

Constraining Quasar UV Variability for the L_X – L_{UV} Distance Ladder

JACKSON LYLE¹

¹*University of Florida
Gainesville, FL 32611, USA*

ABSTRACT

The goal of this lab was to detect rest-frame UV variability at 2500 Å in the quasar SDSS J081331.28+254503.0 ($z = 1.512$) using multi-epoch broadband photometry, in order to assess whether UV variability contributes to the dispersion in the L_X – L_{UV} distance ladder relation. Observations were carried out across two nights at the Rosemary Hill Observatory 14” telescope in the B , r , and i filters, and the 2500 Å flux was extracted by fitting a power-law spectral energy distribution to the measured broadband magnitudes at each set of observations. The first night of observations, taken with the moon below the horizon, achieved photometric errors of 0.022 and 0.043 magnitudes in the r and i bands – well within the range needed to detect the expected variability of 0.1–0.2 magnitudes – however, the second night was significantly degraded by the moon being within $\sim 10^\circ$ of the target at 41% illumination, resulting in measurement errors of 0.102 and 0.135 magnitudes. The night-to-night change in the interpolated 2500 Å magnitude was $\Delta m = -0.089 \pm 0.100$ at a detection significance of 0.88σ , which is not a statistically significant detection of variability. Despite this, the photometric precision achieved on the first night serves as a proof-of-concept for the ability of small-aperture telescopes like RHO to contribute to quasar UV variability studies, and a future campaign with better lunar geometry and a longer baseline would be well-positioned to yield a statistically significant result.

1. INTRODUCTION

In our current understanding of the standard cosmological model (Λ CDM), the primary tools for measuring distances across cosmic time are standard candles – objects with intrinsic luminosities that can be inferred from their observed brightness. The major objects currently used as standard candles include Type Ia supernovae, which have highly constrained intrinsic luminosities but are only bright enough to be observed up to around $z \approx 2$. At extremely large redshifts – $z \approx 1100$ – the cosmic microwave background (CMB) allows for measurement of cosmological parameters. Between these two sources, however, lies an extremely large observational gap. Quasars, being the most luminous known objects, are an attractive candidate to fill this gap. Unfortunately, due to their complex physics and geometry, quasars show little standardization in luminosity. However, given a large enough sample size, even relatively weak correlations between spectral features and luminosity could be used for cosmological measurements (Risaliti & Lusso (2015)).

One such relationship was found between the X-ray and ultraviolet luminosities of quasars, specifically in the 2500 Å and 2 keV fluxes. In a large sample of quasars, these bands have a non-linear relationship, resulting in

the X-ray to UV luminosity ratio decreasing with increasing luminosity, allowing their distance to be inferred. The major problem with using this relationship for cosmological measurements was the high dispersion initially found, around 0.35–0.4 dex. After careful selection of quasars with ideal spectra and X-ray measurements, as well as removing dust reddened or obscured objects, one study was able to reduce the dispersion to ~ 0.21 – 0.24 dex, listing variability as a likely source of residual dispersion (Risaliti & Lusso (2019)). Since quasars are known to vary in the UV continuum by ~ 0.1 – 0.2 magnitudes on timescales of months to years (Suberlak et al. (2021)), with this variability thought to be driven by stochastic processes in the accretion disk surrounding the central supermassive black hole that modulate the UV and optical continuum emissions, a single-epoch measurement likely does not reflect the true average luminosity of the object. Despite this, few studies have directly quantified the contribution of UV variability to the dispersion as a standalone term – X-ray variability has been addressed, but the UV side remains an open question (Signorini et al. (2024)).

In this lab, I observed SDSS J081331.28+254503.0, a spectroscopically confirmed quasar at $z = 1.512$ with $r = 15.73$ (Lyke et al. (2020)), across two nights of broadband photometric observations in the B , r , and i filters

at the 14" Meade LX200GPS telescope at the Rosemary Hill Observatory (RHO). At this redshift, the observed r -band maps almost exactly onto a rest-frame wavelength of $\sim 2456 \text{ \AA}$, allowing me to directly probe the UV continuum variability relevant to the L_X - L_{UV} relation. By fitting the spectral energy distribution at each set of observations to extract the monochromatic flux at $2500 \text{ \AA}$, I measured how the UV flux changed between the two nights of observations, separated by a nine-day baseline. This project also serves as a proof-of-concept of the ability for small-aperture telescopes to contribute to quasar UV variability studies at cosmologically relevant redshifts.

2. METHODS

For this lab, two nights of data were collected – the first on 4/14/26 and the second on 4/23/26. Both nights used the Rosemary Hill Observatory (RHO) 14" telescope and SBIG CCD detector to take broadband photometry measurements. Calibration frames were taken using the "Dome Flats" option using a circular flat light panel to take all of the necessary flat calibration frames at the end of the night. Dark calibration frames were also taken at the end of the night by closing the shutter and taking exposures at each science and flat frame exposure time. I also took biases for both nights, using 0.001 second exposures as representative of the bias frames.

The science frames differed night-to-night. The first night of observations had nearly perfect conditions, with the proposed quasar at the upper altitude limit of RHO at 10:30PM, as well as the moon being below the horizon at the time resulting in a very dim background sky. This meant that I was able to observe the object in all three of the observed filters with good SNR, despite the B filter being much dimmer and often difficult to observe a 16+ magnitude object. It is important to note that on this first night I was able to take 3x300 second exposures in each filter as proposed which allowed for a larger SNR.

The second night of observations led to more difficulties, however. While the quasar was observable at the upper altitude limit for RHO at 10:00PM, the angular separation to the moon was under 10° , with the illumination at 41%. This separation is extremely low, resulting in a very bright background sky making the detection of a 16+ magnitude object very difficult. Because of this, I found that 300 second exposures resulted in near-saturation of the background sky, forcing me to use shorter exposures for the r filter. To make up for the shorter integrations, I decided to take 5 frames instead of 3 for any filter taken shorter than 300 seconds. It is important to note for *Analysis* that the moon-illumination

effects were the worst-case scenario for these observations of small variability and it was known that the error on these measurements would likely be greater than the necessary error for statistical significance. This is explored further in later sections of this paper.

The observations of this lab resulted in the exposures summarized in *Table 1*.

To reduce these data, I adapted the Python pipeline I had originally written for *Lab 4*. The conceptual structure is unchanged; the only adaptations were that the filter is now read from the FITS `FILTER` header rather than parsed from the filename, and that the dark-current interpolation routine that previously only operated on flat frames was extended to the science-frame path (necessary for the 300 s 2026-04-23 i -band stack, since no 300 s dark was acquired that night). The pipeline was run separately for each of the two nights' data.

2.1. Detector Characterization

Before applying any calibration frames, I extracted the read noise and dark current of the SBIG CCD from the bias and dark frames. Read noise is estimated as the standard deviation of the median-combined master bias in counts per pixel, converted to electrons using the header gain of $\text{GAIN} = 0.37 \text{ e}^-/\text{count}$. Dark current is estimated by subtracting the master bias from each master dark and dividing the mean residual by the exposure time, again in $\text{e}^-/\text{pix/s}$. Both quantities were recorded for each night separately; all frames were acquired at approximately 0°C CCD temperature. The numerical results are reported in Section 3.

2.2. Creating Master Darks

The first step in reducing astronomical images is typically to eliminate the read noise and dark current accrued in each of your light frames and flat frames. Because flat frames are typically reduced from your dark and bias frames, it is commonplace to create a set of "master darks", or a combined set of individual dark frames, to reduce the rest of your data. The function that does this reads the FITS header of each dark frame to extract the exposure times and sort the frames into a dict using each exposure time as a key. It then uses `numpy` to create a median stacked image for each exposure time, creating a master dark. To create a fits containing the dark frame's noise, I used the following equation:

Table 1. Science and calibration frames used in this analysis. All frames acquired with the RHO 14” telescope and SBIG CCD at 3×3 binning.

Night	Type	Filter	Exposure (s)	Count
2026-04-14	Science (J0813)	<i>B</i>	300	3
2026-04-14	Science (J0813)	<i>r</i>	300	3
2026-04-14	Science (J0813)	<i>i</i>	300	3
2026-04-14	Bias	...	0.001	9
2026-04-14	Dark	...	300	3
2026-04-14	Dark	...	24	5
2026-04-14	Dark	...	4	6
2026-04-14	Dark	...	1	5
2026-04-14	Flat	<i>B</i>	1	6
2026-04-14	Flat	<i>r</i>	4	5
2026-04-14	Flat	<i>i</i>	24	5
2026-04-23	Science (J0813)	<i>r</i>	180	6
2026-04-23	Science (J0813)	<i>i</i>	300	3 ^a
2026-04-23	Bias ^b	...	0.001	5
2026-04-23	Dark	...	240	3
2026-04-23	Dark	...	180	4
2026-04-23	Dark	...	70	5
2026-04-23	Dark	...	24	5
2026-04-23	Dark	...	4	10
2026-04-23	Flat	<i>B</i>	70	6
2026-04-23	Flat	<i>r</i>	4	10
2026-04-23	Flat	<i>i</i>	24	6

^aTwo additional *i*-band exposures (one at 180s, one at 240s) were acquired on 2026-04-23 but excluded from the stack to maintain a uniform 300s effective exposure.

^bBias frames for 2026-04-23 were saved with **Darks** filenames per the observing log; identified as 0.001s exposures.

NOTE—No 300s dark frame was acquired on 2026-04-23; the dark current for the 300s *i*-band stack was interpolated from the available shorter-exposure darks and the master bias. Calibration dark exposure times other than 300s were chosen to match the corresponding flat-field exposure times. The 70s dark and *B*-band flat on 2026-04-23 were processed but not applied to any science frame, since no *B*-band data was acquired that night due to moon-driven sky brightness.

$$\sigma_{\text{dark}}(x, y) = \frac{1}{\sqrt{N}} \sqrt{\frac{1}{N-1} \sum_{i=1}^N (D_i(x, y) - \bar{D}(x, y))^2} \quad (1)$$

This equation first measures how much each pixel fluctuates using the equation for standard deviation, and then reduces that value by the square root of the total number of frames because averaging multiple frames de-

creases random noise by this factor. These frames were all saved in two separate dicts for use later.

2.3. Creating Master Biases

While not always necessary, it is good practice to take bias frames while taking the rest of your reduction frames. If you have dark frames for every exposure time that your flat and light frames were imaged in, they are not strictly necessary. However, without bias frames

it is much more difficult to scale dark frames to different exposure times. This is because dark frames contain both dark current noise and read noise, whereas bias frames essentially only encode read noise due to such a short exposure time. It is best practice to both plan to take all of the appropriate dark frames, but also capture bias frames just in case. As it turns out, on 2026-04-23 I did not acquire a 300s dark frame, requiring the use of the master bias to scale the available shorter-exposure darks up to the 300s science exposure time for the *i*-band stack. The bias frames for that night were also saved with `Darks.*` filenames per the observing log and are identified as the 0.001 s exposures within that set.

The process of creating a master bias frame is relatively simple, as they are all taken at a nearly 0 second exposure time and can be taken in any filter. My code for this section does essentially the same process as I used for the dark frames, only without the need to sort multiple master bias frames into a dict.

2.4. Creating Master Flats

Flat frames are integral to creating good final images. Without them, you leave dust rings, vignetting, and gradients in your image. To create master flat frames for each filter, I first had to sort the frame paths into a dict using each filter as a key. For this lab, the `FITS FILTER` header was populated, so I sort frames by reading that header value rather than parsing the filename as I did in *Lab 4*.

Next, I had to similarly create the master flat frames and accompanying noise through stacking the individual frames in each filter. This process was slightly more involved, however, since the noise propagation is not as straightforward. As in *Lab 4*, the pipeline includes a fallback for cases where no dark frame matches the exposure time of a flat: I made a simple check to see if I had the correct dark frame for a flat, and if I did to just set that frame as my calibration frame. If not, however, an else statement went through the following process:

1. Find the dark that is closest in exposure time to the needed time
2. Use that closest dark frame and subtract the master bias frame
3. Divide this new frame by the exposure time of the dark frame to get the dark current
4. Calculate the dark current noise by taking the square root of the dark noise squared plus the bias noise squared, divided by the dark exposure time
5. Create the calibration frame as the master bias plus the dark current times the exposure time
6. Create the calibration frame noise as the square root of the bias noise squared plus the dark current noise times the exposure time of the flat frame squared.

Once the calibration frame and calibration noise are defined, I had to stack each set of flat frames. For each filter I create a flat noise array that held the square root of the flat frame values plus the square of the calibration noise values. This is done because when you subtract the dark frames from a flat or science image, the noise adds in quadrature. I then subtract the calibration frame (equating the dark frame) from each flat frame. With this new array of corrected flats, I create a master flat through taking a mean of each filter's frames. I take a mean here instead of a median for the flat frames as the majority of the noise in flat frames is from Poisson noise. Because I am less worried about cosmic rays and read noise here, since all pixels are around half-well depth, mean combining uses more of the data. Since I am mean combining, the total flat noise is calculated by taking the square root of the sum of the noise stack squared, divided by the number of frames in each filter. Finally, both the flat noise frames and master flats are divided by the mean of each master flat to properly normalize the flat frames. This makes it so that, when the flats are applied to the lights, regions of less light will become brighter and vice versa.

2.5. Creating Master Lights

After creating all of the calibration frames, it was time to apply these reductions to the light frames themselves. The first part of my code is quite similar to what was done earlier with the flats and darks. I sort each science frame into a dict with the keys being filters, then exposure time, reading both values from the `FITS` header as in the master flats step. While, technically, this would not be necessary if only a single exposure time were used for each filter, I wanted to make the pipeline more robust so I included the exposure time as a key.

With the light frames sorted by filter and exposure time, I then proceeded by subtracting the dark frames and creating the new noise files. The dark subtraction was the same process as was used in the master flat section, where I subtracted the master dark frame – or, for the 300s 2026-04-23 *i*-band stack where no matching dark was available, an interpolated calibration frame computed via the same routine described above – from each science image, and added the poisson noise and dark noise in quadrature for the noise frame.

Next, I used the appropriate flat frame from each filter and divided the dark subtracted light frame by the normalized master flat. For the error estimate I had to

utilize a slightly different equation since I am dividing by the flat frame:

$$\frac{\sigma_T}{T} = \sqrt{\left(\frac{\sigma_A}{A}\right)^2 + \left(\frac{\sigma_B}{B}\right)^2} \quad (2)$$

For my code, A represents the dark subtracted light frame, B represents the master flat frame, and T is the dark and flat reduced light frame. I did have to pay careful attention to ensuring that there were no zero values for the flat and dark subtracted light frames, as dividing by zero would raise errors. To fix this, I created a new array for both frames using numpy's zeros-like function. I then used a mask to determine what pixels had values greater than zero, then created the fractional elements only using these values, leaving the rest to be zero.

The final step in creating my master light images was to align and stack the images for each filter. Due to the large number of files that I would be reducing across multiple filters and nights, I decided to use the *astroalign* package to attempt to automatically find and apply the correct transform. This is extremely helpful as, in an earlier homework, I found that the data for M42 had significant field rotation, meaning that for a proper alignment I would have to additionally employ a rotation matrix transform. *Astroalign* is able to handle both translations and rotations, however, and is even able to interpolate between pixels.

To use *astroalign*, I first set the reference frame to be the first frame in each filter's dict. While this could lead to problems if the first file had bad tracking or other significant aberrations, I did not find these troubles. I created new dicts for my aligned lights and aligned noise files, and found the transform between each image and the reference frame using the *find transform* function. Once the transform was found, it was applied to both the light frame and the corresponding noise frame to ensure both had the same transformation. I wrapped the *find and apply transform* section in a *try, except* clause as a safety net, but for this lab's data *astroalign* did not raise a single error and the final stacks appeared to be properly aligned.

Once all of the images were aligned, it was time to stack to create the master images. For this part, I opted to use a median combination instead of a mean combination. This is because I wanted to mitigate the effect of cosmic rays or the occasional satellite. However, the math behind error propagation for a median combined image is much more complex than it is for a mean combination. For simplicity, I assumed the error scaled as a mean combination, but this is explored further in *Error Analysis*.

To perform the median combine, I first concatenated all of the possible exposure times for an individual filter into one array, as well as for the noise frames. I then used the *numpy* median function to create one master light per filter. Note that using a median combine for different exposure times could possibly lead to some weirdness depending on how many frames were taken at each exposure time. For this lab, all science frames within a single filter and night used a uniform exposure time after excluding the two non-300 s *i*-band frames on 2026-04-23 (Table 1, footnote a).

For the master noise frames, I assumed a mean combine and therefore used the following equation to create the final frames:

$$\sigma_F = \sqrt{\frac{\sum_i F^2}{N_{frames}}} \quad (3)$$

I then saved these final frames as fits files with corresponding names.

2.6. Photometry Pipeline

Now that I had all of the reduced image stacks for each filter, I created a photometry pipeline in order to extract the photometric data needed for analysis. This pipeline consisted of the following logic:

1. Plate solve each stacked frame using *astrometry.net* and save the new solved .fits file
2. Load in each noise frame and corresponding solved frame
3. Open a display of a single stacked frame and query to select a star
4. User selects the Target, 3 References, and 3 Test stars by clicking the image
5. For each selection, convert the pixel coordinates to RA and DEC values using the WCS information
6. Query multiple Vizier catalogues in a 5 arcsecond radius around the user selection and return the magnitude in that filter
7. Measure and store the FWHM of each source
8. Use the FWHM value to create and analyze an aperture and background annulus
9. Background subtract the aperture values using the annulus and convert pixel values to zeropoints based on the magnitudes pulled from Vizier for each reference star

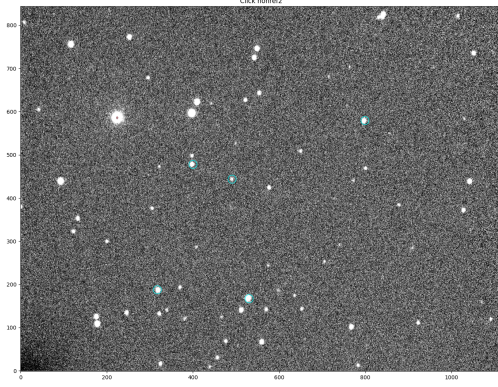


Figure 1. Example of the `display_and_select` widget used to choose target, reference, and test sources. Already selected sources are shown with blue apertures.

10. Use the calculated zeropoint to measure the apparent magnitude and errors for the target source and 3 test stars.
11. Store all of the photometry information for each frame as a CSV for further analysis

To code this pipeline, I relied heavily on functions to allow for the processes to be repeated over each frame. To begin, I defined a `load_stack` function that takes the frame file path as a parameter and pulls the image data, header, and WCS information from the solved image:

```
def load_stack(path):
    with fits.open(path) as hdul:
        image = hdul[0].data
        header = hdul[0].header
        wcs = WCS(header)
    return image, header, wcs
```

Next, I wanted to be able to see the stretched frame to manually select the quasar and the necessary reference and test stars. I thought that this would be easiest to avoid manually hard-coding pixel values for each frame since there are slight deviations in the source locations frame-to-frame. To do this, I created a `display_and_select` function that defines 7 "roles": *target*, *ref1*, *ref2*, *ref3*, *nonref1*, *nonref2*, *nonref3*. The nonref stars are used to ensure that the selected reference stars are adequate and lead to values that match Vizier catalogue values. Once the image is displayed, the title of the plot changes to indicate which role you are selecting, and the `plt.ginput` function is used to store the coordinates of the clicked location on the image and then use a small marker to indicate which source was chosen, as shown in *Figure 1*.

To avoid selecting saturated stars, I also plotted an overlay with matplotlib.pyplot which highlights regions above a certain pixel value to represent stars that are

nearly saturated. Since the frames on 4/23 had very high background-sky counts, I chose this threshold to be 62000 counts observationally.

```
def display_and_select(image, roles=None):
    if roles is None:
        roles = ["target", "ref1", "ref2", "ref3",
                "nonref1", "nonref2", "nonref3"]
    fig, ax = plt.subplots(figsize=(10, 10))
    vmin, vmax = np.percentile(image, [5, 99])
    ax.imshow(image, cmap='gray', vmin=vmin, vmax=vmax,
              origin='lower')

    sat_overlay = np.where(image >= SAT_THRESHOLD, 1.0,
                           np.nan)
    ax.imshow(sat_overlay, cmap='Reds', vmin=0, vmax=1,
              alpha=0.5, origin='lower')

    selections = {}
    for role in roles:
        ax.set_title(f"Click {role}")
        fig.canvas.draw()
        x, y = plt.ginput(1, timeout=0)[0]
        selections[role] = (x, y)
        ax.plot(x, y, 'o', mfc='none', mec='cyan', ms=15)
        fig.canvas.draw()

    plt.close(fig)
    return selections
```

To identify the target quasar in each frame, I used two methods. First, I used the SIMBAD finderchart to identify the patterns or nearby stars. Second, I used the WCS information to correlate the pixel position of each frame to the RA and DEC of the SIMBAD coordinates of the quasar, allowing me to identify if my guess with the finder chart was accurate. For reference stars, I chose to use three random stars of decent but not saturated SNR. It is important to note that the reference stars used in each frame differed frame-to-frame, and I relied on the variance of their zeropoints to determine if the selected reference stars for one frame were adequate. The non-reference stars used were also randomly chosen stars with decent SNR, and also differed frame-to-frame.

To query the Vizier catalogues for each star, I created another robust function to search multiple databases in an attempt to find magnitude values for dim, magnitude ~ 16 stars. One of the problems that I ran into early on was that different catalogues used different votable field labels for each filter, with some catalogues using `r_mag` and others using `r_mag`, for example. Because of this, I looked for at least two catalogues that included each filter's magnitude and stored the field label in a dict:

```
target_cats = {
    'B': {'Bmag': ['I/322A/out', 'II/336/apass9']},
    'R': {'rmag': ['I/322A/out', 'II/349/ps1'],
          'r_mag': ['II/336/apass9']},
    'I': {'imag': ['I/322A/out', 'II/349/ps1'],
          'i_mag': ['II/336/apass9']},
}
```

To query each object, I converted the RA and DEC coordinates to ICRS coordinates and used the `Vizier.query_region` function with a 5 arcsecond radius to loop over each catalogue and search for a value. Once a value for one filter is found, the loop is broken and it moves on to the next filter/object:

```
def query_catalog(ra, dec, filter_band,
                 radius_arcsec=5.0):
    coord = ICRS(ra * u.degree, dec * u.degree)
    for col_name, cat_list in
        target_cats[filter_band].items():
        for cat in cat_list:
            print(cat)
            try:
                result = Vizier.query_region(coord,
                                             radius=radius_arcsec * u.arcsec,
                                             catalog=cat)
                if result is not None:
                    time.sleep(0.1)
            except Exception as err:
                print(f"Query failed for {cat}: {err}")
                continue
            if result and len(result) > 0:
                value = float(result[0][col_name][0])
                if not np.isnan(value):
                    print(f"{filter_band}: {cat}")
                    return value, cat
    return None, None
```

Before performing aperture photometry on any of the targets, I wanted to measure the FWHM of each selected object. In my `measure_fwhm` function, I used the moment operations used in the photometry homework notebooks to estimate the FWHM based on a cutout of the object and a `sigma_to_fwhm` conversion. This function returns values for the FWHM, x-centroid, and y-centroid:

```
def measure_fwhm(image, x, y, cutout_size=21):
    half = cutout_size // 2
    x_int, y_int = int(round(x)), int(round(y))
    cutout = image[y_int - half:y_int + half + 1, x_int
                  - half:x_int + half + 1]
    cutout_sub = np.maximum(cutout - np.median(cutout),
                           0)

    yc, xc = np.indices(cutout_sub.shape)
    total = cutout_sub.sum()
    mom1_x = (xc * cutout_sub).sum() / total
    mom1_y = (yc * cutout_sub).sum() / total
    mom2_x = ((xc - mom1_x)**2 * cutout_sub).sum() /
        total
    mom2_y = ((yc - mom1_y)**2 * cutout_sub).sum() /
        total

    sigma_x, sigma_y = mom2_x**0.5, mom2_y**0.5
    sigma_to_fwhm = np.sqrt(8 * np.log(2))
    fwhm_x, fwhm_y = sigma_x * sigma_to_fwhm, sigma_y *
        sigma_to_fwhm
    fwhm = (fwhm_x * fwhm_y)**0.5

    x_centroid = (x_int - half) + mom1_x
    y_centroid = (y_int - half) + mom1_y
```

```
return fwhm, x_centroid, y_centroid
```

For the photometry analysis function, I utilized parts of the `photutils.aperture` package to make the analysis more efficient. Specifically, I used the following imports: *CircularAperture*, *CircularAnnulus*, *ApertureStats*, and *aperture_photometry*. For each object, I create a Circular Aperture centered around the object coordinates and with a radius of 2x the calculated FWHM, as well as a Circular Annulus with an inner radius of 3x the FWHM and outer radius of 5x the FWHM. The *ApertureStats* function is used to analyze the annulus, allowing the median and standard deviation of the background sky to be stored. Next, the *aperture_photometry* function stores the aperture sum and the aperture area. The net flux is calculated by subtracting the median sky times the aperture area from the net aperture sum. The error is propagated in quadrature by taking the square root of the raw error squared plus the sky standard deviation squared scaled by the aperture area. The values for net flux, net error, background sky, and background sky standard deviation are returned:

```
def measure_aperture(image, noise, x, y, fwhm):
    pos = (x, y)
    src_ap = CircularAperture(pos, r=2 * fwhm)
    sky_an = CircularAnnulus(pos, r_in=3 * fwhm, r_out=5
                             * fwhm)

    sky_stats = ApertureStats(image, sky_an,
                              sigma_clip=SigmaClip(sigma=3.0))
    sky_per_pix = sky_stats.median
    sky_std = sky_stats.std

    phot = aperture_photometry(image, src_ap,
                               error=noise)
    raw_sum = phot['aperture_sum'][0]
    raw_err = phot['aperture_sum_err'][0]

    ap_area = src_ap.area
    net_flux = raw_sum - sky_per_pix * ap_area
    net_err = np.sqrt(raw_err**2 + ap_area * sky_std**2)

    return net_flux, net_err, sky_per_pix, sky_std
```

Using the stats from aperture photometry, I then created two other functions to first compute the zero points and then convert instrumental magnitudes to apparent magnitudes. To compute the zeropoints I use the three reference stars and associated exposure times and use the following relationship to return the zeropoints of each star:

$$ZP_i = m_{\text{cat},i} + 2.5 \log_{10} \left(\frac{F_i}{t_{\text{exp}}} \right) \quad (4)$$

where $m_{\text{cat},i}$ is the catalog magnitude of reference star i (from the Vizier query), F_i is the net flux of that star (from the aperture photometry above), and t_{exp} is the

exposure time of the stack. The mean and standard deviation of these three zero points are then stored and returned.

Then, the calibrate function is defined such that the instrumental magnitude and instrumental error are calculated using the following definitions:

$$m_i = -2.5 * \log_{10} \left(\frac{F_i}{t_{exp}} \right) \quad (5)$$

$$\sigma_i = \left(\frac{2.5}{\ln(10)} \right) * \left(\frac{\sigma_F}{F} \right) \quad (6)$$

Where *Equation 6* is derived from the error propagation formula:

$$\begin{aligned} m &= -2.5 \log_{10} \left(\frac{F}{t_{exp}} \right) \\ m &= -\frac{2.5}{\ln 10} \ln \left(\frac{F}{t_{exp}} \right) \\ \sigma_m &= \left| \frac{dm}{dF} \right| \sigma_F \\ \frac{dm}{dF} &= -\frac{2.5}{\ln 10} \frac{d}{dF} \ln \left(\frac{F}{t_{exp}} \right) \\ \frac{dm}{dF} &= -\frac{2.5}{\ln 10} \frac{1}{F} \\ \sigma_m &= \left| -\frac{2.5}{F \ln 10} \right| \sigma_F \\ \sigma_m &= \frac{2.5}{\ln 10} * \frac{\sigma_F}{F} \end{aligned}$$

To convert from the instrumental magnitude to the apparent magnitude, I simply added the zero point of the filter to the instrumental magnitude. The error was also propagated in quadrature to the apparent magnitude calculation:

```
def compute_zp(refs, exptime):
    zps = [cat_mag + 2.5 * np.log10(flux / exptime) for
           flux, cat_mag in refs]
    zp_mean = np.mean(zps)
    zp_std = np.std(zps, ddof=1)
    return zp_mean, zp_std

def calibrate(flux, flux_err, exptime, zp_mean, zp_std):
    instr_mag = -2.5 * np.log10(flux / exptime)
    instr_err = (2.5 / np.log(10)) * (flux_err / flux)
    calib_mag = instr_mag + zp_mean
    calib_err = np.sqrt(instr_err**2 + zp_std**2)
    return calib_mag, calib_err, instr_mag
```

To bring all of these individual functions together, I then created one final photometry function which takes in the stacked image path, the corresponding noise frame path, the exposure time, and the filter of the image. The function begins by calling the load_stack function

described earlier, which returns the image, header, and WCS information from the plate-solved FITS file. Since the header itself is not needed in this function, I ignore it using an underscore. The noise frame is then loaded separately using fits.getdata, since only the pixel values of the noise image are needed.

The first major step in this function is to call display_and_select on the stacked image. This returns a dictionary where each key is the role of the selected source, such as *target*, *ref1*, or *nonref1*, and each value is the clicked pixel coordinate. I then create a new empty dictionary called measurements which is used to store all of the relevant information for each selected object as the function progresses. This allows the function to keep adding new measurements to the same object rather than creating separate arrays for position, flux, magnitude, and error.

The function then loops over each selected source. For every role and pixel coordinate, I use the WCS solution to convert the clicked x and y pixel values into RA and DEC coordinates. If the selected object is the target quasar, I set the catalog magnitude and catalog ID to None because the target is not used to compute the zero point. For every other object, I call the query_catalog function using the RA, DEC, and filter band. I also included a while loop here as a safety check. If the Vizier query does not return a catalog magnitude, the code prints a message asking the user to re-click that source, calls display_and_select again for only that one role, and repeats the catalog query. This ensures that all reference and non-reference stars have valid catalog magnitudes before the pipeline continues.

Once the positions and catalog information are stored, the function loops through the measurements dictionary again to measure the FWHM of each object. For each star, I pass the image and original clicked coordinates into measure_fwhm, which returns the FWHM and centroided x and y positions. These new values are added back into that object's dictionary using the update method. After every object has a measured FWHM, I collect the FWHM values into a list and calculate the mean and standard deviation. The mean FWHM is then used for all of the aperture measurements in that image so that the aperture and annulus sizes are consistent across the target, reference stars, and test stars.

The next loop performs the actual aperture photometry. For each object, the code calls measure_aperture using the image, noise frame, centroided coordinates, and mean FWHM. This returns the net flux, flux error, median sky value, and sky standard deviation. I also round the centroid position to the nearest pixel and take the

maximum value in a small 5 by 5 pixel box around the source. This peak value is stored as a simple saturation diagnostic. These measured quantities, along with the aperture radius defined as two times the mean FWHM, are again added into the same dictionary entry for that object.

After measuring the fluxes, I create a list of tuples containing the fluxes and catalog magnitudes for the three reference stars. This list is passed into `compute_zp` along with the exposure time to calculate the mean zeropoint and its standard deviation. The function then loops over every measured object one final time and calls `calibrate` using that object’s flux, flux error, exposure time, and the calculated zeropoint values. This adds the instrumental magnitude, calibrated magnitude, and calibrated magnitude error to each object’s stored measurements.

At the end of the function, I print several diagnostic values to the terminal. These include the mean FWHM, zeropoint, calibrated target magnitude, and the comparison between the derived and catalog magnitudes of the non-reference stars. This final comparison is important because the non-reference stars were not used to calculate the zeropoint, so they provide a check on whether the reference-star calibration is reasonable.

Finally, I define a list of column names and create an output CSV filename by replacing the `.fits` extension of the stacked image with `_photometry.csv`. Before writing the table itself, I write several commented header lines containing the stack name, filter, exposure time, mean FWHM, FWHM scatter, zeropoint, and zeropoint scatter. I then use Python’s csv writer to save one row for each object in the measurements dictionary. The function finishes by printing the CSV path and returning the full measurements dictionary along with the zeropoint mean and standard deviation.

The full photometry pipeline can be found in the *Appendix*.

One consequence of the interactive selection approach is that the reference stars were not held fixed across stacks. Since reference star brightness varies significantly between filters – a star that is bright and unsaturated in *r* may be near the noise floor in *B* – I selected the three best-available references independently for each stack based on flux and saturation at the time of clicking. This means that a traditional differential lightcurve (target flux divided by the sum of reference fluxes over time) is not well-defined here, because the denominator changes composition between stacks. More importantly, the science case for this lab does not require differential photometry in the usual sense: with only two epochs of stacked data (one per night per filter), there is no within-night time series to track. The reference

stars serve purely as calibrators to establish the per-stack zeropoint, and the target variability is measured as a change in absolute calibrated magnitude between epochs rather than as a ratio to a common set of references. The stability of this calibration is validated by the sanity-check residuals on the three non-reference stars reported in Table 8.

2.7. Sky Brightness and Airmass

For each reduced stack, the sky background level was measured by taking the median pixel value of a source-free region of the image after dark subtraction. This count-per-pixel value was converted to a surface brightness in magnitudes per square arcsecond using the photometric zeropoint computed for that stack and the quoted plate scale of 0.94 arcsec/pix:

$$\mu_{\text{sky}} = ZP - 2.5 \log_{10} \left(\frac{S_{\text{sky}}}{t_{\text{exp}} \cdot s^2} \right) \quad (7)$$

where S_{sky} is the median sky in counts, t_{exp} is the exposure time, and $s = 0.94$ arcsec/pix is the plate scale. The airmass at mid-exposure for each stack was computed from the target coordinates, the DATE-OBS header, and the Rosemary Hill Observatory site (29.4°N, 82.3°W) using `astropy.coordinates`, since the FITS headers do not contain AIRMASS or OBJCTALT. The resulting sky brightness and zeropoint values are reported in Section 3.

2.8. SED Fitting

Using the calculated apparent magnitudes and associated errors, I am now able to use the two nights of data to look for variability. Ultimately, the goal of this lab is to interpolate the 2500 Å UV flux for two different nights and observe variability. Since it is impractical to directly measure the monochromatic flux using broadband filters, I can use the known power-law relationship for quasar spectral energy distributions to perform a computational regression and interpolation of the 2500 Å flux. To perform the regression, I first needed to convert the magnitude to spectral flux density with units of Janskys. The calculation for this is different depending on the zero point of the magnitude system used, which changes between the Vega system used by the *B* filter and the AB system used by the *r* and *i* filters. I opted to use a known offset for the Johnson-Cousins *B* filter to convert from the Vega system to the AB system:

$$M_{B_{AB}} = M_B - 0.125 \quad (8)$$

Then, I used the following equation to convert each magnitude to spectral energy density:

$$F_\nu = 3631 * 10^{\left(\frac{m}{-2.5}\right)} \quad (9)$$

I also converted the effective wavelengths for each filter to frequencies by dividing the speed of light constant by each effective wavelength. The error propagation between units was relatively simple, and only took a scaling of $\frac{1}{2.5}$ to convert to error in the log-flux space. To take the power law regression, I found it easier to do a linear fit in log space. Because of this, I took the log of all of the flux and frequency values as well as computed weights for each $\log(\text{flux})$ value by taking the inverse of the log error for each measurement. I then used numpy's polyfit function with these weights to measure the slope, intercept, and covariance matrix for each night of data. Essentially, this code takes the 2-3 measurements of different filters for each night and takes a best-fit line weighted by the inverse of the uncertainty of each measurement. Since this is done in log space, the resulting curve is a power-law spectral energy density (SED) once converted back out of log space:

$$\log_{10}(F_\nu) = m * \log_{10}(\nu) + b \quad (10)$$

$$F_\nu = 10^b * \nu^m \quad (11)$$

To then estimate the rest-frame 2500 Å flux, I first redshifted the rest-frame wavelength into the observed frame using the known redshift of the quasar ($z = 1.512$):

$$\nu_{2500, \text{obs}} = \frac{c}{(1+z)\lambda_{2500}} \quad (12)$$

Evaluating the best-fit line at $\log_{10}(\nu_{2500, \text{obs}})$ gives the predicted log-flux at that frequency. The uncertainty on this interpolated value is calculated using the covariance matrix returned by *numpy.polyfit*, which captures both the variance of the slope and intercept individually as well as their correlation. Since the prediction at a fixed $x = \log_{10}(\nu_{2500, \text{obs}})$ is a linear combination of the fit parameters ($y = mx + b$), the variance follows the standard linear-combination formula:

$$\sigma_{\log F_\nu}^2 = x^2 \text{Var}(m) + \text{Var}(b) + 2x \text{Cov}(m, b) \quad (13)$$

where $\text{Var}(m)$ and $\text{Var}(b)$ are the diagonal entries of the covariance matrix, and $\text{Cov}(m, b)$ is the off-diagonal entry. Without the covariance term, the uncertainty would be overestimated, since the slope and intercept of a least-squares fit are anti-correlated when the lever arm of the data is offset from $x = 0$ (which is the case here, since $\log_{10}(\nu)$ is on the order of 14.5–14.9). Including the covariance properly accounts for this correlation.

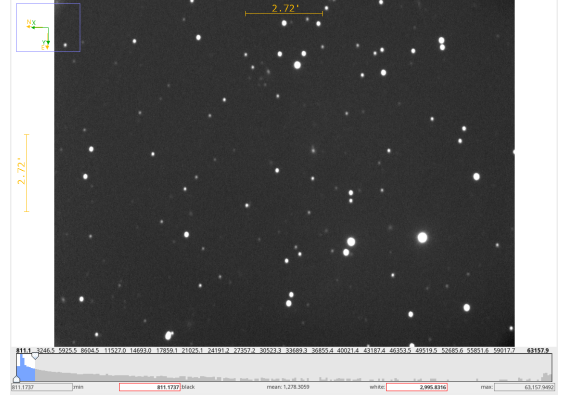


Figure 2. Solved and stacked frame of the target in the R band; taken on 4/14/26.

Finally, to convert the uncertainty from log-flux space back to linear flux space, I used the same first-order error propagation as in Equation 6, but applied to the inverse relationship $F_\nu = 10^{\log_{10}(F_\nu)}$:

$$\frac{dF_\nu}{d(\log_{10} F_\nu)} = 10^{\log_{10}(F_\nu)} \ln(10) = F_\nu \ln(10)$$

$$\sigma_{F_\nu} = F_\nu \ln(10) \sigma_{\log F_\nu}$$

which gives the final uncertainty on the interpolated 2500 Å flux for each night. These final values were plotted with their errors against the best fit line and the measured magnitudes from each night.

3. ANALYSIS

Using the python pipelines outlined in *Methods*, the ultimate goal was to take the science and calibration images and return interpolated values for the 2500 Å flux to analyze the variability of SDSS J081331.28+254503.0 in the UV. Table 2 lists all master calibration frames produced by the pipeline for each night, and Table 3 lists the five reduced science stacks that serve as inputs to the photometry pipeline.

Each science stack was plate-solved using the *astrometry.net* web interface prior to the photometry run. The plate-solved output strips the *EXPTIME* and *FILTER* header keywords, so both values were passed explicitly via the *stacks* list in *photometry.py* rather than read from the fits header.

The *data_reduction.py* pipeline was run for all 5 observations, resulting in 5 final stacked frames. Figures 2–5 show example reduced science frames and master calibration frames that came from this pipeline. It is

Table 2. Master calibration frames produced by the reduction pipeline for each night. All frames are stored under `masters/J0813-{night}/` relative to the lab directory.

Night	Type	Exp / Filter	Filename
2026-04-14	Bias	0.001 s	<code>master_bias.fits</code>
2026-04-14	Dark	1 s	<code>master_dark.1.0s.fits</code>
2026-04-14	Dark	1.3 s ^a	<code>master_dark.1.3s.fits</code>
2026-04-14	Dark	4 s	<code>master_dark.4.0s.fits</code>
2026-04-14	Dark	24 s	<code>master_dark.24.0s.fits</code>
2026-04-14	Dark	300 s	<code>master_dark.300.0s.fits</code>
2026-04-14	Flat	<i>B</i>	<code>master_flat.B.fits</code>
2026-04-14	Flat	<i>r</i>	<code>master_flat.R.fits</code>
2026-04-14	Flat	<i>i</i>	<code>master_flat.I.fits</code>
2026-04-23	Bias	0.001 s	<code>master_bias.fits</code>
2026-04-23	Dark	4 s	<code>master_dark.4.0s.fits</code>
2026-04-23	Dark	24 s	<code>master_dark.24.0s.fits</code>
2026-04-23	Dark	70 s	<code>master_dark.70.0s.fits</code>
2026-04-23	Dark	180 s	<code>master_dark.180.0s.fits</code>
2026-04-23	Dark	240 s	<code>master_dark.240.0s.fits</code>
2026-04-23	Flat	<i>B</i> ^b	<code>master_flat.B.fits</code>
2026-04-23	Flat	<i>r</i>	<code>master_flat.R.fits</code>
2026-04-23	Flat	<i>i</i>	<code>master_flat.I.fits</code>

^aThese four 1.3 s darks existed on disk but were flagged as “not taken” in the observing log and do not correspond to any science or flat exposure time. The master was produced by the pipeline but was not applied to any frame.

^bThe 2026-04-23 *B*-band master flat exhibited an anomalously wide normalized pixel range (min -0.73 , max 1.75), likely caused by a bad pixel or unmasked cosmic ray in the flat stack. This flat was not applied to any science frame since no *B*-band data was acquired on 2026-04-23.

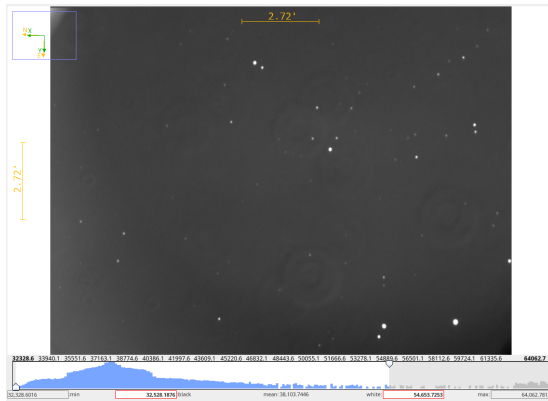


Figure 3. Solved and stacked frame of the target in the R band; taken on 4/23/26. Noticeably, several dust rings are still visible in this reduced frame.

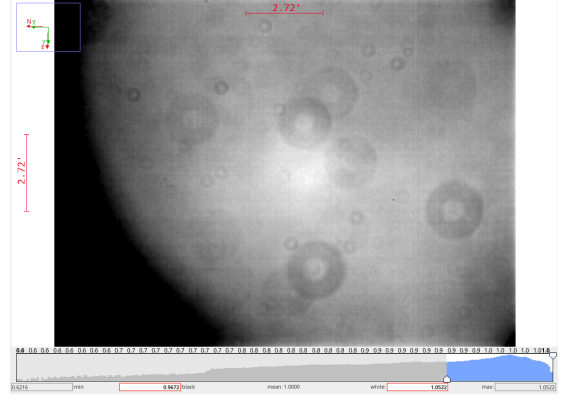


Figure 4. Master flat in the R filter; taken on 4/23/26.

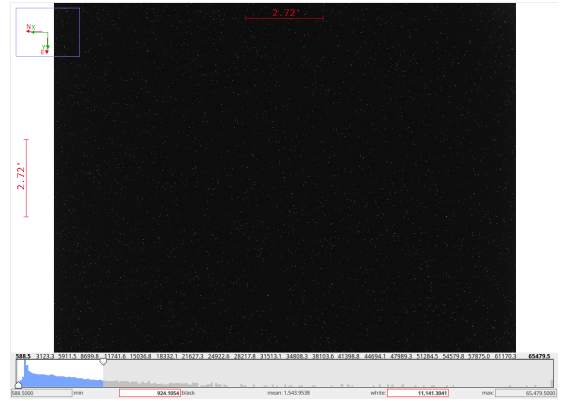


Figure 5. 180 second master dark frame; taken on 4/23/26.

noticeable that in *Figure 3*, there are still dust rings visible despite the flat frame division supposedly being done correctly. To better understand this, it is important to note the observational context of the data. Based on the altitude plots in *Figures 6-7*, it is shown that the angular separation between the target and the moon was less than 10° at the time of observation. The result of this is a very bright background sky, as seen in the histogram for *Figure 3*.

The belief is that because of the extreme background sky and unevenly illuminated field for the science frames taken on 4/23, the flat division was not fully adequate to remove the extra gradient and dust rings caused by the moon’s proximity. Nevertheless, the object still appears to be visible in these frames, so aperture photometry was still done to analyze the object’s magnitude. It was assumed, however, that due to the much lower SNR of this night’s data, I would likely be unable to achieve statistically significant variability detection. Despite this, the proposal is a proof-of-concept for the ability of RHO-sized telescopes to observe and constrain the variability of quasars in the rest-frame UV, and even one night of

Table 3. Reduced science stacks used for photometry. Each stack is the median combination of the frames listed. Median counts are measured from the final stacked frame prior to plate-solving.

Night	Filter	Exp (s)	Frames	Filename	Median (ct)
2026-04-14	<i>B</i>	300	3	J0813_2026-04-14_B_300.0_stack.fits	142
2026-04-14	<i>r</i>	300	3	J0813_2026-04-14_R_300.0_stack.fits	1,235
2026-04-14	<i>i</i>	300	3	J0813_2026-04-14_I_300.0_stack.fits	791
2026-04-23	<i>r</i>	180	6	J0813_2026-04-23_R_180.0_stack.fits	38,082
2026-04-23	<i>i</i>	300	3 ^a	J0813_2026-04-23_I_300.0_stack.fits	32,014

^aTwo additional *i*-band frames acquired on 2026-04-23 (one at 180s, one at 240s) were excluded from the stack to maintain a uniform 300s effective exposure time. Only the three 300s frames were median-combined.

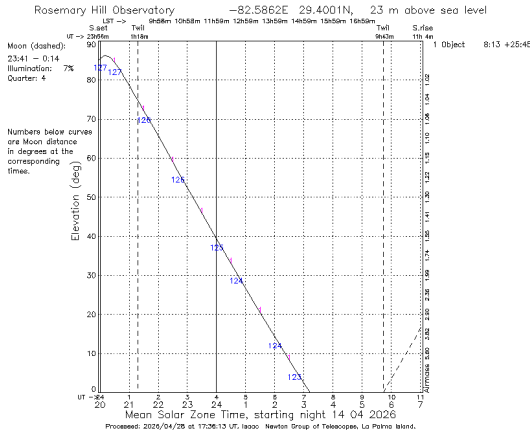


Figure 6. Altitude plot of SDSS J081331.28+254503.0 for RHO on 4/14/2026

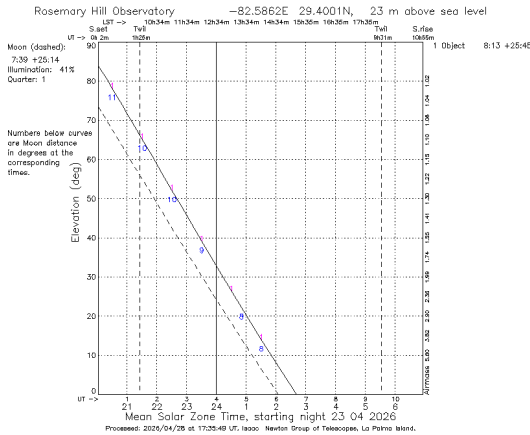


Figure 7. Altitude plot of SDSS J081331.28+254503.0 for RHO on 4/23/2026. The numbers in blue represent the angular separation between the object and the moon.

good observations could yield insightful results about the precision achievable for this magnitude of object.

Table 4. Detector characterization from master bias and dark frames. GAIN = $0.37e^-/\text{count}$; CCD temperature $\approx 0^\circ\text{C}$ both nights. Dark current reported at the science-frame exposure time.

Night	Read Noise (ct/pix)	Read Noise (e^-/pix)	Dark Current (ct/pix/s)	Dark Current ($e^-/\text{pix/s}$)
2026-04-14	49.7	18.4	0.986	0.365
2026-04-23	51.1	18.9	1.011	0.374

Table 5. Sky background per science stack. $\bar{S}_{\text{sky}}$ is the mean of the sigma-clipped annulus medians across all seven selected stars; σ_{sky} is the mean per-pixel annulus scatter. μ_{sky} is converted via Equation 7 using $s = 0.94 \text{ arcsec/pix}$.

Night	Filter	$\bar{S}_{\text{sky}}$ (ct/pix)	σ_{sky} (ct/pix)	μ_{sky} (mag/arcsec ²)
2026-04-14	<i>B</i>	142.8	39.5	21.05 ± 0.30
2026-04-14	<i>r</i>	1239.9	48.2	20.18 ± 0.04
2026-04-14	<i>i</i>	799.9	45.6	19.66 ± 0.06
2026-04-23	<i>r</i>	38,083.7	171.1	15.91 ± 0.01
2026-04-23	<i>i</i>	32,149.1	195.7	15.60 ± 0.01

NOTE—The $\sim 4\text{--}5 \text{ mag/arcsec}^2$ brightening between nights in both *r* and *i* is consistent with the Moon being $\sim 10^\circ$ from the target at 41% illumination on 2026-04-23, versus below the horizon on 2026-04-14.

Using the reduced frames, I then used astrometry.net to plate-solve and save the WCS information using the new_image.fits file. These new files were then used for the photometry pipeline, and the file paths were hard-coded into the *stacks* array. The photometry pipeline

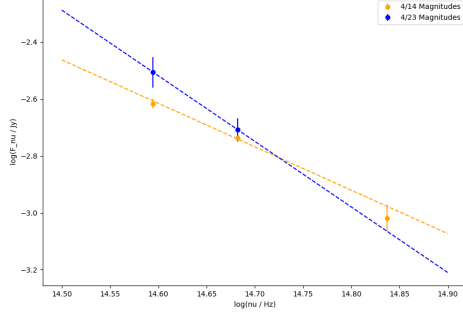


Figure 8. Plot of $\log(F_\nu)$ vs. $\log(\nu)$ with errorbars

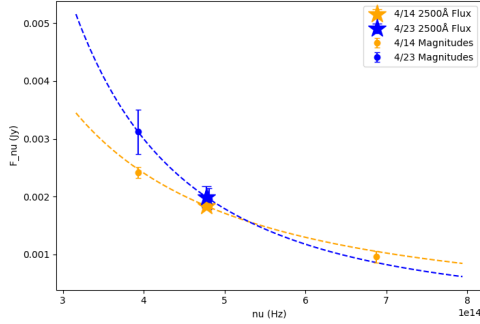


Figure 9. Plot showing the measured magnitude values for 4/14 (orange circles) and 4/23 (blue circles), power-law best-fit regression lines (dashed lines), and interpolated values for the 2500AA F_ν (stars).

created a csv file for each stacked frame consisting of a list of all of the information calculated and defined in the pipeline. Table 6 shows an example of the photometry output for the 2026-04-23 *r* stack. For brevity, a summary of the key photometry quantities for all five stacks is given in Table 7, and the sanity-check comparison of derived versus catalog magnitudes for the three non-reference stars per stack is given in Table 8. The complete CSV files for all five stacks are submitted separately as supplementary material.

Notably, the *r*-band zeropoint is nearly identical between the two nights (21.858 vs. 21.859), despite the dramatically different sky conditions, which indicates that the photometric calibration is stable and not significantly affected by the moon-driven background.

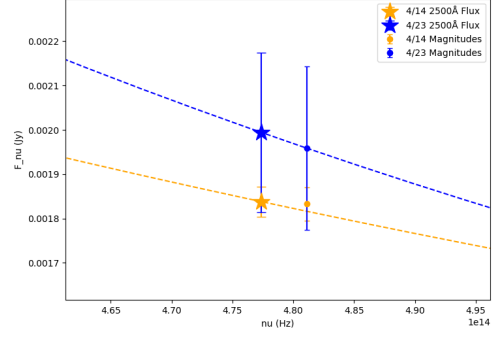


Figure 10. Cropped plot showing the measured F_{nu} values for the *r* filter (circles), power-law best-fit regression lines (dashed lines), and interpolated values for the 2500AA F_{nu} (stars).

Looking at Table 5, I find very informative results in the σ_m column. The standard deviation of the B frame on 2026-04-14 and *r/i* frames on 2026-04-23 all have values slightly above 0.1 magnitudes. This value is quite high, and is close to the proposed variability of 0.1–0.2 magnitudes, meaning that, alone, these values would not be sufficient to detect any real variability. However, the *r* and *i* frames from 2026-04-14 have values of 0.022 and 0.043, respectively. Both values are well below the proposed variability and when combined through SED fitting with multiple filters could allow for scientifically useful measurement of the variability.

After inserting the magnitude and magnitude error values for each frame into the `sed_fitting.py` pipeline, I was able to perform a linear regression in log space along with an interpolation of the 2500 Å band. The results of this regression are shown in Figure 8. After converting all of the measurements and regression back into a power-law fit, the interpolated values and associate errors of the 2500AA band were plotted (see Figures 9-10).

Using the calibrated apparent magnitudes from Table 7, the power-law SED fitting described in *Methods* was applied to each night’s data to interpolate the rest-frame 2500 Å flux. At $z = 1.512$, rest-frame 2500 Å corresponds to an observed-frame wavelength of $\lambda_{\text{obs}} = (1+z) \times 2500 = 6280$ Å, or $\nu_{\text{obs}} = 4.774 \times 10^{14}$ Hz. Table 9 summarizes the best-fit spectral slopes, interpolated fluxes, and derived apparent magnitudes for each night, along with the night-to-night variability amplitude and its detection significance.

The detection significance of 0.88σ is quite low, unfortunately, and is due mainly to the large error for the 2026-04-23 data, which has an error of 0.098 as opposed to the error of 0.020 for 2026-04-14. From this data

Table 6. Photometry pipeline output for the 2026-04-23 r -band stack ($t_{\text{exp}} = 180\text{s}$, ZP = 21.859 ± 0.061 , mean FWHM = $6.09 \pm 1.25\text{ pix}$).

Role	Cat. Mag	Catalog	Net Flux (ct)	Sky (ct/pix)	m_{inst}	m_{calib}	σ_m
target	...	...	53,797	38,228	-6.189	15.670	0.102
ref1	13.538	II/349/ps1	389,959	37,298	-8.339	13.520	0.062
ref2	14.042	I/322A/out	252,411	38,329	-7.867	13.992	0.065
ref3	14.027	I/322A/out	229,422	38,254	-7.763	14.096	0.065
nonref1	14.753	I/322A/out	113,116	38,164	-6.996	14.863	0.072
nonref2	14.040	I/322A/out	247,569	38,117	-7.846	14.013	0.064
nonref3	13.553	II/349/ps1	410,023	38,195	-8.394	13.465	0.062

Table 7. Summary of photometry pipeline results for all five science stacks.

Night	Filter	Exp (s)	m_{target}	σ_m	ZP	σ_{ZP}	FWHM (pix)	σ_{FWHM}
2026-04-14	B	300	16.571	0.119	20.379	0.016	6.28	1.70
2026-04-14	r	300	15.742	0.022	21.858	0.016	5.88	0.14
2026-04-14	i	300	15.442	0.043	20.856	0.034	5.64	0.36
2026-04-23	r	180	15.670	0.102	21.859	0.061	6.09	1.25
2026-04-23	i	300	15.165	0.135	20.806	0.085	6.58	1.89

Table 8. Sanity-check comparison of derived magnitudes against catalog values for the three non-reference stars in each stack. $\Delta m = m_{\text{derived}} - m_{\text{catalog}}$.

Night	Filter	Star	Catalog	m_{derived}	m_{catalog}	Δm
2026-04-14	B	nonref1	I/322A/out	14.483	14.537	-0.054
		nonref2	I/322A/out	14.646	14.708	-0.062
		nonref3	I/322A/out	13.037	13.019	+0.018
2026-04-14	r	nonref1	I/322A/out	13.909	13.944	-0.035
		nonref2	I/322A/out	13.331	13.366	-0.035
		nonref3	I/322A/out	12.169	12.181	-0.012
2026-04-14	i	nonref1	I/322A/out	11.952	11.893	+0.059
		nonref2	I/322A/out	13.585	13.545	+0.040
		nonref3	I/322A/out	13.099	12.940	+0.159 ^a
2026-04-23	r	nonref1	I/322A/out	14.863	14.753	+0.110
		nonref2	I/322A/out	14.013	14.040	-0.027
		nonref3	II/349/ps1	13.465	13.553	-0.088
2026-04-23	i	nonref1	II/349/ps1	12.844	12.954	-0.110
		nonref2	II/349/ps1	13.832	13.948	-0.116
		nonref3	II/349/ps1	15.053	15.082	-0.029

^aLargest single residual in the dataset; likely from the star's own variability rather than a calibration failure.

Table 9. SED fitting results and interpolated rest-frame 2500 Å flux for each night. The spectral slope α is defined by $F_\nu \propto \nu^\alpha$. $\Delta m_{2500} = m_{2500,n2} - m_{2500,n1}$.

Night	Filters	α	$F_{2500} \pm \sigma_F$ (mJy)	$m_{2500} \pm \sigma_m$
2026-04-14	B, r, i	-1.527	1.838 ± 0.034	15.739 ± 0.020
2026-04-23	r, i	-2.306	1.994 ± 0.180	15.651 ± 0.098
Δm_{2500}				-0.089 ± 0.100
Detection significance				0.88σ

NOTE—The night 2 fit uses only two photometric points (r and i); the fit is therefore exact with zero degrees of freedom and no internal goodness-of-fit check. The uncertainties for night 2 reflect propagation of the input magnitude errors only.

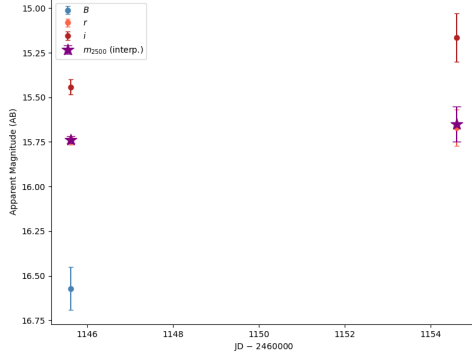


Figure 11. Plot of the observed magnitudes (circles) and interpolated magnitudes (stars) against the Julian Date.

alone, the detection significance is not high enough to claim that I have captured variability, however, based on the tight errors for the first night of observations – 2026-04-14 – I believe that it is possible to observe true variability with repeated measurements of this target with RHO. The close proximity of the moon to the target during the second night of observations is an extreme observational circumstance and was mainly a result of the strict time constraint that came with this lab. However, over a longer observational campaign I would expect to yield errors more similar to those of the 2026-04-14 measurements: around ± 0.02 magnitudes. The period of the target’s variability is not directly measurable from these two measurements, either, and would require a longer observational campaign with more regular observations to be determined.

The F_ν values and errors were converted back into magnitudes and graphed vs. day (see Figure 11) to

show another visualization of the variability of the object across filters.

4. ERROR ANALYSIS

This lab required the proper propagation of errors throughout all of the calculations. This was done following basic error formulas for the addition, subtraction, multiplication, and division for the data reduction pipeline, identical to what was done in Lab 4. Of note, in the stacking procedure I used the median pixel value for combination, yet propagated the error as if it was a mean combination. These lead to slightly different values, but are a decent approximation.

The major source of error in this lab came from the low magnitude of the background sky on 2026-04-23 caused by the close proximity to the moon. According to the altitude plot, the angular separation between the object and the moon was below 10° and dropping, with the illumination at 41%. This is an extremely low separation, and resulted in the already dim object being much harder to resolve above the background floor. The results of this is clear in Figure 10, where the error bars for 4/23 are significantly larger than those for 4/14.

Another source of error comes from the SED fitting procedure. While it is commonly cited that quasar SEDs in the UV follow a power-law spectrum, I did not take into account the pollution of the observed fluxes by emission lines. Since I am using broadband photometry, the total flux received by the detector is integrated over the passthrough curves for each specific filter which span 100’s of Angstroms. This means that it is very likely that we overestimated the magnitude of the 2500Å line as we included non-blackbody emission-line flux in some/all of our measurements. In the future, these effects could be mitigated through more rigorous reduction and SED-fitting techniques, as outlined in Shen et al. (2011).

5. ANSWERING SPECIFIC QUESTIONS

Question: State what the phase of the moon was during your observations. How close was each target to the moon during the observations?

Answer: On 2026-04-14, the moon was below the horizon at the time of observation, contributing negligible background sky flux and allowing for clean photometry in all three filters. On 2026-04-23, the angular separation between the target and the moon was under 10° , with the illumination at 41%, as shown in the altitude plots in Figures 6–7. This proximity was the dominant source of increased sky background noise on the second night.

Question: Determine the error on your zero-point by measuring it for at least 3 stars per filter. Is there a clear

relation between the zeropoint and airmass? Comment on what you observe in the zeropoint as a function of time.

Answer: The zeropoint error was determined from the star-to-star scatter across the three reference stars per stack, as reported in Table 7. The r -band zeropoint is nearly identical between the two nights (21.858 vs. 21.859), despite the dramatically different sky conditions, which indicates that the photometric calibration is stable and not significantly affected by the moon-driven background. Without multiple observations in each filter over the course of one night, a per-frame airmass trend cannot be resolved from the two-epoch stacked dataset; however, the consistency of the r -band ZP across two nights with very different airmasses and sky conditions suggests no strong extinction trend was present.

Question: Is there a significant difference between the overall sky brightness on different observing nights? If so, what might have caused the difference? Do you see any trends in sky brightness as a function of azimuthal angle or airmass?

Answer: There is a significant difference in sky brightness between the two nights. The r -band sky background increased from $\sim 1,240$ ct/pix on 2026-04-14 to $\sim 38,084$ ct/pix on 2026-04-23, a brightening of $\sim 4\text{--}5$ mag/arcsec² in both r and i (Table 5). This is consistent with the Moon being $\sim 10^\circ$ from the target at 41% illumination on 2026-04-23, versus below the horizon on 2026-04-14. Sky brightness versus azimuth and airmass could not be plotted for this dataset because only one stacked frame per filter per night was available, providing a single sky measurement per stack rather than a time-resolved series. The dominant variation in sky brightness is therefore the night-to-night difference driven by moon proximity.

Question: Use your finder chart to determine which of the stars in your science images is your variable target. What makes you sure that you have identified the correct star?

Answer: The target quasar was identified using two independent methods. First, I used the SIMBAD finderchart to identify the pattern of nearby stars in the field and visually locate the quasar. Second, I used the WCS solution from the plate-solved frame to convert the published SIMBAD coordinates of SDSS J081331.28+254503.0 to pixel coordinates and confirm that the visually selected source was consistent with the catalog position.

Question: Please comment qualitatively on your data: do you detect variation? Was it the variation you expected to detect? Are there artifacts in your data that

worry you? What could they potentially be caused by? How did the observing conditions impact your results? Can your data be used to answer the science questions that motivated the writing of your proposal?

Answer: The detection significance of 0.88σ means that from these two nights alone, I cannot claim a confirmed detection of variability. The measurement errors on the 2026-04-14 r and i bands are 0.022 and 0.043 magnitudes respectively, both well below the proposed variability amplitude of 0.1–0.2 magnitudes. However, the 2026-04-23 data is significantly degraded by the bright background sky caused by the close moon proximity, resulting in errors of 0.102 and 0.135 magnitudes. Artifacts visible in the 2026-04-23 reduced frames include residual dust rings not fully removed by flat division, likely because the moon-contaminated sky illuminated the focal plane differently than the dome flat panel. The first night’s precision demonstrates that RHO is capable of achieving the photometric accuracy needed to detect quasar variability; the second night’s conditions were not adequate for a statistically significant detection given the small expected variability amplitude.

Question: Is the variation periodic? What is its timescale or duration? What is the uncertainty on that duration or period?

Answer: The period of the target’s variability is not directly measurable from only two observations separated by nine days. A longer observational campaign with more regular cadence would be required to characterize the variability timescale. Quasar UV variability typically occurs on timescales of weeks to months, so a campaign spanning at least one semester with more consistent moon geometry would be needed to constrain the period.

Question: What is the magnitude of your variable object at its peak brightness? What is its magnitude at the minimum brightness? What are the uncertainties on these measurements? Is this consistent with your expectations?

Answer: The interpolated rest-frame 2500 Å magnitude was 15.739 ± 0.020 on 2026-04-14 and 15.651 ± 0.098 on 2026-04-23 (Table 9). The brighter measurement (lower magnitude) occurred on 2026-04-23, with a night-to-night difference of $\Delta m_{2500} = -0.089 \pm 0.100$. This is broadly consistent with the expected 0.1–0.2 magnitude variability proposed in the observing proposal, but the error on the second night is too large to confirm the detection at meaningful significance.

Question: What is the uncertainty of each of your brightness measurements? Quantitatively, how sure is your detection of variation in this star?

Answer: The per-epoch uncertainties on the interpolated 2500 Å magnitude are $\sigma = 0.020$ mag (2026-04-14) and $\sigma = 0.098$ mag (2026-04-23), combining photon noise, sky background noise, and zeropoint scatter in quadrature. The combined uncertainty on the night-to-night magnitude difference is $\sigma_{\Delta m} = 0.100$ mag. With a measured $|\Delta m| = 0.089$ mag, the detection significance is 0.88σ — not a statistically significant detection of variability.

6. CONCLUSIONS AND FUTURE WORK

In this lab, I carried out two nights of broadband photometric observations of SDSS J081331.28+254503.0 at RHO, with the goal of detecting rest-frame UV variability at 2500 Å using power-law SED fitting across the B , r , and i filters. The first night of observations on 2026-04-14 produced clean, well-calibrated photometry in all three filters, with measurement errors of 0.022 and 0.043 magnitudes in the r and i bands respectively — both well below the expected variability of 0.1–0.2 magnitudes. The second night on 2026-04-23 was significantly impacted by the moon being within $\sim 10^\circ$ of the target at 41% illumination, resulting in sky backgrounds nearly $30\times$ brighter and measurement errors of 0.102 and 0.135 magnitudes in the r and i bands. The interpolated rest-frame 2500 Å magnitude was 15.739 ± 0.020 on the first night and 15.651 ± 0.098 on the second night, giving a night-to-night difference of $\Delta m = -0.089 \pm 0.100$ at a detection significance of 0.88σ . From these two observations alone, I cannot claim a confirmed detection of variability — however, the photometric precision achieved on the first night demonstrates that RHO is capable of the accuracy needed to detect quasar variability at the proposed level, and I believe that a longer observational campaign would yield more conclusive results.

The most important improvement for future observations would be to plan around the lunar cycle and avoid moon proximity on any science night. The 2026-04-14 observations, taken when the moon was below the horizon, achieved photometric precision close to the proposed ~ 0.004 magnitudes for the r band, validating the feasibility of this type of observation with the RHO 14" telescope. A longer observational campaign spanning several months, with observations timed around the lunar cycle, would allow for a meaningful characterization of the variability timescale and amplitude of the target — something that is not possible from only two observations. It is also worth noting that the stability of the r -band zeropoint between the two nights (21.858 vs. 21.859) despite dramatically different sky conditions is an encouraging result, as it suggests that the photometric calibration approach is robust even un-

der poor conditions. Overall, even with only two nights of data and suboptimal conditions on the second night, this lab serves as a useful proof-of-concept of the ability for small-aperture telescopes like RHO to contribute to quasar UV variability studies at cosmologically relevant redshifts.

APPENDIX

A. DATA REDUCTION PIPELINE

```

import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
import astroalign as aa
import shutil
import os
import glob

night = "2026-04-23"
obj_name = f"J0813_{night}"
dark_paths = glob.glob(f"{night}/Darks/*.fits")
bias_paths = glob.glob(f"{night}/Biases/*.fits")
flat_paths = glob.glob(f"{night}/Flats/*.fits")
light_paths = glob.glob(f"{night}/Lights/*.fits")

def create_master_darks(dark_paths):
    darks = {}
    master_darks = {}
    dark_noise = {}

    for frame in dark_paths:
        header = fits.getheader(frame)
        exp = header['EXPTIME']
        if exp not in darks:
            darks[exp] = []
        darks[exp].append(frame)

    for exp, paths in darks.items():
        stack = np.array([fits.getdata(path) for path in paths])
        master_darks[exp] = np.median(stack, axis=0)
        dark_noise[exp] = np.std(stack, axis=0) / np.sqrt(len(stack))

    return master_darks, dark_noise

def create_master_bias(bias_paths):
    stack = np.array([fits.getdata(path) for path in bias_paths])
    master_bias = np.median(stack, axis=0)
    bias_noise = np.std(stack, axis=0) / np.sqrt(len(stack))
    return master_bias, bias_noise

def create_master_flats(flat_paths, master_darks, dark_noise, bias_paths):
    flats = {}
    master_flats = {}
    flat_noise = {}
    total_flat_noise = {}

    master_bias, bias_noise = create_master_bias(bias_paths)

    for frame in flat_paths:
        header = fits.getheader(frame)
        filter = header['FILTER']
        if filter not in flats:
            flats[filter] = []
            flat_noise[filter] = []
        flats[filter].append(frame)

    for filter, paths in flats.items():
        stack = []
        for path in paths:
            frame = fits.getdata(path)
            header = fits.getheader(path)
            exp = header["EXPTIME"]

```

```

    if exp in master_darks:
        calib_frame = master_darks[exp]
        calib_noise = dark_noise[exp]
    else:
        #If no exact darks are found for an exposure time, use the closest possible dark
        dark_exp = min(master_darks.keys(), key=lambda x: abs(x - exp))

        dark_current = (master_darks[dark_exp] - master_bias) / dark_exp
        dark_current_noise = np.sqrt(dark_noise[dark_exp]**2 + bias_noise**2) / dark_exp

        calib_frame = master_bias + dark_current * exp
        calib_noise = np.sqrt(bias_noise**2 + (exp * dark_current_noise)**2)

    flat_noise[filter].append(np.sqrt(frame + calib_noise**2))
    stack.append(frame - calib_frame)

    noise_stack = np.array(flat_noise[filter])
    total_flat_noise[filter] = np.sqrt(np.sum(noise_stack**2, axis=0)) / len(noise_stack)

    master_flat = np.mean(np.array(stack), axis=0)
    master_flats[filter] = master_flat / np.mean(master_flat)

    total_flat_noise[filter] = total_flat_noise[filter] / np.mean(master_flat)

return master_flats, total_flat_noise

def create_master_lights(light_paths, master_darks, dark_noise, master_flats, flat_noise, master_bias, bias_noise,
    obj_name):

    if os.path.exists(obj_name):
        shutil.rmtree(obj_name)
    os.mkdir(obj_name)
    lights = {}
    light_noise = {}
    calib_lights = {}

    #sort frames by filter and exposure time
    for path in light_paths:
        header = fits.getheader(path)
        filter = header['FILTER']
        exp = header['EXPTIME']

        if filter not in lights:
            lights[filter] = {}
            calib_lights[filter] = {}
            light_noise[filter] = {}

        if exp not in lights[filter]:
            lights[filter][exp] = []
            calib_lights[filter][exp] = []
            light_noise[filter][exp] = []

        lights[filter][exp].append(path)

    for filter, data in lights.items():
        for exp, paths in data.items():
            for path in paths:
                frame = fits.getdata(path)

                if exp in master_darks:
                    calib_frame = master_darks[exp]
                    calib_noise = dark_noise[exp]
                else:
                    #If no exact darks are found for an exposure time, use the closest possible dark
                    dark_exp = min(master_darks.keys(), key=lambda x: abs(x - exp))

                    dark_current = (master_darks[dark_exp] - master_bias) / dark_exp

```

```

dark_current_noise = np.sqrt(dark_noise[dark_exp]**2 + bias_noise**2) / dark_exp

calib_frame = master_bias + dark_current * exp
calib_noise = np.sqrt(bias_noise**2 + (exp * dark_current_noise)**2)

dark_sub = frame - calib_frame
dark_sub_noise = np.sqrt(frame + calib_noise**2)
calib_light = dark_sub / master_flats[filter]
calib_lights[filter][exp].append(calib_light)

shot_est = np.median((np.sqrt(frame) / master_flats[filter])[dark_sub > 0])
dark_est = np.median((calib_noise / master_flats[filter])[dark_sub > 0])
flat_est = np.median((np.abs(calib_light) * flat_noise[filter] / master_flats[filter])[dark_sub > 0])

print(f"{filter} exp={exp}: shot={shot_est:.3f}, dark={dark_est:.3f}, flat={flat_est:.3f}")

frac_dark = np.zeros_like(dark_sub, dtype=float)
frac_flat = np.zeros_like(master_flats[filter], dtype=float)

mask_dark = dark_sub > 0
mask_flat = master_flats[filter] > 0

frac_dark[mask_dark] = (dark_sub_noise[mask_dark] / dark_sub[mask_dark])**2
frac_flat[mask_flat] = (flat_noise[filter][mask_flat] / master_flats[filter][mask_flat])**2

calib_light_noise = np.abs(calib_light) * np.sqrt(frac_dark + frac_flat)
light_noise[filter][exp].append(calib_light_noise)

#Due to the large number of files, I will be using the astroalign package for alignment
aligned_lights = {}
aligned_noise = {}
master_lights = {}
total_light_noise = {}

for filter, data in calib_lights.items():
    aligned_lights[filter] = {}
    aligned_noise[filter] = {}

    #here I am choosing the first frame in each filter as the reference
    #only really matters if there are multiple exposure times for the same filter, which there shouldn't be.
    ref_exp = list(data.keys())[0]
    reference = data[ref_exp][0]

    for exp, frames in data.items():
        aligned_lights[filter][exp] = []
        aligned_noise[filter][exp] = []

        for frame_num, frame in enumerate(frames):
            noise = light_noise[filter][exp][frame_num]

            if exp == ref_exp and frame_num == 0:
                aligned_lights[filter][exp].append(frame)
                aligned_noise[filter][exp].append(noise)
            else:
                try:
                    transform, _ = aa.find_transform(frame, reference)
                    aligned_frame, footprint = aa.apply_transform(transform, frame, reference)
                    aligned_noise_frame, _ = aa.apply_transform(transform, noise, reference)

                    aligned_lights[filter][exp].append(aligned_frame)
                    aligned_noise[filter][exp].append(aligned_noise_frame)
                except:
                    print(f"Failed to find the transform for: {filter}, {exp}, {frame_num}.")

all_frames = []
all_noise = []

#Here, I am combining all of the different exposure times per filter into the final stacked frame

```

```

#Again, it shouldn't matter if the exposure times are all correct, but since I've allowed for
#differing exposure times in my pipeline, I will stay self-consistent

for exp in aligned_lights[filter]:
    all_frames.extend(aligned_lights[filter][exp])
    all_noise.extend(aligned_noise[filter][exp])

frame_stack = np.array(all_frames)
noise_stack = np.array(all_noise)

master_lights[filter] = np.median(frame_stack, axis=0)
total_light_noise[filter] = np.sqrt(np.sum(noise_stack**2, axis=0)) / len(noise_stack)
output_filename = f"{obj_name}/{obj_name}_{filter}_{ref_exp}_stack.fits"
noise_filename = f"{obj_name}/{obj_name}_{filter}_{ref_exp}_noise.fits"
fits.writeto(output_filename, master_lights[filter], overwrite=True)
fits.writeto(noise_filename, total_light_noise[filter], overwrite=True)
return master_lights, total_light_noise

masters_dir = f"masters/{obj_name}"
os.makedirs(masters_dir, exist_ok=True)

master_darks, dark_noise = create_master_darks(dark_paths)
master_bias, bias_noise = create_master_bias(bias_paths)
master_flats, flat_noise = create_master_flats(flat_paths, master_darks, dark_noise, bias_paths)
master_lights, total_light_noise = create_master_lights(light_paths, master_darks, dark_noise, master_flats,
    flat_noise, master_bias, bias_noise, obj_name)

fits.writeto(f"{masters_dir}/master_bias.fits", master_bias, overwrite=True)
for exp, dark in master_darks.items():
    fits.writeto(f"{masters_dir}/master_dark_{exp}.fits", dark, overwrite=True)
for filt, flat in master_flats.items():
    fits.writeto(f"{masters_dir}/master_flat_{filt}.fits", flat, overwrite=True)

read_noise = np.std(master_bias)
print(f"Calibration Stats: {obj_name}")
print(f"Read noise (std of master bias): {read_noise:.4f} counts/pixel")
for exp in sorted(master_darks.keys()):
    dark_current = np.mean((master_darks[exp] - master_bias) / exp)
    print(f"Dark current ({exp:6.3f}s dark): {dark_current:.6f} counts/pixel/sec")

```

B. PHOTOMETRY PIPELINE

```

import csv
import time
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
from astropy.wcs import WCS
from astropy.coordinates import ICRS
from astropy import units as u
from astropy.stats import SigmaClip
from photutils.aperture import CircularAperture, CircularAnnulus, ApertureStats, aperture_photometry
from astroquery.vizier import Vizier

SAT_THRESHOLD = 62000

target_cats = {
    'B': {'Bmag': ['I/322A/out', 'II/336/apass9']},
    'R': {'rmag': ['I/322A/out', 'II/349/ps1'],
        'r_mag': ['II/336/apass9']},
    'I': {'imag': ['I/322A/out', 'II/349/ps1'],
        'i_mag': ['II/336/apass9']},
}

priority_order = ['I/322A/out', 'II/336/apass9', 'II/349/ps1']

```

```

def sort_by_priority(cat_list):
    return sorted(cat_list, key=lambda c: priority_order.index(c) if c in priority_order else len(priority_order))

for band in target_cats:
    for col_name in target_cats[band]:
        target_cats[band][col_name] = sort_by_priority(target_cats[band][col_name])

def load_stack(path):
    with fits.open(path) as hdul:
        image = hdul[0].data
        header = hdul[0].header
        wcs = WCS(header)
    return image, header, wcs

def display_and_select(image, roles=None):
    if roles is None:
        roles = ["target", "ref1", "ref2", "ref3", "nonref1", "nonref2", "nonref3"]
    fig, ax = plt.subplots(figsize=(10, 10))
    vmin, vmax = np.percentile(image, [5, 99])
    ax.imshow(image, cmap='gray', vmin=vmin, vmax=vmax, origin='lower')

    sat_overlay = np.where(image >= SAT_THRESHOLD, 1.0, np.nan)
    ax.imshow(sat_overlay, cmap='Reds', vmin=0, vmax=1, alpha=0.5, origin='lower')

    selections = {}
    for role in roles:
        ax.set_title(f"Click {role}")
        fig.canvas.draw()
        x, y = plt.ginput(1, timeout=0)[0]
        selections[role] = (x, y)
        ax.plot(x, y, 'o', mfc='none', mec='cyan', ms=15)
        fig.canvas.draw()

    plt.close(fig)
    return selections

def query_catalog(ra, dec, filter_band, radius_arcsec=5.0):
    coord = ICRS(ra * u.degree, dec * u.degree)
    for col_name, cat_list in target_cats[filter_band].items():
        for cat in cat_list:
            print(cat)
            try:
                result = Vizier.query_region(coord, radius=radius_arcsec * u.arcsec, catalog=cat)
                if result is not None:
                    time.sleep(0.1)
            except Exception as err:
                print(f"Query failed for {cat}: {err}")
                continue
            if result and len(result) > 0:
                value = float(result[0][col_name][0])
                if not np.isnan(value):
                    print(f"{filter_band}: {cat}")
                    return value, cat
    return None, None

def measure_fwhm(image, x, y, cutout_size=21):
    half = cutout_size // 2
    x_int, y_int = int(round(x)), int(round(y))
    cutout = image[y_int - half:y_int + half + 1, x_int - half:x_int + half + 1]
    cutout_sub = np.maximum(cutout - np.median(cutout), 0)

    yc, xc = np.indices(cutout_sub.shape)
    total = cutout_sub.sum()
    mom1_x = (xc * cutout_sub).sum() / total
    mom1_y = (yc * cutout_sub).sum() / total
    mom2_x = ((xc - mom1_x)**2 * cutout_sub).sum() / total
    mom2_y = ((yc - mom1_y)**2 * cutout_sub).sum() / total

```

```

sigma_x, sigma_y = mom2_x**0.5, mom2_y**0.5
sigma_to_fwhm = np.sqrt(8 * np.log(2))
fwhm_x, fwhm_y = sigma_x * sigma_to_fwhm, sigma_y * sigma_to_fwhm
fwhm = (fwhm_x * fwhm_y)**0.5

x_centroid = (x_int - half) + mom1_x
y_centroid = (y_int - half) + mom1_y

return fwhm, x_centroid, y_centroid

def measure_aperture(image, noise, x, y, fwhm):
    pos = (x, y)
    src_ap = CircularAperture(pos, r=2 * fwhm)
    sky_an = CircularAnnulus(pos, r_in=3 * fwhm, r_out=5 * fwhm)

    sky_stats = ApertureStats(image, sky_an, sigma_clip=SigmaClip(sigma=3.0))
    sky_per_pix = sky_stats.median
    sky_std = sky_stats.std

    phot = aperture_photometry(image, src_ap, error=noise)
    raw_sum = phot['aperture_sum'][0]
    raw_err = phot['aperture_sum_err'][0]

    ap_area = src_ap.area
    net_flux = raw_sum - sky_per_pix * ap_area
    net_err = np.sqrt(raw_err**2 + ap_area * sky_std**2)

    return net_flux, net_err, sky_per_pix, sky_std

def compute_zp(refs, exptime):
    zps = [cat_mag + 2.5 * np.log10(flux / exptime) for flux, cat_mag in refs]
    zp_mean = np.mean(zps)
    zp_std = np.std(zps, ddof=1)
    return zp_mean, zp_std

def calibrate(flux, flux_err, exptime, zp_mean, zp_std):
    instr_mag = -2.5 * np.log10(flux / exptime)
    instr_err = (2.5 / np.log(10)) * (flux_err / flux)
    calib_mag = instr_mag + zp_mean
    calib_err = np.sqrt(instr_err**2 + zp_std**2)
    return calib_mag, calib_err, instr_mag

def photometry(stack_path, noise_path, exptime, filter_band):
    image, _, wcs = load_stack(stack_path)
    noise = fits.getdata(noise_path)

    print(f"\n=== {stack_path} ===")
    print(f"Filter: {filter_band}, EXPTIME: {exptime}s")

    selections = display_and_select(image)
    measurements = {}

    for role, (x, y) in selections.items():
        ra, dec = wcs.pixel_to_world_values(x, y)
        if role == 'target':
            cat_mag, cat_id = None, None
        else:
            cat_mag, cat_id = query_catalog(float(ra), float(dec), filter_band)
            while cat_mag is None:
                print(f"No catalog match for {role}. Re-click.")
                new_sel = display_and_select(image, roles=[role])
                x, y = new_sel[role]
                ra, dec = wcs.pixel_to_world_values(x, y)
                cat_mag, cat_id = query_catalog(float(ra), float(dec), filter_band)
            measurements[role] = {'x': x, 'y': y, 'ra': float(ra), 'dec': float(dec),
                                  'cat_mag': cat_mag, 'cat_id': cat_id}

    for role, star in measurements.items():

```

```

    fwhm, x_centroid, y_centroid = measure_fwhm(image, star['x'], star['y'])
    star.update({'fwhm': fwhm, 'x_centroid': x_centroid, 'y_centroid': y_centroid})

fwhms = [star['fwhm'] for star in measurements.values()]
fwhm_mean = float(np.mean(fwhms))
fwhm_std = float(np.std(fwhms, ddof=1))

for role, star in measurements.items():
    flux, flux_err, sky, sky_std = measure_aperture(image, noise, star['x_centroid'], star['y_centroid'], fwhm_mean)
    x_int, y_int = int(round(star['x_centroid'])), int(round(star['y_centroid']))
    peak = float(image[max(y_int - 2, 0):y_int + 3, max(x_int - 2, 0):x_int + 3].max())
    star.update({'flux': flux, 'flux_err': flux_err, 'sky': sky,
                'sky_std': sky_std, 'peak': peak, 'ap_radius': 2 * fwhm_mean})

refs = [(measurements[role]['flux'], measurements[role]['cat_mag']) for role in ('ref1', 'ref2', 'ref3')]
zp_mean, zp_std = compute_zp(refs, exptime)

for role, star in measurements.items():
    calibrated_mag, calibrated_err, instrumental_mag = calibrate(star['flux'], star['flux_err'], exptime, zp_mean,
                                                                zp_std)
    star.update({'instr_mag': instrumental_mag, 'calib_mag': calibrated_mag, 'calib_err': calibrated_err})

print(f"Mean FWHM: {fwhm_mean:.2f} {fwhm_std:.2f} pix")
print(f"ZP: {zp_mean:.3f} {zp_std:.3f}")
print(f"Target calib mag: {measurements['target']['calib_mag']:.3f} {measurements['target']['calib_err']:.3f}")
print("Sanity check (nonrefs):")
for role in ('nonref1', 'nonref2', 'nonref3'):
    star = measurements[role]
    print(f" {role}: derived={star['calib_mag']:.3f} cat={star['cat_mag']:.3f}
          ={star['calib_mag']-star['cat_mag']:+.3f}")

cols = ['role', 'x', 'y', 'ra', 'dec', 'fwhm', 'peak', 'ap_radius',
        'flux', 'flux_err', 'sky', 'sky_std',
        'cat_mag', 'cat_id', 'instr_mag', 'calib_mag', 'calib_err']
csv_path = stack_path.replace('.fits', '_photometry.csv')
with open(csv_path, 'w', newline='') as csv_file:
    csv_file.write(f"# stack={stack_path}\n")
    csv_file.write(f"# filter={filter_band}\n")
    csv_file.write(f"# exptime={exptime}\n")
    csv_file.write(f"# fwhm_mean={fwhm_mean:.4f}\n")
    csv_file.write(f"# fwhm_std={fwhm_std:.4f}\n")
    csv_file.write(f"# zp_mean={zp_mean:.4f}\n")
    csv_file.write(f"# zp_std={zp_std:.4f}\n")
    writer = csv.writer(csv_file)
    writer.writerow(cols)
    for role, star in measurements.items():
        writer.writerow([role] + [star.get(col) for col in cols[1:]])
print(f"Wrote {csv_path}")

return measurements, zp_mean, zp_std

stacks = [
    ('J0813_2026-04-14/J08_14_B_solved.fits',
     'J0813_2026-04-14/J0813_2026-04-14_B_300.0_noise.fits',
     300.0, 'B'),
    ('J0813_2026-04-14/J08_14_R_solved.fits',
     'J0813_2026-04-14/J0813_2026-04-14_R_300.0_noise.fits',
     300.0, 'R'),
    ('J0813_2026-04-14/J08_14_I_solved.fits',
     'J0813_2026-04-14/J0813_2026-04-14_I_300.0_noise.fits',
     300.0, 'I'),
    ('J0813_2026-04-23/J08_23_R_solved.fits',
     'J0813_2026-04-23/J0813_2026-04-23_R_180.0_noise.fits',
     180.0, 'R'),
    ('J0813_2026-04-23/J08_23_I_solved.fits',
     'J0813_2026-04-23/J0813_2026-04-23_I_300.0_noise.fits',
     300.0, 'I'),

```

```
]
for stack_path, noise_path, exptime, filter_band in stacks:
    photometry(stack_path, noise_path, exptime, filter_band)
```

C. SED FITTING PIPELINE

```
import numpy as np
import matplotlib.pyplot as plt

magnitudes = {
    '4_14': {
        'B': {'Mag': 16.5712310415556, 'Err': 0.119056867092615},
        'r': {'Mag': 15.7421356414714, 'Err': 0.0223830455408943},
        'i': {'Mag': 15.441636338418, 'Err': 0.0434749167576849}
    },
    '4_23': {
        'r': {'Mag': 15.6703043789582, 'Err': 0.102007662572268},
        'i': {'Mag': 15.1649002694476, 'Err': 0.134509502668298}
    }
}

nu_eff = {
    'B': 4361,
    'r': 6231,
    'i': 7625
}

#First, I have to convert from apparent magnitudes to Fnu
#B filter is JC, meaning that the Magnitude-system flux zero-point is different

M_B_AB = magnitudes['4_14']['B']['Mag'] - 0.125
F_nu_B_n1 = 3631 * 10 ** (-M_B_AB/2.5)
F_nu_r_n1 = 3631 * 10 ** (-magnitudes['4_14']['r']['Mag']/2.5)
F_nu_i_n1 = 3631 * 10 ** (-magnitudes['4_14']['i']['Mag']/2.5)
F_nu_r_n2 = 3631 * 10 ** (-magnitudes['4_23']['r']['Mag']/2.5)
F_nu_i_n2 = 3631 * 10 ** (-magnitudes['4_23']['i']['Mag']/2.5)

nu_B = 2.998e18 / nu_eff['B']
nu_r = 2.998e18 / nu_eff['r']
nu_i = 2.998e18 / nu_eff['i']

sigma_mag_n1 = np.array([magnitudes['4_14'][x]['Err'] for x in ['B', 'r', 'i']])
sigma_mag_n2 = np.array([magnitudes['4_23'][x]['Err'] for x in ['r', 'i']])

sigma_log_F_n1 = sigma_mag_n1 / 2.5
sigma_log_F_n2 = sigma_mag_n2 / 2.5

log_nu_n1 = np.log10([nu_B, nu_r, nu_i])
log_f_n1 = np.log10([F_nu_B_n1, F_nu_r_n1, F_nu_i_n1])
log_nu_n2 = np.log10([nu_r, nu_i])
log_f_n2 = np.log10([F_nu_r_n2, F_nu_i_n2])

weights_n1 = 1 / sigma_log_F_n1
weights_n2 = 1 / sigma_log_F_n2

(slope_n1, intercept_n1), cov_n1 = np.polyfit(log_nu_n1, log_f_n1, 1, w=weights_n1, cov='unscaled')
(slope_n2, intercept_n2), cov_n2 = np.polyfit(log_nu_n2, log_f_n2, 1, w=weights_n2, cov='unscaled')
print(f"For night 1:\nslope = {slope_n1}\nintercept = {intercept_n1}\ncovariance = {cov_n1}")
print(f"For night 2:\nslope = {slope_n2}\nintercept = {intercept_n2}\ncovariance = {cov_n2}")

def n1_fit(x):
    y = slope_n1 * x + intercept_n1
    return y

def n2_fit(x):
    y = slope_n2 * x + intercept_n2
```

```

    return y

x = [14.5, 14.9]
y1 = [n1_fit(x_val) for x_val in x]
y2 = [n2_fit(x_val) for x_val in x]

plt.plot(x, y1, ls='--', color='orange')
plt.plot(x, y2, ls='--', color='blue')
plt.errorbar(log_nu_n1, log_f_n1, sigma_log_F_n1, marker='o', label='4/14 Magnitudes', ls='', color='orange')
plt.errorbar(log_nu_n2, log_f_n2, sigma_log_F_n2, marker='o', label='4/23 Magnitudes', ls='', color='blue')
plt.xlabel("log(nu / Hz)")
plt.ylabel("log(F_nu / Jy)")
plt.legend()
plt.show()

#Second plot: same data and best-fit lines in linear (non-log) space
F_nu_n1 = np.array([F_nu_B_n1, F_nu_r_n1, F_nu_i_n1])
F_nu_n2 = np.array([F_nu_r_n2, F_nu_i_n2])
nu_n1 = np.array([nu_B, nu_r, nu_i])
nu_n2 = np.array([nu_r, nu_i])
sigma_F_n1 = F_nu_n1 * np.log(10) * sigma_log_F_n1
sigma_F_n2 = F_nu_n2 * np.log(10) * sigma_log_F_n2

A_n1 = 10**intercept_n1
A_n2 = 10**intercept_n2
nu_smooth = np.logspace(14.5, 14.9, 100)
F_smooth_n1 = A_n1 * nu_smooth**slope_n1
F_smooth_n2 = A_n2 * nu_smooth**slope_n2

z = 1.512
c = 2.998e18
lam_2500_obs = (1 + z) * 2500
nu_2500_obs = c / lam_2500_obs
log_nu_2500_obs = np.log10(nu_2500_obs)

log_F_2500_n1 = n1_fit(log_nu_2500_obs)
log_F_2500_n2 = n2_fit(log_nu_2500_obs)
sigma_log_F_2500_n1 = np.sqrt(log_nu_2500_obs**2 * cov_n1[0, 0] + cov_n1[1, 1] + 2 * log_nu_2500_obs * cov_n1[0, 1])
sigma_log_F_2500_n2 = np.sqrt(log_nu_2500_obs**2 * cov_n2[0, 0] + cov_n2[1, 1] + 2 * log_nu_2500_obs * cov_n2[0, 1])
F_2500_n1 = 10**log_F_2500_n1
F_2500_n2 = 10**log_F_2500_n2
sigma_F_2500_n1 = F_2500_n1 * np.log(10) * sigma_log_F_2500_n1
sigma_F_2500_n2 = F_2500_n2 * np.log(10) * sigma_log_F_2500_n2

plt.figure()
plt.errorbar(nu_2500_obs, F_2500_n1, sigma_F_2500_n1, color='orange', ls='', marker='*', ms=20, capsize=5, label='4/14  
2500 Flux')
plt.errorbar(nu_2500_obs, F_2500_n2, sigma_F_2500_n2, color='blue', ls='', marker='*', ms=20, capsize=5, label='4/23  
2500 Flux')
plt.plot(nu_smooth, F_smooth_n1, ls='--', color='orange')
plt.plot(nu_smooth, F_smooth_n2, ls='--', color='blue')
plt.errorbar(nu_n1, F_nu_n1, sigma_F_n1, marker='o', label='4/14 Magnitudes', ls='', color='orange', capsize=3)
plt.errorbar(nu_n2, F_nu_n2, sigma_F_n2, marker='o', label='4/23 Magnitudes', ls='', color='blue', capsize=3)
plt.xlabel("nu (Hz)")
plt.ylabel("F_nu (Jy)")
plt.legend()
plt.show()

#Third plot: apparent magnitude vs time for all filters and interpolated 2500AA value
from astropy.time import Time

m_2500_n1 = -2.5 * np.log10(F_2500_n1 / 3631)
m_2500_n2 = -2.5 * np.log10(F_2500_n2 / 3631)
sigma_m_2500_n1 = (2.5 / np.log(10)) * (sigma_F_2500_n1 / F_2500_n1)
sigma_m_2500_n2 = (2.5 / np.log(10)) * (sigma_F_2500_n2 / F_2500_n2)

jd_n1 = Time('2026-04-15T03:00:00', format='isot', scale='utc').jd
jd_n2 = Time('2026-04-24T02:45:00', format='isot', scale='utc').jd

```

```

jd_offset = 2460000

fig, ax = plt.subplots()
ax.errorbar(jd_n1 - jd_offset, magnitudes['4_14']['B']['Mag'], magnitudes['4_14']['B']['Err'],
            marker='o', ls='', color='steelblue', capsize=3, label='$B$')
ax.errorbar(jd_n1 - jd_offset, magnitudes['4_14']['r']['Mag'], magnitudes['4_14']['r']['Err'],
            marker='o', ls='', color='tomato', capsize=3, label='$r$')
ax.errorbar(jd_n2 - jd_offset, magnitudes['4_23']['r']['Mag'], magnitudes['4_23']['r']['Err'],
            marker='o', ls='', color='tomato', capsize=3)
ax.errorbar(jd_n1 - jd_offset, magnitudes['4_14']['i']['Mag'], magnitudes['4_14']['i']['Err'],
            marker='o', ls='', color='firebrick', capsize=3, label='$i$')
ax.errorbar(jd_n2 - jd_offset, magnitudes['4_23']['i']['Mag'], magnitudes['4_23']['i']['Err'],
            marker='o', ls='', color='firebrick', capsize=3)
ax.errorbar(jd_n1 - jd_offset, m_2500_n1, sigma_m_2500_n1,
            marker='*', ls='', ms=15, color='purple', capsize=5, label=r'$m_{2500}$ (interp.)')
ax.errorbar(jd_n2 - jd_offset, m_2500_n2, sigma_m_2500_n2,
            marker='*', ls='', ms=15, color='purple', capsize=5)
ax.invert_yaxis()
ax.set_xlabel(f"JD $-$ {jd_offset}")
ax.set_ylabel("Apparent Magnitude (AB)")
ax.legend()
plt.show()

```

REFERENCES

- Risaliti, G. & Lusso, E. 2015, *ApJ*, 815, 33.
doi:10.1088/0004-637X/815/1/33
- Risaliti, G. & Lusso, E. 2019, *Nature Astronomy*, 3, 272.
doi:10.1038/s41550-018-0657-z
- Suberlak, K. L., Ivezić, Ž., & MacLeod, C. 2021, *ApJ*, 907, 96. doi:10.3847/1538-4357/ac1346
- Signorini, M., Risaliti, G., Lusso, E., et al. 2024, *A&A*, 687, A32. doi:10.1051/0004-6361/202348941
- Lyke, B. W., Higley, A. N., McLane, J. N., et al. 2020, *ApJS*, 250, 8. doi:10.3847/1538-4365/aba623
- Shen, Y., Richards, G. T., Strauss, M. A., et al. 2011, *ApJS*, 194, 45. doi:10.1088/0067-0049/194/2/45