Lab_3

November 6, 2025

1 Lab 3

Lab 3 centers on the photometric analysis of a short-period Cepheid variable star using a sequence of exposures in different filters, as well as associated calibration data.

As before, this notebook will first illustrate some Python features.

[1]:
```
import numpy as np
%matplotlib inline
import matplotlib as mpl
import pylab
import sys; sys.path.append('/home/dyson/fall25/180ASTR/Lab3/')
from scipy.interpolate import interp1d
import os
import matplotlib.pyplot as plt
import math
import warnings

from astropy.io import fits
from astropy.time import Time
from scipy.interpolate import UnivariateSpline, PchipInterpolator, interp1d

from a180 import ap_phot
from phot_calcs import find_star_center, detect_trails

# phot_calc.py is located in the vega server at /data/2025-Fall/Lab3/Group4/
```

1.1 Photometric Analysis of a Cepheid Variable Star

Let's get organized first, and specify our file locations.

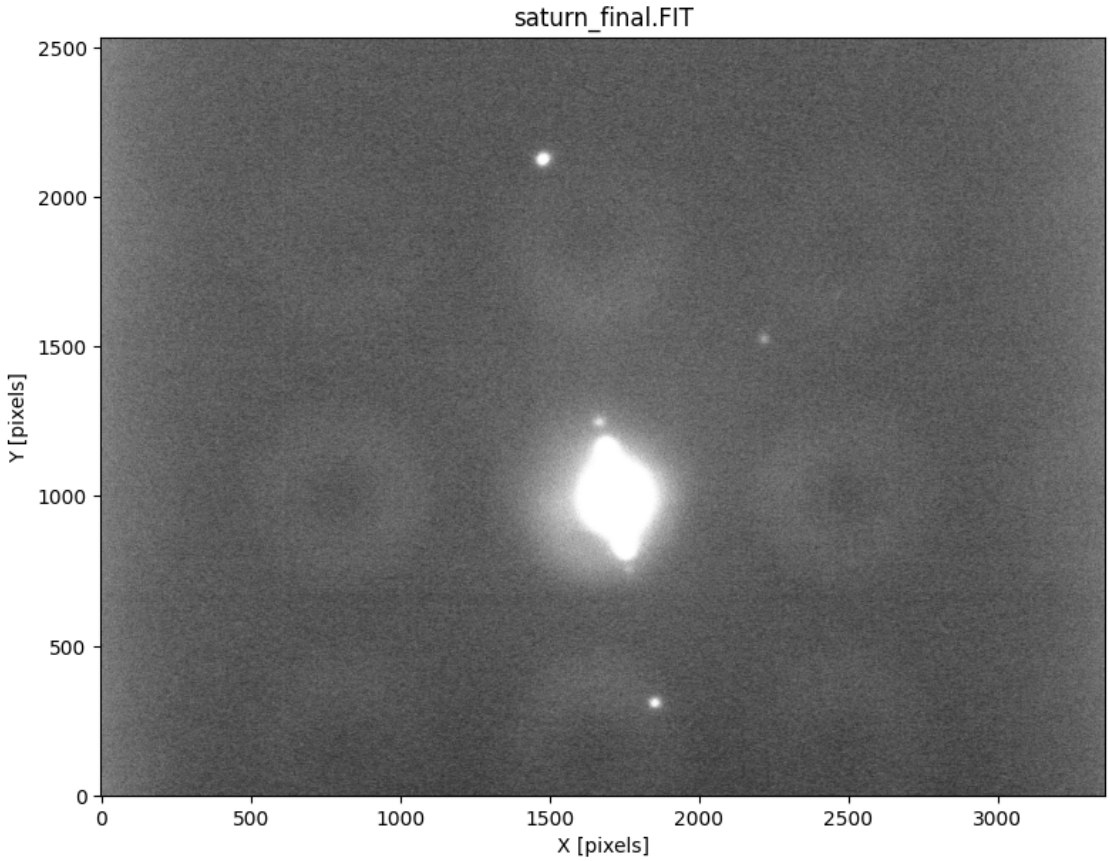
We can't save our processed data in the raw data directory; we must specify a different location. If it doesn't exist, we need to make it first. You can do this from within Jupyter. Suppose you want to make a directory called `foo`. You would open a Terminal and type `mkdir foo`.

[2]:
```
raw_data_dir = '/home/dyson/fall25/180ASTR/Lab3/raw/' # directory where raw␣
↪data are stored
proc_data_dir = '/home/dyson/fall25/180ASTR/Lab3/processed_data/' # place to␣
↪store our processed data
```

```
# dark frames corresponding to the flats
raw_dark_files = []

# flat fields in V and R band
raw_flatV_files = []
raw_flatR_files = []

# sky images in V and R band
raw_skyV_files = []
raw_skyR_files = []

# skies for the photometric standard stars
raw_photskyV_files = []
raw_photskyR_files = []

# observations of the photometric standard star (Landolt standard)
raw_photV_files = []
raw_photR_files = []

# science target observations in V and R band
raw_targV_files = []
raw_targR_files = []

n = 30
for i in range(1, n + 1):
    if i < 3:
        # flats (1,2)
        raw_flatV_files.append(f'FLAT_V{i}.FIT')
        raw_flatR_files.append(f'FLAT_R{i}.FIT')

    if i < 4:
        # darks (1-3)
        raw_dark_files.append(f'DARK_{i}.FIT')

        # sky (1-3)
        raw_skyV_files.append(f'SKYCY_V{i}.FIT')
        raw_skyR_files.append(f'SKYCY_R{i}.FIT')
        raw_photskyV_files.append(f'176SKY_V{i}.FIT')
        raw_photskyR_files.append(f'176SKY_R{i}.FIT')

    if i < 5:
        # calibration images (1-4)
        raw_photV_files.append(f'176_V{i}.FIT')
        raw_photR_files.append(f'176_R{i}.FIT')

    # science images (1-30)
```

```
    raw_targV_files.append(f'CYA_V{i}.FIT')
    raw_targR_files.append(f'CYA_R{i}.FIT')
```

[3]:
```
# Load the Saturn image
saturn_file = os.path.join(raw_data_dir, 'saturn_final.FIT')
saturn_data = fits.getdata(saturn_file)

# Create figure
fig, ax = plt.subplots(figsize=(8, 8))

# Display the image with appropriate scaling
ax.imshow(saturn_data, cmap='gray', origin='lower',
          vmin=np.percentile(saturn_data, 1),
          vmax=np.percentile(saturn_data, 99))
ax.set_title('saturn_final.FIT')
ax.set_xlabel('X [pixels]')
ax.set_ylabel('Y [pixels]')

plt.tight_layout()
plt.show()

# Print some basic info
print(f"Image shape: {saturn_data.shape}")
print(f"Data type: {saturn_data.dtype}")
print(f"Min value: {np.min(saturn_data)}")
print(f"Max value: {np.max(saturn_data)}")
print(f"Mean value: {np.mean(saturn_data):.2f}")
```




```
Image shape: (2536, 3358)
Data type: uint16
Min value: 0
Max value: 65535
Mean value: 1101.56
```

1.1.1 Calibrating the data

We'll need to calibrate our data. For our target star images, we'll need to subtract out the background light using our sky exposures, as well as divide by the response function (the "flat field").

Creating the flat fields in each band For each band (V and R), we must first construct the flat field; this is done by removing the expected bias and dark current levels.

We're going to be loading and combining files fiarly often in our calibrations, so lets define a load and combine function. We'll use median combination rather than averaging since it is more robust to errors that come from cosmic rays.

Let's first apply it to the dark exposure.

[4]:
```
def load_and_combine(filenames, prefix=''):
    "Load and use median combination on a list of exposures.  Returns a numpy␣
 ↪array."
    images = [] # define an empty list
    for fn in filenames:
        images.append(fits.getdata(prefix+fn)) # populate the list with image␣
 ↪arrays from each file
    images = np.array(images) # turn the list into a 3d numpy array
    combined_im = np.median(images, axis=0) # use median combination along the␣
 ↪first axis (image index)
    return combined_im
```

[5]:
```
# process dark
dark_im = load_and_combine(raw_dark_files, prefix=raw_data_dir) # load and␣
 ↪combine dark exposures into a dark frame
dark_fn = 'dark.fits' # filename for our combined dark frame
fits.writeto(proc_data_dir+dark_fn, dark_im, overwrite=True) # store the␣
 ↪combined dark frame in a FITS file
```

Now that we have a dark exposure, we can create the flat fields.

In a given band, we will load and combine the flat field exposures. We will subtract out the dark frame, and then normalize by the median value so that the flat field frame represents a relative response level. We can then save it to file.

[6]:
```
# Create V-band flat field
flatV_im = load_and_combine(raw_flatV_files, prefix=raw_data_dir) - dark_im  #␣
 ↪subtract dark frame
flatV_im /= np.median(flatV_im)  # normalize by median value
flatV_fn = 'flatV.fits'
fits.writeto(proc_data_dir + flatV_fn, flatV_im, overwrite=True)  # save␣
 ↪combined V-band flat field

# Create R-band flat field
flatR_im = load_and_combine(raw_flatR_files, prefix=raw_data_dir) - dark_im  #␣
 ↪subtract dark frame
flatR_im /= np.median(flatR_im)  # normalize by median value
flatR_fn = 'flatR.fits'
fits.writeto(proc_data_dir + flatR_fn, flatR_im, overwrite=True)  # save␣
 ↪combined R-band flat field
```

Processing sky exposures We won't have to do anything fancy to the sky background exposures, just load them and combine them for each set.

[7]:
```
# process and store sky exposures for target, V band
skyV_im = load_and_combine(raw_skyV_files, prefix=raw_data_dir)  # combine␣
 ↪target sky V-band frames
```

```
skyV_fn = 'sky_target_V.fits'
fits.writeto(proc_data_dir + skyV_fn, skyV_im, overwrite=True)

# process and store sky exposures for target, R band
skyR_im = load_and_combine(raw_skyR_files, prefix=raw_data_dir)  # combine␣
 ↪target sky R-band frames
skyR_fn = 'sky_target_R.fits'
fits.writeto(proc_data_dir + skyR_fn, skyR_im, overwrite=True)

# process and store sky exposures for photometric standard, V band
phot_skyV_im = load_and_combine(raw_photskyV_files, prefix=raw_data_dir)  #␣
 ↪combine photometric sky V-band frames
phot_skyV_fn = 'sky_photV.fits'
fits.writeto(proc_data_dir + phot_skyV_fn, phot_skyV_im, overwrite=True)

# process and store sky exposures for photometric standard, R band
phot_skyR_im = load_and_combine(raw_photskyR_files, prefix=raw_data_dir)  #␣
 ↪combine photometric sky R-band frames
phot_skyR_fn = 'sky_photR.fits'
fits.writeto(proc_data_dir + phot_skyR_fn, phot_skyR_im, overwrite=True)
```

Calibrating photometric exposures We have several data sets we need to calibrate, namely our target and photometric standard star exposures in each filter band.

To do this, we'll load each exposure, subtract the sky background, and divide by the flat field response. We'll store the output as a file. Let's try to simplify the process by writing a function.

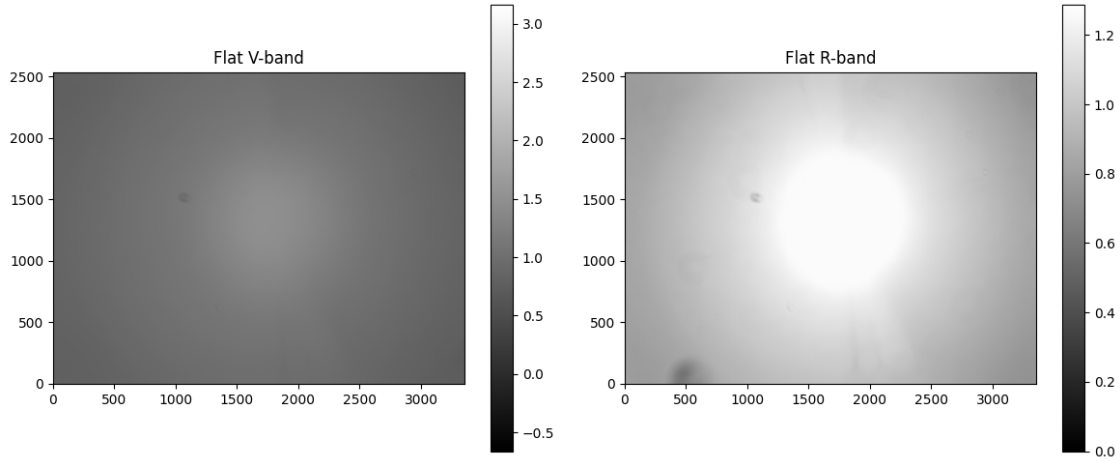
[8]:
```
def process_photometry(raw_filename, sky_im, flat_im, raw_dir='', out_dir='',␣
 ↪prefix='proc-'):
    "Calibrate a photometric exposure"
    out_fn = prefix + raw_filename
    im = fits.getdata(raw_dir + raw_filename)

    # Calibration steps: subtract sky, divide by flat
    # Suppress warnings about division by zero/invalid values
    with warnings.catch_warnings():
        warnings.simplefilter("ignore", RuntimeWarning)
        proc_im = (im - sky_im) / flat_im

    # Replace any NaN or inf values with zero (these are bad pixels anyway)
    proc_im = np.nan_to_num(proc_im, nan=0.0, posinf=0.0, neginf=0.0)

    fits.writeto(out_dir + out_fn, proc_im, overwrite=True)
    return out_fn
```

[9]:
```
# Check the flat field
print("Flat V-band statistics:")
```

```
print(f"  Shape: {flatV_im.shape}")
print(f"  Min: {np.min(flatV_im)}")
print(f"  Max: {np.max(flatV_im)}")
print(f"  Mean: {np.mean(flatV_im)}")
print(f"  Number of zeros: {np.sum(flatV_im == 0)}")
print(f"  Number of NaNs: {np.sum(np.isnan(flatV_im))}")
print(f"  Number < 0.1: {np.sum(flatV_im < 0.1)}")

print("\nFlat R-band statistics:")
print(f"  Shape: {flatR_im.shape}")
print(f"  Min: {np.min(flatR_im)}")
print(f"  Max: {np.max(flatR_im)}")
print(f"  Mean: {np.mean(flatR_im)}")
print(f"  Number of zeros: {np.sum(flatR_im == 0)}")
print(f"  Number of NaNs: {np.sum(np.isnan(flatR_im))}")
print(f"  Number < 0.1: {np.sum(flatR_im < 0.1)}")

# Visualize the flat field to see where the problem areas are
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

im0 = axes[0].imshow(flatV_im, cmap='gray', origin='lower')
axes[0].set_title('Flat V-band')
plt.colorbar(im0, ax=axes[0])

im1 = axes[1].imshow(flatR_im, cmap='gray', origin='lower')
axes[1].set_title('Flat R-band')
plt.colorbar(im1, ax=axes[1])

plt.tight_layout()
plt.show()
```

```
Flat V-band statistics:
  Shape: (2536, 3358)
  Min: -0.6601873300433074
  Max: 3.163460570047336
  Mean: 1.0247967824751179
  Number of zeros: 17
  Number of NaNs: 0
  Number < 0.1: 20

Flat R-band statistics:
  Shape: (2536, 3358)
  Min: 0.0
  Max: 1.2875751503006012
  Mean: 1.0108757053411948
  Number of zeros: 17
  Number of NaNs: 0
```

```
Number < 0.1: 23
```



[10]:
```
# process photometric standard exposures, V band
proc_photV_files = []
for fn in raw_photV_files:
    proc_fn = process_photometry(fn, phot_skyV_im, flatV_im,
                                 raw_data_dir,
                                 proc_data_dir)
    proc_photV_files.append(proc_fn)

# process photometric standard exposures, R band
proc_photR_files = []
for fn in raw_photR_files:
    proc_fn = process_photometry(fn, phot_skyR_im, flatR_im,
                                 raw_dir=raw_data_dir,
                                 out_dir=proc_data_dir)
    proc_photR_files.append(proc_fn)

# process target star exposures, V band
proc_targV_files = []
for fn in raw_targV_files:
    proc_fn = process_photometry(fn, skyV_im, flatV_im,
                                 raw_dir=raw_data_dir,
                                 out_dir=proc_data_dir)
    proc_targV_files.append(proc_fn)

# process target star exposures, R band
proc_targR_files = []
for fn in raw_targR_files:
    proc_fn = process_photometry(fn, skyR_im, flatR_im,
                                 raw_dir=raw_data_dir,
```

```
                                    out_dir=proc_data_dir)
    proc_targR_files.append(proc_fn)
```

Now all our data have had the first level of calibration performed, and we're ready to start making photometric measurements.

1.1.2 Getting uncalibrated photometry

We need to add up the light from our measurements. This will give us uncalibrated photometry, in units of DN/s.

To do this we'll define an "aperture," which is a virtual region in the image over which we'll add up the counts. This will be a circular aperture centered on the star.

We also expect that our sky background subtraction is not perfect, so we'll define a "sky annulus," a ring-like region outside of our photometric aperture over which the residual sky level will be determiend and subtracted from our target aperture.

So we'll need to find: * star center in pixel coordinates * radius of the photometric aperture * inner and outer radii of the sky annulus

To simplify the analysis, we'll use the same aperture parameters on all exposures. That way we only have to find the stellar centers in each exposure.

Let's start with an example exposure.

Example exposure photometry Let's load and display an example exposure. We'll overlay the photometric and sky annulus apertures on the image to check consistency, and we'll get the photometric measurement.

We should be sure the apertures are centered on the star.

The ideal size for the photometric aperture is roughly to have it as large as possible without being so big as being dominated by sky background noise. So we should have it sized so that it encompasses the majority of the visible starlight.

Note that since we are using the same aperture sizes for all exposures, it won't be optimal in all cases. We're aiming for "good enough" here.

The sky annulus inner radius should be large enough so that no signal from the star is in the sky annulus. The outer radius should be large enough so that a good number of pixels are included (ideally more area in the sky annulus than the target aperture), but not so large that systematic errors from sky nonuniformity creep in. There is no grea recipe for the sky annulus; just get something good enough.

Once we have our aperture set, let's do the photometry.

[11]:
```
flatV1 = fits.getdata(os.path.join(raw_data_dir, 'FLAT_V1.FIT')).astype(float)
flatV2 = fits.getdata(os.path.join(raw_data_dir, 'FLAT_V2.FIT')).astype(float)
flatR1 = fits.getdata(os.path.join(raw_data_dir, 'FLAT_R1.FIT')).astype(float)
flatR2 = fits.getdata(os.path.join(raw_data_dir, 'FLAT_R2.FIT')).astype(float)

dark1 = fits.getdata(os.path.join(raw_data_dir, 'DARK_1.FIT')).astype(float)
```

```
dark2 = fits.getdata(os.path.join(raw_data_dir, 'DARK_2.FIT')).astype(float)

gainV = ((np.mean(flatV1) + np.mean(flatV2)) - (np.mean(dark1) + np.
 ↪mean(dark2))) / \
        (np.var(flatV1 - flatV2) - np.var(dark1 - dark2))

gainR = ((np.mean(flatR1) + np.mean(flatR2)) - (np.mean(dark1) + np.
 ↪mean(dark2))) / \
        (np.var(flatR1 - flatR2) - np.var(dark1 - dark2))
print("GainV =", gainV, "e-/DN")
print("GainR =", gainR, "e-/DN")
```

```
GainV = 0.3806225801199869 e-/DN
GainR = 0.34881623902860226 e-/DN
```

Calculating the gain instead of getting it from header barely changed the values

It doesn't seem to change the measured mag (at least to 7 decimals)

It only changes the error in R mag by ~0.0000243

Positions for all star exposures We need to get the star position for all of our exposures. The simplest way is through visual inspection.

This section uses find_star_center

[12]:
```
# Find positions for all target V-band exposures (already sky-subtracted)
CY_radV = 0
targV_xys = []
for i, fn in enumerate(proc_targV_files):
    im = fits.getdata(proc_data_dir + fn)
    x, y, temp = find_star_center(im)
    targV_xys.append((x, y))
    CY_radV += temp

CY_radV = math.ceil(CY_radV / np.size(proc_targV_files))

# Find positions for all target R-band exposures (already sky-subtracted)
CY_radR = 0
targR_xys = []
for i, fn in enumerate(proc_targR_files):
    im = fits.getdata(proc_data_dir + fn)
    x, y, temp = find_star_center(im)
    targR_xys.append((x, y))
    CY_radR += temp

CY_radR = math.ceil(CY_radR / np.size(proc_targR_files))

# Find positions for all standard V-band exposures (already sky-subtracted)
```

```
std_radV = 0
stdV_xys = []
for i, fn in enumerate(proc_photV_files):
    im = fits.getdata(proc_data_dir + fn)
    x, y, temp = find_star_center(im)
    stdV_xys.append((x, y))
    std_radV += temp

std_radV = math.ceil(std_radV / np.size(proc_photV_files))

# Find positions for all standard R-band exposures (already sky-subtracted)
std_radR = 0
stdR_xys = []
for i, fn in enumerate(proc_photR_files):
    im = fits.getdata(proc_data_dir + fn)
    x, y, temp = find_star_center(im)
    stdR_xys.append((x, y))
    std_radR += temp

std_radR = math.ceil(std_radR / np.size(proc_photR_files))

print(std_radV, ' ', std_radR, ' ', CY_radV, ' ', CY_radR)
```

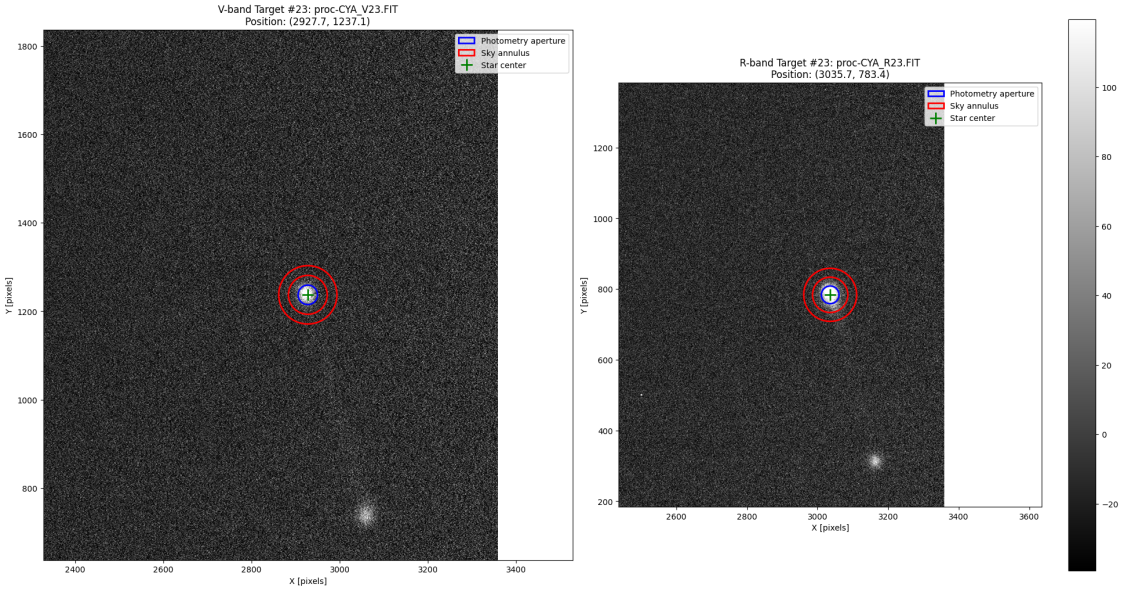
```
23   36   22   25
```

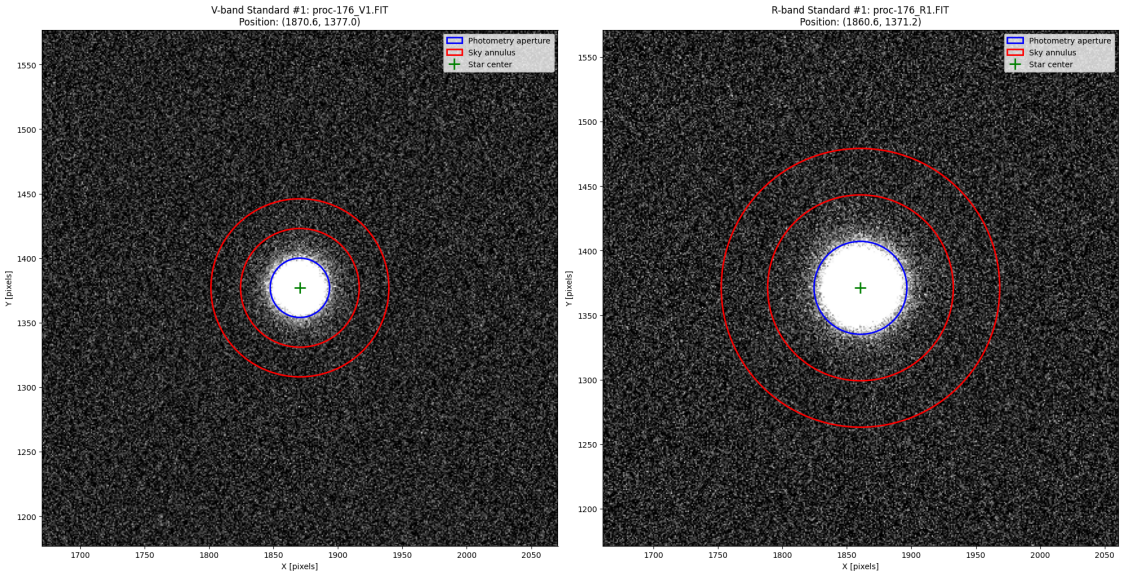
[13]:
```
# Display target star images with apertures (V and R side-by-side)
for i in range(22, 23): # this range shows the images where the target shifted␣
 ↪during the exposure
#for i in range(min(3, len(proc_targV_files))):
    # Load both V and R images
    im_V = fits.getdata(proc_data_dir + proc_targV_files[i])
    im_R = fits.getdata(proc_data_dir + proc_targR_files[i])

    x_V, y_V = targV_xys[i]
    x_R, y_R = targR_xys[i]

    # Create figure with two subplots
    fig, (ax1, ax2) = pylab.subplots(1, 2, figsize=(20, 10))

    # V-band image
    vmin_V, vmax_V = np.percentile(im_V[np.isfinite(im_V)], [30, 99.9])
    ax1.imshow(im_V, cmap='gray', origin='lower', vmin=vmin_V, vmax=vmax_V)

    # V-band apertures
    sky_ann_inner_V = CY_radV * 2
    sky_ann_outer_V = CY_radV * 3
```

```
    ax1.add_patch(mpl.patches.Circle((x_V, y_V), radius=CY_radV,
                                     ec='b', linewidth=2, fill=False,␣
↪label='Photometry aperture'))
    ax1.add_patch(mpl.patches.Circle((x_V, y_V), radius=sky_ann_inner_V,
                                     ec='r', linewidth=2, fill=False,␣
↪label='Sky annulus'))
    ax1.add_patch(mpl.patches.Circle((x_V, y_V), radius=sky_ann_outer_V,
                                     ec='r', linewidth=2, fill=False))
    ax1.plot(x_V, y_V, 'g+', markersize=15, markeredgewidth=2, label='Star␣
↪center')

    ax1.set_title(f'V-band Target #{i+1}: {proc_targV_files[i]}\nPosition:␣
↪({x_V:.1f}, {y_V:.1f})')
    ax1.set_xlabel('X [pixels]')
    ax1.set_ylabel('Y [pixels]')
    ax1.legend()

    zoom_size = 600
    ax1.set_xlim(x_V - zoom_size, x_V + zoom_size)
    ax1.set_ylim(y_V - zoom_size, y_V + zoom_size)

    # R-band image
    vmin_R, vmax_R = np.percentile(im_R[np.isfinite(im_R)], [30, 99.9])
    a = ax2.imshow(im_R, cmap='gray', origin='lower', vmin=vmin_R, vmax=vmax_R)

    # R-band apertures
    sky_ann_inner_R = CY_radR * 2
    sky_ann_outer_R = CY_radR * 3

    ax2.add_patch(mpl.patches.Circle((x_R, y_R), radius=CY_radR,
                                     ec='b', linewidth=2, fill=False,␣
↪label='Photometry aperture'))
    ax2.add_patch(mpl.patches.Circle((x_R, y_R), radius=sky_ann_inner_R,
                                     ec='r', linewidth=2, fill=False,␣
↪label='Sky annulus'))
    ax2.add_patch(mpl.patches.Circle((x_R, y_R), radius=sky_ann_outer_R,
                                     ec='r', linewidth=2, fill=False))
    ax2.plot(x_R, y_R, 'g+', markersize=15, markeredgewidth=2, label='Star␣
↪center')

    ax2.set_title(f'R-band Target #{i+1}: {proc_targR_files[i]}\nPosition:␣
↪({x_R:.1f}, {y_R:.1f})')
    ax2.set_xlabel('X [pixels]')
    ax2.set_ylabel('Y [pixels]')
    ax2.legend()
```

```
    ax2.set_xlim(x_R - zoom_size, x_R + zoom_size)
    ax2.set_ylim(y_R - zoom_size, y_R + zoom_size)

    fig.colorbar(a)
    pylab.tight_layout()
    pylab.show()


# Display standard star images with apertures (V and R side-by-side)
for i in range(min(1, len(proc_photV_files))):
    # Load both V and R images
    im_V = fits.getdata(proc_data_dir + proc_photV_files[i])
    im_R = fits.getdata(proc_data_dir + proc_photR_files[i])

    x_V, y_V = stdV_xys[i]
    x_R, y_R = stdR_xys[i]

    # Create figure with two subplots
    fig, (ax1, ax2) = pylab.subplots(1, 2, figsize=(20, 10))

    # V-band image
    vmin_V, vmax_V = np.percentile(im_V[np.isfinite(im_V)], [30, 99.9])
    ax1.imshow(im_V, cmap='gray', origin='lower', vmin=vmin_V, vmax=vmax_V)

    # V-band apertures
    sky_ann_inner_V = std_radV * 2
    sky_ann_outer_V = std_radV * 3

    ax1.add_patch(mpl.patches.Circle((x_V, y_V), radius=std_radV,
                                     ec='b', linewidth=2, fill=False,␣
 ↪label='Photometry aperture'))
    ax1.add_patch(mpl.patches.Circle((x_V, y_V), radius=sky_ann_inner_V,
                                     ec='r', linewidth=2, fill=False,␣
 ↪label='Sky annulus'))
    ax1.add_patch(mpl.patches.Circle((x_V, y_V), radius=sky_ann_outer_V,
                                     ec='r', linewidth=2, fill=False))
    ax1.plot(x_V, y_V, 'g+', markersize=15, markeredgewidth=2, label='Star␣
 ↪center')

    ax1.set_title(f'V-band Standard #{i+1}: {proc_photV_files[i]}\nPosition:␣
 ↪({x_V:.1f}, {y_V:.1f})')
    ax1.set_xlabel('X [pixels]')
    ax1.set_ylabel('Y [pixels]')
    ax1.legend()

    zoom_size = 200
    ax1.set_xlim(x_V - zoom_size, x_V + zoom_size)
```

```
    ax1.set_ylim(y_V - zoom_size, y_V + zoom_size)

    # R-band image
    vmin_R, vmax_R = np.percentile(im_R[np.isfinite(im_R)], [30, 99.9])
    ax2.imshow(im_R, cmap='gray', origin='lower', vmin=vmin_R, vmax=vmax_R)

    # R-band apertures
    sky_ann_inner_R = std_radR * 2
    sky_ann_outer_R = std_radR * 3

    ax2.add_patch(mpl.patches.Circle((x_R, y_R), radius=std_radR,
                                     ec='b', linewidth=2, fill=False,␣
↪label='Photometry aperture'))
    ax2.add_patch(mpl.patches.Circle((x_R, y_R), radius=sky_ann_inner_R,
                                     ec='r', linewidth=2, fill=False,␣
↪label='Sky annulus'))
    ax2.add_patch(mpl.patches.Circle((x_R, y_R), radius=sky_ann_outer_R,
                                     ec='r', linewidth=2, fill=False))
    ax2.plot(x_R, y_R, 'g+', markersize=15, markeredgewidth=2, label='Star␣
↪center')

    ax2.set_title(f'R-band Standard #{i+1}: {proc_photR_files[i]}\nPosition:␣
↪({x_R:.1f}, {y_R:.1f})')
    ax2.set_xlabel('X [pixels]')
    ax2.set_ylabel('Y [pixels]')
    ax2.legend()

    ax2.set_xlim(x_R - zoom_size, x_R + zoom_size)
    ax2.set_ylim(y_R - zoom_size, y_R + zoom_size)

    pylab.tight_layout()
    pylab.show()
```





Photometry for all exposures Let's loop and get photometric measurements for all exposures. Try writing a function to loop over an exposure sequence, and return numpy arrays of the photometry and photometric error, each in DN/s.

[14]:
```
# Extract time offsets from FITS headers for V band
t_offs_V = []
t_start_V = None
for fn in raw_targV_files:  # Use raw files since they have the original headers
```

```
    path = os.path.join(raw_data_dir, fn)
    with fits.open(path) as hdul:
        header = hdul[0].header
        date_obs = header.get("DATE-OBS")
        if date_obs:
            exptime = header.get("EXPTIME", "N/A")
            obs_time = Time(date_obs, format='isot')
            if t_start_V is None:
                t_start_V = obs_time
            t_offs_V.append((obs_time - t_start_V).sec)
        else:
            print(f"Warning: No DATE-OBS in {fn}")

# Extract time offsets from FITS headers for R band
t_offs_R = []
t_start_R = None
for fn in raw_targR_files:  # Use raw files since they have the original headers
    path = os.path.join(raw_data_dir, fn)
    with fits.open(path) as hdul:
        header = hdul[0].header
        date_obs = header.get("DATE-OBS")
        if date_obs:
            obs_time = Time(date_obs, format='isot')
            if t_start_R is None:
                t_start_R = obs_time
            t_offs_R.append((obs_time - t_start_R).sec)
        else:
            print(f"Warning: No DATE-OBS in {fn}")
```

[15]:
```
def get_dns(gain, image_files, positions, radius, t_exp):
    """
    Perform aperture photometry on a sequence of images.

    Parameters:
    -----------
    gain : float
        Gain of the detector [e-/DN]
    image_files : list
        List of image filenames
    positions : list of tuples
        List of (x, y) positions for each image
    radius : float
        Photometric aperture radius [pixels]
    t_exp : float
        Exposure time [seconds]

    Returns:
```

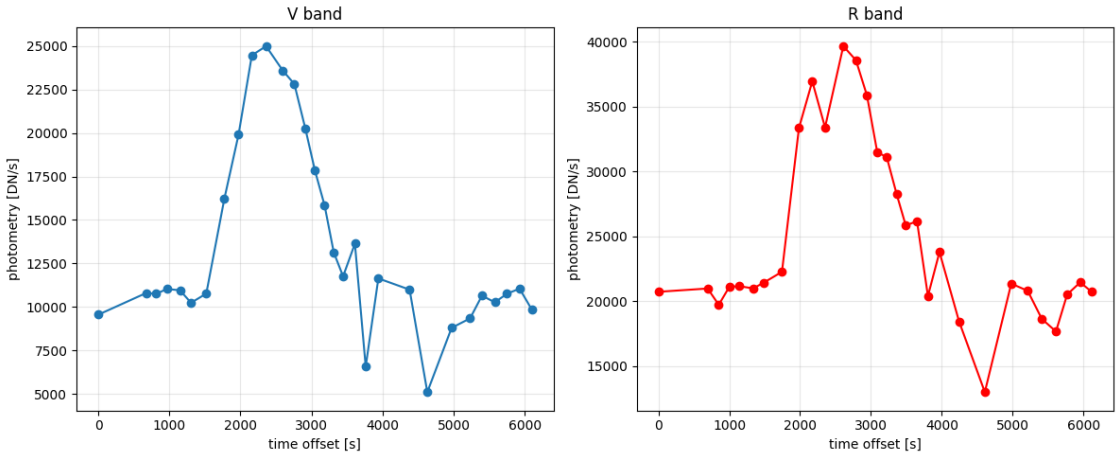
```
    --------
    F : numpy array
        Photometry measurements [DN/s]
    F_err : numpy array
        Photometry uncertainties [DN/s]
    """
    # Calculate sky annulus radii based on photometric aperture
    sky_inner = radius * 2
    sky_outer = radius * 3

    F = np.zeros(len(image_files))
    F_err = np.zeros(len(image_files))

    for i, (fn, (x, y)) in enumerate(zip(image_files, positions)):
        # Load the image
        im = fits.getdata(proc_data_dir + fn)

        # Perform aperture photometry
        phot, phot_err = ap_phot(im, x, y,
                                 radius,
                                 sky_in=sky_inner,
                                 sky_out=sky_outer,
                                 gain=gain)

        F[i] = phot
        F_err[i] = phot_err

    # Convert from DN to DN/s
    F /= t_exp
    F_err /= t_exp

    return F, F_err
```

[16]:
```
# Get photometry for all target V-band images (uses CY_radV)
F_V_targ, F_V_targ_err = get_dns(gainV, proc_targV_files, targV_xys, CY_radV,␣
 ↪exptime)

# Get photometry for all target R-band images (uses CY_radR)
F_R_targ, F_R_targ_err = get_dns(gainR, proc_targR_files, targR_xys, CY_radR,␣
 ↪exptime)

# Get photometry for all standard V-band images (uses std_radV)
F_V_std, F_V_std_err = get_dns(gainV, proc_photV_files, stdV_xys, std_radV,␣
 ↪exptime)

# Get photometry for all standard R-band images (uses std_radR)
```

```
F_R_std, F_R_std_err = get_dns(gainR, proc_photR_files, stdR_xys, std_radR,␣
↪exptime)
```

Plotting the raw photometry vs. time Let's do a quick check and examine our photometry (and errors) vs. time in each band.

[17]:
```
# Convert to numpy arrays
t_offs_V = np.array(t_offs_V)
t_offs_R = np.array(t_offs_R)

# set up figure
fig = pylab.figure(figsize=(12, 5))
ax1 = fig.add_subplot(121) # one row, two columns, first subplot
ax2 = fig.add_subplot(122) # one row, two columns, second subplot

# plot V band
ax1.errorbar(t_offs_V, F_V_targ, F_V_targ_err, fmt='o-')
ax1.set_title('V band')
ax1.set_xlabel('time offset [s]')
ax1.set_ylabel('photometry [DN/s]')
ax1.grid(True, alpha=0.3)

# plot R band
ax2.errorbar(t_offs_R, F_R_targ, F_R_targ_err, fmt='o-', color='red')
ax2.set_title('R band')
ax2.set_xlabel('time offset [s]')
ax2.set_ylabel('photometry [DN/s]')
ax2.grid(True, alpha=0.3)

# finalize plot
pylab.tight_layout()
pylab.show()
```



Calibrating the photometry We'll use the standard star photometry and its known magnitudes in each filter to calibrate our starget star photometry and put it into magnitude units.

Let's first get an average DN/s level for our standard star in each band, along with an uncertainty.

[18]:
```
F0_V = np.mean(F_V_std)  # [DN/s]
F0_V_err = np.std(F_V_std) / np.sqrt(len(F_V_std))  # [DN/s] - standard error␣
 ↪of the mean
F0_R = np.mean(F_R_std)  # [DN/s]
F0_R_err = np.std(F_R_std) / np.sqrt(len(F_R_std))  # [DN/s] - standard error␣
 ↪of the mean

print(f"Standard star V-band: {F0_V:.2f} ± {F0_V_err:.2f} DN/s")
print(f"Standard star R-band: {F0_R:.2f} ± {F0_R_err:.2f} DN/s")
```

```
Standard star V-band: 19145.03 ± 484.03 DN/s
Standard star R-band: 122210.62 ± 780.01 DN/s
```

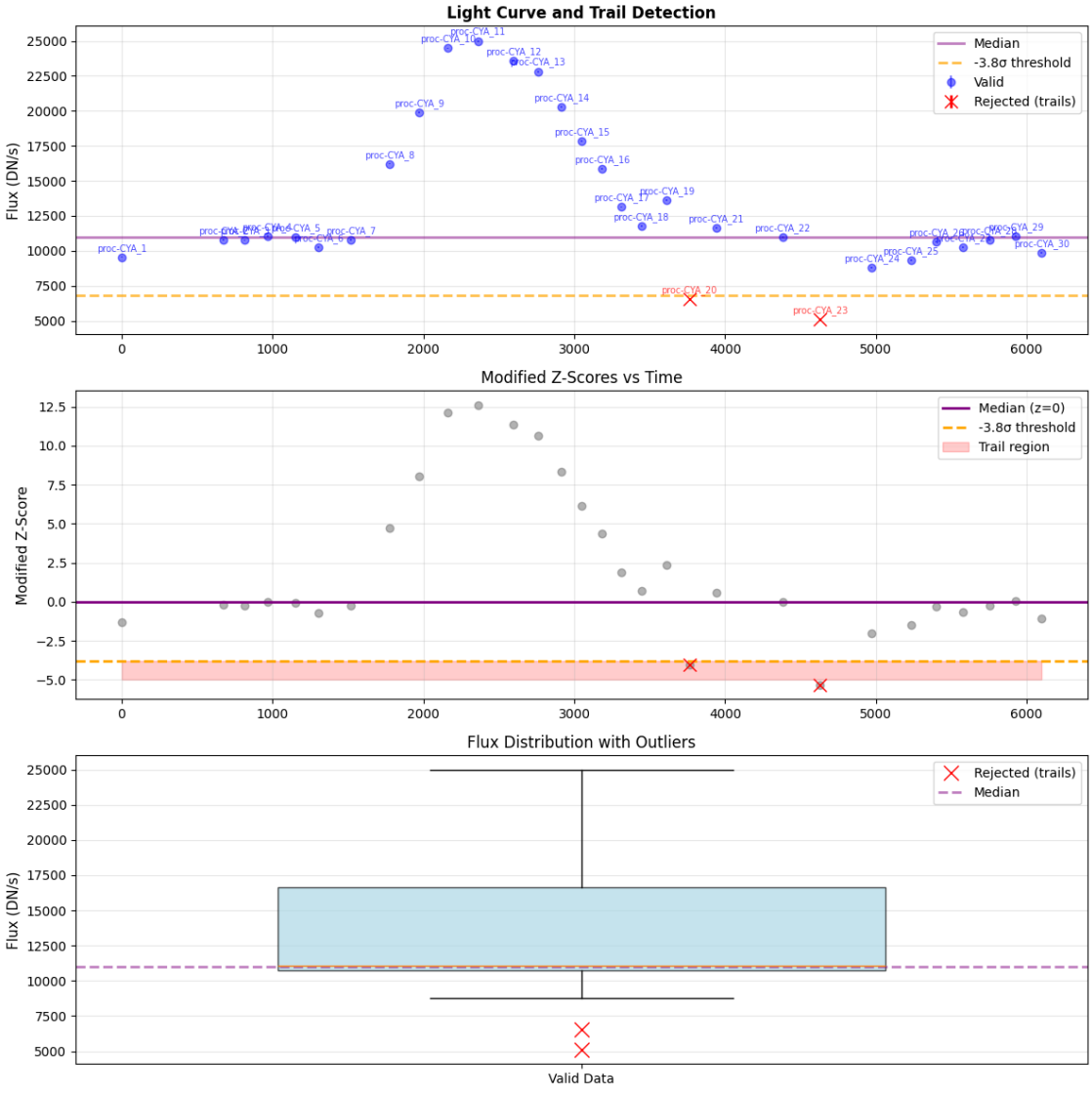
1.1.3 The second use of the module I made

of 3.8 was chosen slightly arbitrarily as it was the minimum needed to exclude the images that had lithering during the exposure

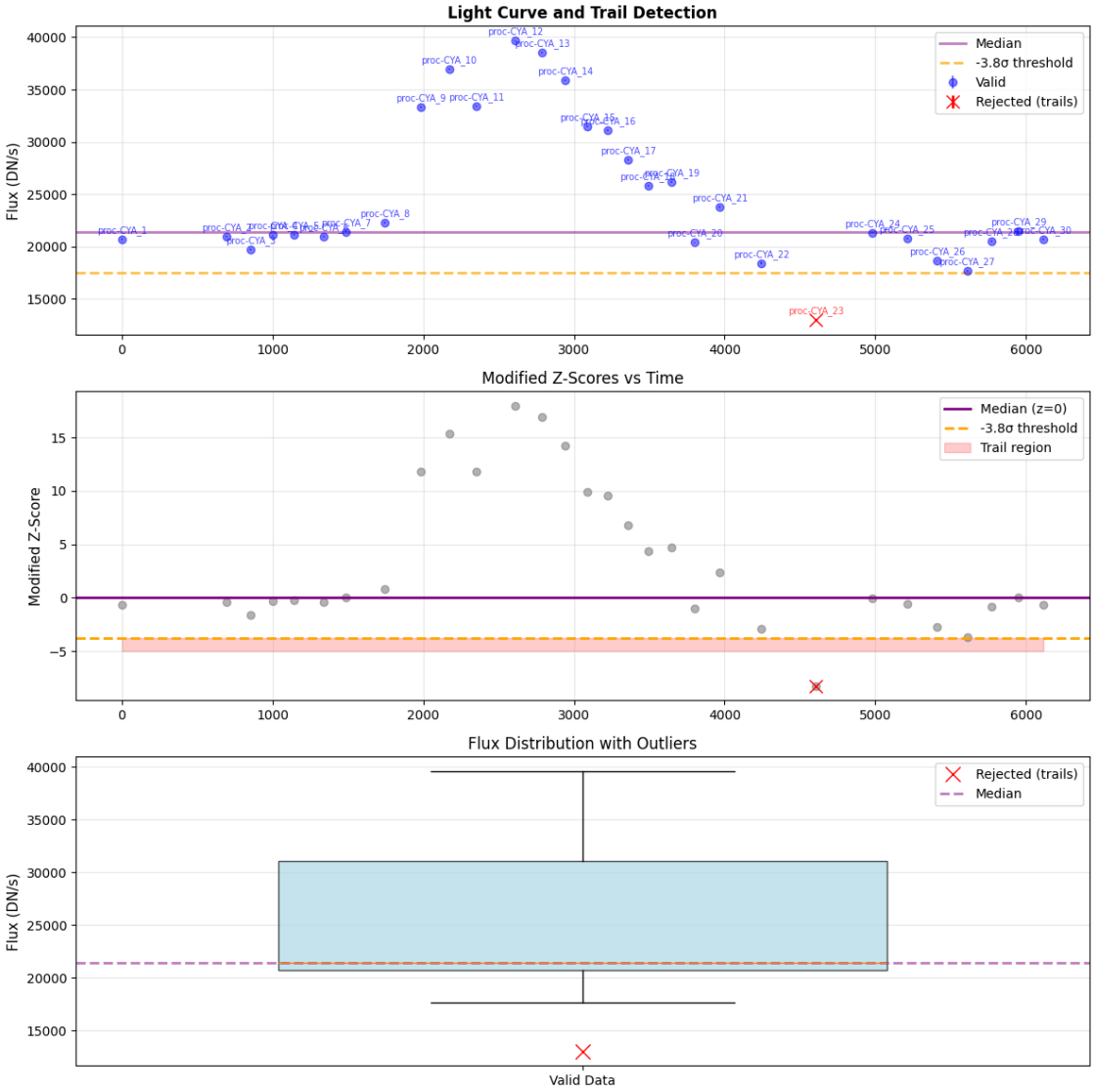
[19]:
```
# Detect trails in V-band images with plot
print("\nChecking V-band images for star trails...")
valid_V_idx = detect_trails(
    proc_targV_files, targV_xys, F_V_targ, F_V_targ_err, t_offs_V,
    outlier_threshold=3.8, plot=True
)

# Detect trails in R-band images with plot
print("\nChecking R-band images for star trails...")
valid_R_idx = detect_trails(
    proc_targR_files, targR_xys, F_R_targ, F_R_targ_err, t_offs_R,
    outlier_threshold=3.8, plot=True
)

# Filter the data
proc_targV_files = [proc_targV_files[i] for i in valid_V_idx]
proc_targR_files = [proc_targR_files[i] for i in valid_R_idx]
targV_xys = [targV_xys[i] for i in valid_V_idx]
targR_xys = [targR_xys[i] for i in valid_R_idx]
t_offs_V = t_offs_V[valid_V_idx]
t_offs_R = t_offs_R[valid_R_idx]
F_V_targ = F_V_targ[valid_V_idx]
F_V_targ_err = F_V_targ_err[valid_V_idx]
```

```
F_R_targ = F_R_targ[valid_R_idx]
F_R_targ_err = F_R_targ_err[valid_R_idx]
```

```
Checking V-band images for star trails…
============================================================
STAR TRAIL DETECTION (Modified Z-Score)
============================================================
Threshold: 3.8 modified z-score (below median only)
Median flux: 11012.3 DN/s
MAD (negative deviations): 747.3 DN/s
Equivalent sigma: 1107.9 DN/s
```



```
    TRAIL DETECTED in proc-CYA_V20.FIT:
      Measured flux: 6564.9 DN/s
      Median flux: 11012.3 DN/s
      Modified Z-Score: -4.01
      Time: 3763.0 s
    TRAIL DETECTED in proc-CYA_V23.FIT:
      Measured flux: 5097.6 DN/s
      Median flux: 11012.3 DN/s
      Modified Z-Score: -5.34
      Time: 4630.0 s

============================================================
SUMMARY: Found 2 trails, 28 good images
Excluded images: ['proc-CYA_V20.FIT', 'proc-CYA_V23.FIT']
============================================================

Checking R-band images for star trails…
============================================================
STAR TRAIL DETECTION (Modified Z-Score)
============================================================
Threshold: 3.8 modified z-score (below median only)
Median flux: 21386.9 DN/s
MAD (negative deviations): 683.8 DN/s
Equivalent sigma: 1013.8 DN/s
```




```
TRAIL DETECTED in proc-CYA_R23.FIT:
  Measured flux: 12988.9 DN/s
  Median flux: 21386.9 DN/s
  Modified Z-Score: -8.28
  Time: 4607.0 s

============================================================
SUMMARY: Found 1 trails, 29 good images
Excluded images: ['proc-CYA_R23.FIT']
============================================================
```

Now we'll use this observed flux level and the known magnitude to get a zero point (and associated uncertainty).

[20]:
```
# Known magnitudes for the standard star
m_std_V = 9.239  # [mag]
m_std_R = m_std_V - 0.800  # [mag]

# Calculate zero points using: m0 = m_standard + 2.5 * log10(F)
# where F is the observed flux in DN/s
m0_V = m_std_V + 2.5 * np.log10(F0_V)  # [mag]
m0_R = m_std_R + 2.5 * np.log10(F0_R)  # [mag]

# Calculate uncertainties using error propagation
# For m = m0 + 2.5*log10(F), dm/dF = 2.5 / (F * ln(10))
# So: sigma_m = |2.5 / (F * ln(10))| * sigma_F
m0_V_err = 2.5 / (F0_V * np.log(10)) * F0_V_err  # [mag]
m0_R_err = 2.5 / (F0_R * np.log(10)) * F0_R_err  # [mag]

print(f"Zero point V-band: {m0_V:.3f} ± {m0_V_err:.3f} mag")
print(f"Zero point R-band: {m0_R:.7f} ± {m0_R_err:.7f} mag")
```

```
Zero point V-band: 19.944 ± 0.027 mag
Zero point R-band: 21.1567724 ± 0.0069297 mag
```

Finally we can use these to calibrate our target star photometry.

[21]:
```
# calibrated target star V-band photometry [mag] with uncertainty
m_V_targ = m0_V - 2.5 * np.log10(F_V_targ)  # [mag]

# Propagate uncertainties: sigma_m^2 = sigma_m0^2 + (2.5/(F*ln(10)))^2 *␣
 ↪sigma_F^2
m_V_targ_err = np.sqrt(m0_V_err**2 + (2.5 / (F_V_targ * np.log(10)) *␣
 ↪F_V_targ_err)**2)  # [mag]

# calibrated target star R-band photometry [mag] with uncertainty
m_R_targ = m0_R - 2.5 * np.log10(F_R_targ)  # [mag]

# Propagate uncertainties
m_R_targ_err = np.sqrt(m0_R_err**2 + (2.5 / (F_R_targ * np.log(10)) *␣
 ↪F_R_targ_err)**2)  # [mag]

print(f"Target star V-band magnitudes: {m_V_targ.mean():.3f} ± {m_V_targ_err.
 ↪mean():.3f} mag (mean)")
print(f"Target star R-band magnitudes: {m_R_targ.mean():.7f} ± {m_R_targ_err.
 ↪mean():.7f} mag (mean)")

# Print all V-band measurements
print("\n" + "="*70)
print("V-BAND MEASUREMENTS")
print("="*70)
```

```
print(f"{'#':<4} {'Filename':<20} {'Time [s]':<12} {'Mag':<10} {'Error':<10}")
print("-"*70)
for i, (fn, t, m, m_err) in enumerate(zip(proc_targV_files, t_offs_V, m_V_targ,␣
 ↪m_V_targ_err)):
    print(f"{i+1:<4} {fn:<20} {t:<12.1f} {m:<10.4f} {m_err:<10.4f}")

# Print all R-band measurements
print("\n" + "="*70)
print("R-BAND MEASUREMENTS")
print("="*70)
print(f"{'#':<4} {'Filename':<20} {'Time [s]':<12} {'Mag':<10} {'Error':<10}")
print("-"*70)
for i, (fn, t, m, m_err) in enumerate(zip(proc_targR_files, t_offs_R, m_R_targ,␣
 ↪m_R_targ_err)):
    print(f"{i+1:<4} {fn:<20} {t:<12.1f} {m:<10.4f} {m_err:<10.4f}")

print("\n" + "="*70)
print(f"V-band: {len(m_V_targ)} measurements, mean = {m_V_targ.mean():.4f} ±␣
 ↪{m_V_targ_err.mean():.4f} mag")
print(f"R-band: {len(m_R_targ)} measurements, mean = {m_R_targ.mean():.4f} ±␣
 ↪{m_R_targ_err.mean():.4f} mag")
print("="*70)
```

```
Target star V-band magnitudes: 9.640 ± 0.028 mag (mean)
Target star R-band magnitudes: 10.1842565 ± 0.0078708 mag (mean)

======================================================================
V-BAND MEASUREMENTS
======================================================================
#    Filename             Time [s]     Mag        Error
----------------------------------------------------------------------
1    proc-CYA_V1.FIT      0.0          9.9938     0.0281
2    proc-CYA_V2.FIT      675.0        9.8605     0.0280
3    proc-CYA_V3.FIT      817.0        9.8652     0.0280
4    proc-CYA_V4.FIT      965.0        9.8375     0.0280
5    proc-CYA_V5.FIT      1150.0       9.8450     0.0280
6    proc-CYA_V6.FIT      1305.0       9.9187     0.0280
7    proc-CYA_V7.FIT      1520.0       9.8640     0.0280
8    proc-CYA_V8.FIT      1774.0       9.4189     0.0277
9    proc-CYA_V9.FIT      1974.0       9.1963     0.0277
10   proc-CYA_V10.FIT     2161.0       8.9724     0.0276
11   proc-CYA_V11.FIT     2361.0       8.9493     0.0276
12   proc-CYA_V12.FIT     2598.0       9.0124     0.0276
13   proc-CYA_V13.FIT     2759.0       9.0482     0.0276
14   proc-CYA_V14.FIT     2913.0       9.1769     0.0277
15   proc-CYA_V15.FIT     3049.0       9.3156     0.0277
16   proc-CYA_V16.FIT     3184.0       9.4441     0.0278
```

17	proc-CYA_V17.FIT	3314.0	9.6485	0.0279
18	proc-CYA_V18.FIT	3445.0	9.7683	0.0280
19	proc-CYA_V19.FIT	3611.0	9.6072	0.0278
20	proc-CYA_V21.FIT	3941.0	9.7794	0.0281
21	proc-CYA_V22.FIT	4381.0	9.8414	0.0282
22	proc-CYA_V24.FIT	4972.0	10.0835	0.0283
23	proc-CYA_V25.FIT	5232.0	10.0181	0.0282
24	proc-CYA_V26.FIT	5399.0	9.8761	0.0281
25	proc-CYA_V27.FIT	5580.0	9.9157	0.0281
26	proc-CYA_V28.FIT	5752.0	9.8646	0.0281
27	proc-CYA_V29.FIT	5928.0	9.8357	0.0280
28	proc-CYA_V30.FIT	6099.0	9.9599	0.0282

R-BAND MEASUREMENTS

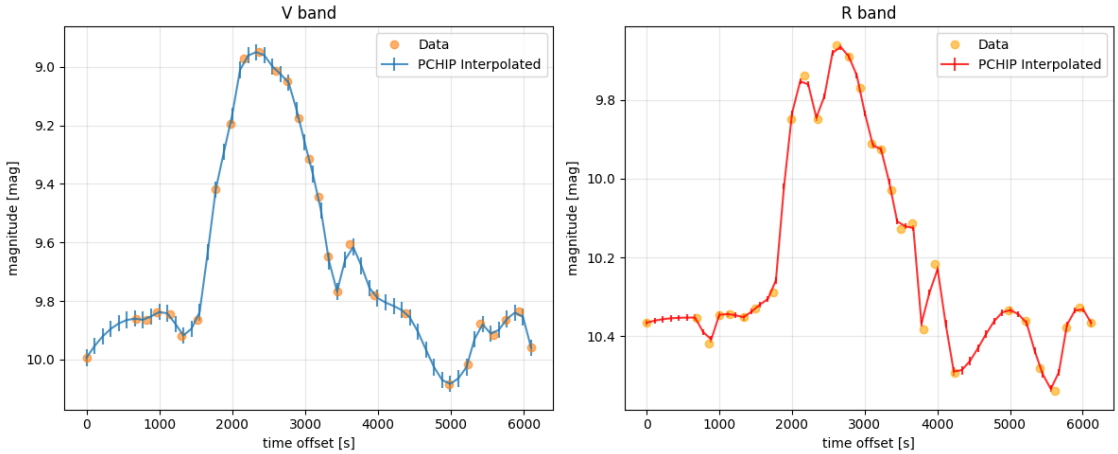
#	Filename	Time [s]	Mag	Error
1	proc-CYA_R1.FIT	0.0	10.3660	0.0079
2	proc-CYA_R2.FIT	695.0	10.3528	0.0079
3	proc-CYA_R3.FIT	852.0	10.4197	0.0080
4	proc-CYA_R4.FIT	1002.0	10.3459	0.0079
5	proc-CYA_R5.FIT	1142.0	10.3438	0.0079
6	proc-CYA_R6.FIT	1339.0	10.3521	0.0079
7	proc-CYA_R7.FIT	1489.0	10.3293	0.0079
8	proc-CYA_R8.FIT	1744.0	10.2882	0.0078
9	proc-CYA_R9.FIT	1984.0	9.8492	0.0074
10	proc-CYA_R10.FIT	2172.0	9.7384	0.0073
11	proc-CYA_R11.FIT	2352.0	9.8479	0.0074
12	proc-CYA_R12.FIT	2610.0	9.6613	0.0073
13	proc-CYA_R13.FIT	2788.0	9.6915	0.0073
14	proc-CYA_R14.FIT	2943.0	9.7704	0.0074
15	proc-CYA_R15.FIT	3089.0	9.9121	0.0075
16	proc-CYA_R16.FIT	3223.0	9.9252	0.0075
17	proc-CYA_R17.FIT	3361.0	10.0284	0.0076
18	proc-CYA_R18.FIT	3491.0	10.1267	0.0077
19	proc-CYA_R19.FIT	3650.0	10.1128	0.0077
20	proc-CYA_R20.FIT	3804.0	10.3834	0.0080
21	proc-CYA_R21.FIT	3967.0	10.2164	0.0078
22	proc-CYA_R22.FIT	4244.0	10.4932	0.0112
23	proc-CYA_R24.FIT	4980.0	10.3335	0.0080
24	proc-CYA_R25.FIT	5212.0	10.3621	0.0080
25	proc-CYA_R26.FIT	5412.0	10.4816	0.0082
26	proc-CYA_R27.FIT	5611.0	10.5394	0.0083
27	proc-CYA_R28.FIT	5769.0	10.3770	0.0080
28	proc-CYA_R29.FIT	5953.0	10.3284	0.0079
29	proc-CYA_R30.FIT	6117.0	10.3667	0.0080

```
======================================================================
V-band: 28 measurements, mean = 9.6399 ± 0.0279 mag
R-band: 29 measurements, mean = 10.1843 ± 0.0079 mag
======================================================================
```

Plotting the calibrated photometry We'll want to plot our photometry (and errors) vs. time.

[22]:
```
# Define even time steps for interpolation
n_points = 56  # number of evenly spaced points
t_V_even = np.linspace(min(t_offs_V), max(t_offs_V), n_points)
t_R_even = np.linspace(min(t_offs_R), max(t_offs_R), n_points)

# Interpolate magnitudes using PCHIP (Piecewise Cubic Hermite Interpolating␣
 ↪Polynomial)
# PCHIP preserves monotonicity and avoids oscillations/overshoots
interp_V = PchipInterpolator(t_offs_V, m_V_targ)
interp_R = PchipInterpolator(t_offs_R, m_R_targ)

# Interpolate errors using linear (errors don't need smooth curves)
interp_V_err = interp1d(t_offs_V, m_V_targ_err, kind='linear')
interp_R_err = interp1d(t_offs_R, m_R_targ_err, kind='linear')

# Generate interpolated values
m_V_interp = interp_V(t_V_even)
m_V_err_interp = interp_V_err(t_V_even)
m_R_interp = interp_R(t_R_even)
m_R_err_interp = interp_R_err(t_R_even)

# Set up figure
fig = pylab.figure(figsize=(12, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

# Plot V band magnitudes (interpolated)
ax1.errorbar(t_V_even, m_V_interp, m_V_err_interp, fmt='-',
             label='PCHIP Interpolated', alpha=0.8)
ax1.plot(t_offs_V, m_V_targ, 'o', label='Data', alpha=0.6)
ax1.set_title('V band')
ax1.set_xlabel('time offset [s]')
ax1.set_ylabel('magnitude [mag]')
ax1.invert_yaxis()
ax1.grid(True, alpha=0.3)
ax1.legend()

# Plot R band magnitudes (interpolated)
ax2.errorbar(t_R_even, m_R_interp, m_R_err_interp, fmt='-',
             color='red', label='PCHIP Interpolated', alpha=0.8)
```

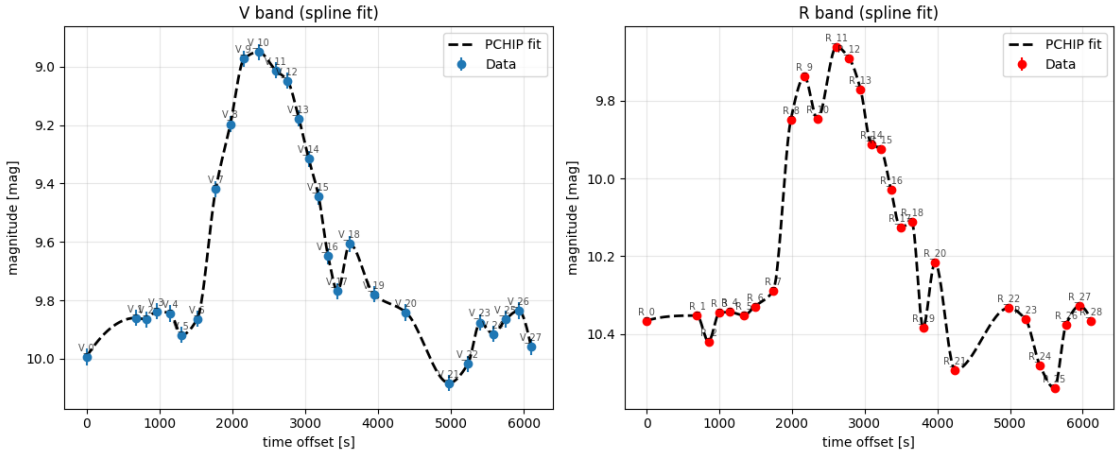
```
ax2.plot(t_offs_R, m_R_targ, 'o', color='orange', label='Data', alpha=0.6)
ax2.set_title('R band')
ax2.set_xlabel('time offset [s]')
ax2.set_ylabel('magnitude [mag]')
ax2.invert_yaxis()
ax2.grid(True, alpha=0.3)
ax2.legend()

# Finalize plot
pylab.tight_layout()
pylab.show()

print(f"V-band: Interpolated {len(t_offs_V)} data points to {n_points} even␣
 ↪points")
print(f"R-band: Interpolated {len(t_offs_R)} data points to {n_points} even␣
 ↪points")
```




```
V-band: Interpolated 28 data points to 56 even points
R-band: Interpolated 29 data points to 56 even points
```

[23]:
```
spl_V = PchipInterpolator(t_offs_V, m_V_targ)
spl_R = PchipInterpolator(t_offs_R, m_R_targ)

# Evaluate spline on even time grid
t_V_fit = np.linspace(min(t_offs_V), max(t_offs_V), 300)
t_R_fit = np.linspace(min(t_offs_R), max(t_offs_R), 300)
m_V_spline = spl_V(t_V_fit)
m_R_spline = spl_R(t_R_fit)

# Calculate residuals
residuals_V = m_V_targ - spl_V(t_offs_V)
```

```
residuals_R = m_R_targ - spl_R(t_offs_R)

# Plot results
fig = pylab.figure(figsize=(12, 5))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)

# V-band
ax1.errorbar(t_offs_V, m_V_targ, m_V_targ_err, fmt='o', label='Data')
ax1.plot(t_V_fit, m_V_spline, '--', color='black', linewidth=2, label='PCHIP␣
 ↪fit')
# Add index labels
for i in range(len(t_offs_V)):
    ax1.annotate(f'V_{i}', (t_offs_V[i], m_V_targ[i]),
                 textcoords="offset points", xytext=(0,5), ha='center',
                 fontsize=7, alpha=0.7)
ax1.set_title('V band (spline fit)')
ax1.set_xlabel('time offset [s]')
ax1.set_ylabel('magnitude [mag]')
ax1.invert_yaxis()
ax1.grid(True, alpha=0.3)
ax1.legend()

# R-band
ax2.errorbar(t_offs_R, m_R_targ, m_R_targ_err, fmt='o', color='red',␣
 ↪label='Data')
ax2.plot(t_R_fit, m_R_spline, '--', color='black', linewidth=2, label='PCHIP␣
 ↪fit')
# Add index labels
for i in range(len(t_offs_R)):
    ax2.annotate(f'R_{i}', (t_offs_R[i], m_R_targ[i]),
                 textcoords="offset points", xytext=(0,5), ha='center',
                 fontsize=7, alpha=0.7)
ax2.set_title('R band (spline fit)')
ax2.set_xlabel('time offset [s]')
ax2.set_ylabel('magnitude [mag]')
ax2.invert_yaxis()
ax2.grid(True, alpha=0.3)
ax2.legend()

pylab.tight_layout()
pylab.show()
```



Computing a color We'll want to look at $V-R$ color vs. time as well. One complication is that we don't have the same timestamps for our V and R exposures, so we can't just subtract them.

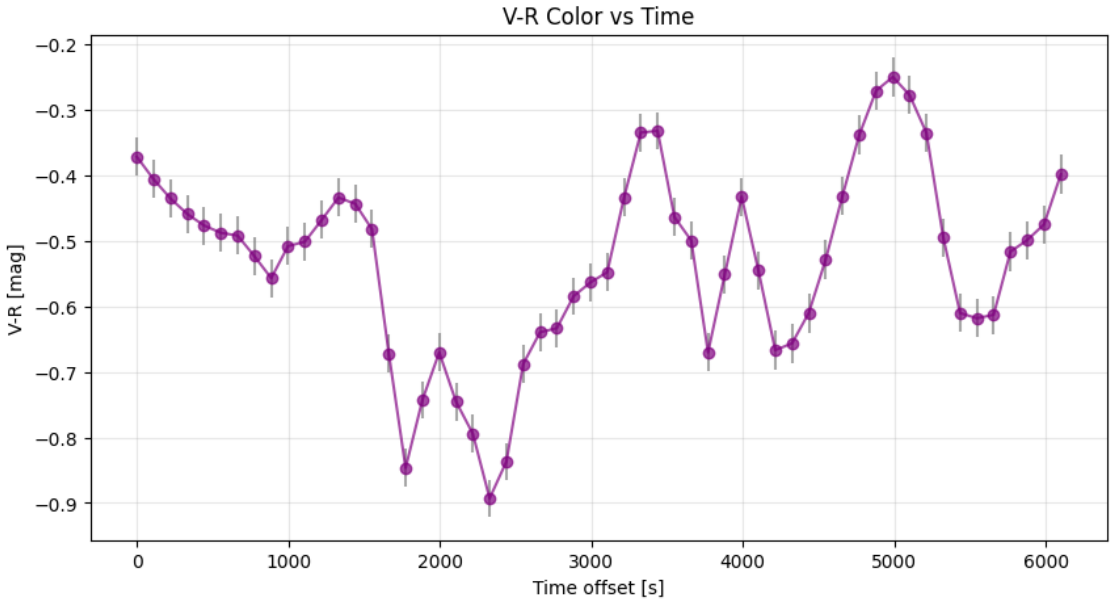
We can get a sense of the color change by interpolating the time sequences.

[24]:
```
# Interpolate both bands to a common time grid
n_points = 56
t_common = np.linspace(max(min(t_V_even), min(t_R_even)), min(max(t_V_even),␣
 ↪max(t_R_even)), n_points)

# Interpolate V and R onto the common grid
m_V_common = interp_V(t_common)
m_R_common = interp_R(t_common)
m_V_err_common = interp_V_err(t_common)
m_R_err_common = interp_R_err(t_common)

# Compute V-R color
color_VR = m_V_common - m_R_common

# Estimate uncertainties (quadrature sum)
color_VR_err = np.sqrt(m_V_err_common**2 + m_R_err_common**2)
```

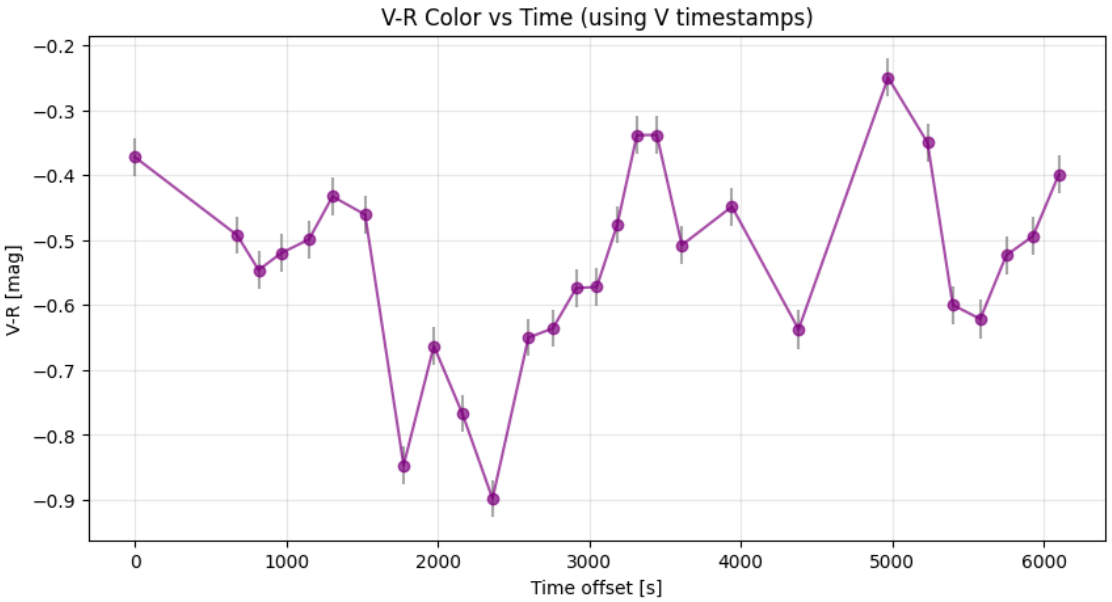
Plotting the color Let's see the results.

[25]:
```
# Plot V-R color vs time
fig, ax = pylab.subplots(figsize=(10, 5))
ax.errorbar(t_common, color_VR, yerr=color_VR_err, fmt='o-', color='purple',␣
 ↪ecolor='gray', alpha=0.7)
ax.set_title('V-R Color vs Time')
ax.set_xlabel('Time offset [s]')
ax.set_ylabel('V-R [mag]')
```

```
ax.grid(True, alpha=0.3)
pylab.show()
```



[26]:
```
# Interpolate R to V-band timestamps
m_R_at_V = interp_R(t_offs_V)        # R magnitudes at V observation times
m_R_err_at_V = interp_R_err(t_offs_V)  # R errors interpolated to V times

# Compute V-R color
color_VR = m_V_targ - m_R_at_V

# Estimate uncertainties using quadrature sum
color_VR_err = np.sqrt(m_V_targ_err**2 + m_R_err_at_V**2)

# Plot V-R color vs V-band time
fig, ax = pylab.subplots(figsize=(10, 5))
ax.errorbar(t_offs_V, color_VR, yerr=color_VR_err, fmt='o-', color='purple',␣
 ↪ecolor='gray', alpha=0.7)
ax.set_title('V-R Color vs Time (using V timestamps)')
ax.set_xlabel('Time offset [s]')
ax.set_ylabel('V-R [mag]')
ax.grid(True, alpha=0.3)
pylab.show()
```


V-R Color vs Time (using V timestamps)
V-R [mag]
−0.2
−0.3
−0.4
−0.5
−0.6
−0.7
−0.8
−0.9
0
1000
2000
3000
4000
5000
6000
Time offset [s]