '2024-04-04',
#     SUCCESS W/ 3.0 '2024-04-06',
#     SUCCESS W/ 3.0 '2024-04-10',
#     SUCCESS W/ 3.0 '2024-04-11',
#     SUCCESS W/ 2.0 '2024-04-13',
#     SUCCESS W/ 3.0 '2024-04-17',
#     SUCCESS W/ 2.0 '2024-04-21',
#     SUCCESS W/ 2.0 '2024-04-22',
#     SUCCESS W/ 2.0 '2024-04-23',
#     SUCCESS W/ 3.0 '2024-04-29',
#     SUCCESS W/ 3.0 '2024-04-30',
#     SUCCESS W/ 3.0 '2024-05-02'
# ]

# NOW THE REAL CHALLENGE -- GETTING THEM ALL TO ALIGN WITH A SINGLE REFERENCE IMAGE

alignment_reference_date = '2024-04-04'
alignment_reference_filter = 'r'
alignment_reference_filename = '00006442.NGC 3443.fit'
alignment_reference_filepath = os.path.join(
    light_directory_for_filter(alignment_reference_date, alignment_reference_filter),
    alignment_reference_filename
)
alignment_reference_image = trimmed_image_reader(alignment_reference_filepath)


observation_dates = [
    '2024-03-20',
    '2024-03-21',
    '2024-03-23',  # 2024-03-23 was hardest to align -- it forced the most change in astroalign defaults
    '2024-03-27',
    '2024-04-02',
    '2024-04-03',
    '2024-04-04',
    '2024-04-06',
    '2024-04-10',
    '2024-04-11',
    '2024-04-13',
    '2024-04-17',
    '2024-04-21',
    '2024-04-22',
    '2024-04-23',
    '2024-04-29',
    '2024-04-30',
    '2024-05-02'
```

```
]


for observation_date in observation_dates:
    observation_directory = os.path.join(os.path.expanduser('~'), '2024 Sessions', observation_date)

    # subdirectories for the 30-second g and r lights

    light_directories_by_filter = {
        filter:os.path.join(observation_directory, filter)
        for filter in filters
    }

    # lights by filter

    light_files_by_filter = {
        filter:ImageFileCollection(light_directory).files_filtered(include_path='True')
        for filter, light_directory in light_directories_by_filter.items()
    }

    lights_by_filter = {
        filter:[trimmed_image_reader(file) for file in light_files]
        for filter, light_files in light_files_by_filter.items()
    }

    for filter, lights in lights_by_filter.items():
        for light in lights:
            confirm_fits_header(light, (1381, 940), 30.0, 3, 0.0, filter)

    subtracted_lights_by_filter = {
        filter:[subtract_dark(light,
                              master_dark,
                              data_exposure=light_exposure,
                              dark_exposure=dark_exposure,
                              scale=False) for light in lights]
        for filter, lights in lights_by_filter.items()
    }

    # Perform flat division
```

```
calibrated_lights_by_filter = {
    filter:[
        flat_correct(light, master_flats_subtracted_by_filter[filter])
        for light in lights
    ]
    for filter, lights in subtracted_lights_by_filter.items()
}

# In this phase of the analysis, the aligned directories are written to not read from.

# create the aligned directories

aligned_directories_by_filter = {
    filter:os.path.join(light_directory, 'aligned')
    for filter, light_directory in light_directories_by_filter.items()
}

for aligned_directory in aligned_directories_by_filter.values():
    if not os.path.exists(aligned_directory):
        os.makedirs(aligned_directory)


lights_aligned_with_footprints_by_filter = { 'r': [], 'g': [] }

# Not using a list comprehension because it is easier with explicit loops to locate registration fai
for filter in filters:
    print(filter)
    for i in range(len(calibrated_lights_by_filter[filter])):
        print(observation_date, filter, i, light_files_by_filter[filter][i])
        ##############################################################
        # THE FOLLOWING CALL IS FUSSY AND OFTEN FAILS ON POOR IMAGES #
        ##############################################################
        lights_aligned_with_footprints_by_filter[filter].append(
            aa.register(calibrated_lights_by_filter[filter][i],
                        alignment_reference_image,
                        detection_sigma=detection_sigma)
        )
# write the aligned lights

for filter in filters:
    lights = lights_by_filter[filter]
    light_files = light_files_by_filter[filter]
    lights_aligned_with_footprints = lights_aligned_with_footprints_by_filter[filter]
    aligned_directory = aligned_directories_by_filter[filter]
    for j in range(len(lights_aligned_with_footprints)):
        # Then we write all the files for that filter
        light_header = lights[j][0].header
        light_aligned_data = lights_aligned_with_footprints[j][0]
        aligned_file = os.path.join(aligned_directory, os.path.basename(light_files[j]))
        aligned_file2 = os.path.splitext(aligned_file)[0] + '_aligned.fit'
        fits.writeto(aligned_file2, light_aligned_data, light_header, overwrite=True)

# read back in and stack the lights

aligned_lights_by_filter = {
    filter:[CCDData.read(file, unit=u.adu)
            for file in ImageFileCollection(aligned_directory).files_filtered(include_path='True')]
    for filter, aligned_directory in aligned_directories_by_filter.items()
}

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

combined_lights_by_filter = {
    filter:combine(lights, method=stacking_combination_method)
    for filter, lights in aligned_lights_by_filter.items()
}

# create the directories where the stacked lights will be written

if not os.path.exists(stacked_directory):
    os.makedirs(stacked_directory)

# write the aligned lights

for filter in filters:
    stacked_header = aligned_lights_by_filter[filter][0].header
    stacked_data = combined_lights_by_filter[filter]
    stacked_file = os.path.join(stacked_directory, observation_date + '-' + filter + '_stacked.fit')
    fits.writeto(stacked_file, stacked_data, stacked_header, overwrite=True)
```