Deblurring and Enhancing Astrophotographs Using PSF Based Digital Image Processing

Group Number: 19

Weekly Report 6

Group Members:

- Hetanshu Parmar
- Purvish Parekh
- Tarjani Patel
- Vrushti Patel
- Ansh Rathva

1 Introduction

To meet the project objective, we applied Stacking, which combined 3 images of the same region with differing wavelengths alongside Wiener filtering and deconvolution to the images. It was done with the intention of retaining the original information of the FITS image. We had used a simple Gaussian PSF to implement this. However, after not getting the desired output after applying multiple processing techniques, we realised that the imaging system of HST has different PSFs at different regions of the image. Hence, it would be more suitable to model a PSF such that it takes into account the separate, varying PSFs of the stars in the image. This also involved modifying our dataset, which includes isolated stars, to facilitate better PSF estimation. For this week, we tried to estimate the PSF from star clusters captured by and specific to the Wide Field Camera(WFC) of the Hubble Space Telescope.

2 Methodology

For estimating the PSF, we first take an isolated star from a star cluster. For that, we took the dataset of isolated stars from Hubble. The isolated star was then centred and taken out as a 31x31 stamp for ease of access. The background is also removed from the stamp to avoid ringing effects due to fainter background noise. This results in the single brightest star at the centre with a zero-intensity background around it. This is necessary for precisely measuring the PSF and to avoid any deviation due to background intensities. Firstly, we modelled the PSF from 56 isolated star images and combined those individual PSFs into a single global PSF by stacking, averaging them out, and fitting the PSF to a Gaussian function.

Below you can see the spread image of isolated stars. It can be observed that the spread is not uniform for all the directions and therefore it can not be modelled using an ideal Gaussian PSF, which we were using previously. Therefore, modeling the PSF specific to the instrument can help generating better quality results.

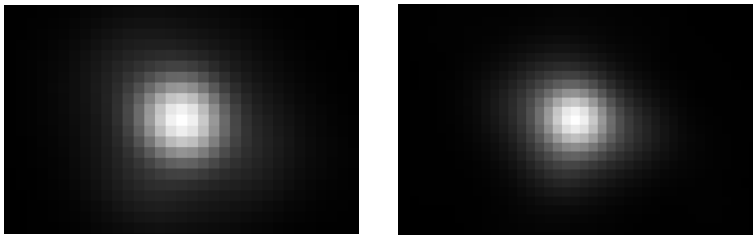

Figure 1: Example PSFs

The graph shows the average counts, representing the number of photons captured by the CCD (Charge-Coupled Device), versus the physical distance between the CCD sensors. We can observe from the graph that it tends to idea Gaussian spread; however, it is not completely similar to it.

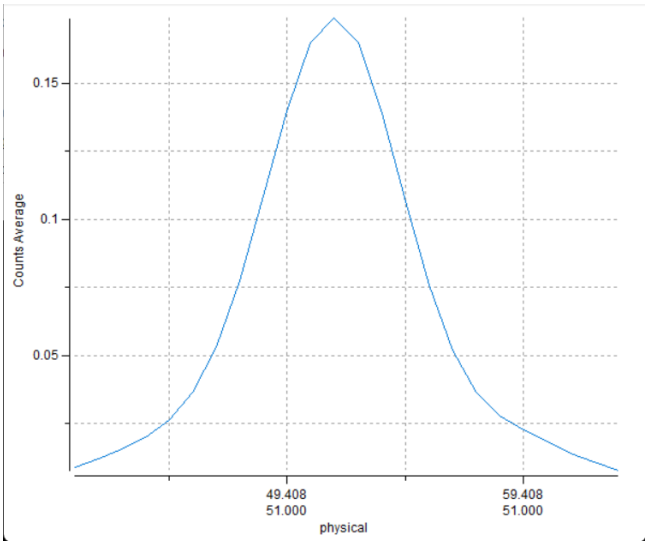

Figure 2: Raw Input Image

Below is the code used for PSF extraction form the provided isolated star images.

2.1 Code:

```
def build_average_psf_from_folder(stars_folder, stamp_size=31,
↪  verbose=True, use_daofind=True,
                                  daofind_fwhm=2.5,
                                  ↪  daofind_thresh_sigma=5.0):
    """Load images from a folder, extract star stamps, center
    ↪  them, stack, and fit an average Gaussian PSF.

    Parameters
    ----------
    stars_folder : str
        Path to folder containing FITS or image files of isolated
        ↪  stars (each file may contain one star).
    stamp_size : int
        Size of square stamp to extract (odd recommended).
    use_daofind : bool
        If True, try to run DAOStarFinder on each image and use
        ↪  the best detected star; otherwise use brightest
        ↪  pixel.
    daofind_fwhm, daofind_thresh_sigma : float
        Parameters passed to DAOStarFinder (FWHM guess, detection
        ↪  threshold in sigma).

    Returns
    -------
    dict
        Contains 'stack_psf', 'gauss_psf',
        ↪  'gauss_psf_from_params', 'fitted_params',
        ↪  'kept_files'.
    """
    patterns = ['*.fits', '*.fit', '*.fz', '*.png', '*.jpg',
    ↪  '*.jpeg', '*.tif', '*.tiff']
    files = []
    for patt in patterns:
        files += sorted(glob.glob(os.path.join(stars_folder,
        ↪  patt)))
    if len(files) == 0:
        raise FileNotFoundError(f"No star stamp images found in
        ↪  {stars_folder}")

    stamps = []
    fitted_params = []
    kept_files = []

    # Lazy import of DAOStarFinder to avoid hard dependency if
    ↪  not used
```

```
if use_daofind:
    from photutils.detection import DAOStarFinder

for f in files:
    try:
        img = read_image_any(f)
        # ensure 2D
        if img.ndim != 2:
            continue
        # Subtract simple local background from stamp-level
        ↪  image
        m, med, s = sigma_clipped_stats(img, sigma=3.0)
        img_bkgsub = img - med

        # Find star position
        if use_daofind:
            try:
                daofind = DAOStarFinder(fwhm=daofind_fwhm,
                ↪  threshold=daofind_thresh_sigma * s)
                sources = daofind(img_bkgsub)
            except Exception:
                sources = None
            if sources is None or len(sources) == 0:
                # fall back to brightest-pixel
                yy, xx = np.unravel_index(np.nanargmax(img_b⌋
                ↪  kgsub), img_bkgsub.shape)
            else:
                # sort by flux and pick the brightest
                ↪  non-saturated candidate
                sources.sort('flux')
                brightest = sources[-1]
                xx = brightest['xcentroid']; yy =
                ↪  brightest['ycentroid']
        else:
            yy, xx =
            ↪  np.unravel_index(np.nanargmax(img_bkgsub),
            ↪  img_bkgsub.shape)

        # Make a stamped cutout centered on (yy,xx)
        half = stamp_size // 2
        y0 = int(np.round(yy)) - half; y1 = int(np.round(yy))
        ↪  + half + 1
        x0 = int(np.round(xx)) - half; x1 = int(np.round(xx))
        ↪  + half + 1
        # handle edges with padding
```

```
stamp = np.zeros((stamp_size, stamp_size),
↪  dtype=float)
y0_src = max(0, y0); y1_src = min(img.shape[0], y1)
x0_src = max(0, x0); x1_src = min(img.shape[1], x1)
y0_dst = y0_src - y0; y1_dst = y0_dst + (y1_src -
↪  y0_src)
x0_dst = x0_src - x0; x1_dst = x0_dst + (x1_src -
↪  x0_src)
stamp[y0_dst:y1_dst, x0_dst:x1_dst] =
↪  img_bkgsub[y0_src:y1_src, x0_src:x1_src]

# Clip negatives and subtract local median to reduce
↪  background bias
m2, med2, s2 = sigma_clipped_stats(stamp, sigma=3.0)
stamp = stamp - med2
stamp[stamp < 0] = 0.0

# center stamp with subpixel precision
try:
    stamp_centered = center_stamp_subpixel(stamp)
except Exception:
    stamp_centered = stamp

total = stamp_centered.sum()
if total <= 0:
    if verbose:
        print(f"Skipping {os.path.basename(f)}:
        ↪  non-positive flux after bg-sub.")
    continue

# normalize before stacking (so we capture shape not
↪  flux differences)
stamp_norm = stamp_centered / total
stamps.append(stamp_norm)
kept_files.append(f)

# fit gaussian to centered stamp and save params
try:
    gfit, model = fit_gaussian_2d(stamp_centered)
    fitted_params.append({
        'amplitude': float(gfit.amplitude.value),
        'x_mean': float(gfit.x_mean.value),
        'y_mean': float(gfit.y_mean.value),
        'x_stddev': float(gfit.x_stddev.value),
        'y_stddev': float(gfit.y_stddev.value),
```

```
                'theta': float(getattr(gfit, 'theta',
                ↪  0.0).value if hasattr(gfit, 'theta') else
                ↪  0.0)
            })
        except Exception:
            # ignore fit failures for single stamps
            pass

        if verbose:
            print(f"Processed {os.path.basename(f)}
            ↪  (sum={total:.1f})")

    except Exception as e:
        if verbose:
            print(f"Skipping {os.path.basename(f)}: {e}")
        continue

if len(stamps) == 0:
    raise RuntimeError("No usable star stamps found in
    ↪  folder.")

# Stack stamps (mean) and fit Gaussian to stack
stack = np.mean(np.stack(stamps), axis=0)
gfit_stack, model_stack = fit_gaussian_2d(stack)
gauss_psf = model_stack
gauss_psf = gauss_psf / gauss_psf.sum()

# Alternative: average parameters and synthesize gaussian PSF
gauss_psf_from_params = None
if len(fitted_params) > 0:
    keys = fitted_params[0].keys()
    avg_params = {k: np.mean([p[k] for p in fitted_params])
    ↪  for k in keys}
    y, x = np.indices(stack.shape)
    g_avg =
    ↪  models.Gaussian2D(amplitude=avg_params['amplitude'],
                         x_mean=avg_params['x_mean'],
                         y_mean=avg_params['y_mean'],
                         x_stddev=avg_params['x_stddev'
                         ↪  ],
                         y_stddev=avg_params['y_stddev'
                         ↪  ],
                         theta=avg_params.get('theta',
                         ↪  0.0))
    gauss_psf_from_params = g_avg(x, y)
    gauss_psf_from_params /= gauss_psf_from_params.sum()
```

```
return {
    'stack_psf': stack,
    'gauss_psf': gauss_psf,
    'gauss_psf_from_params': gauss_psf_from_params,
    'fitted_params': fitted_params,
    'kept_files': kept_files
}
```

3 Results

The shown image is the result generated by processing RGB wavelength intensity data separately and combining the data into the RGB channels of the colored image.

Figure 3: Output Image

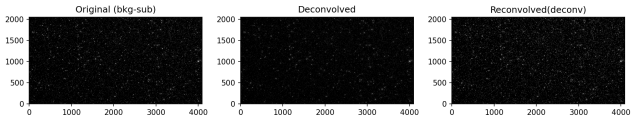

Figure 4: Output Comparison

4 Problems Faced:

- Since, the size of dataset was too big, the processing was computationally heavy.

- The RAW FITS image obtained, missed information since it was completely uncalibrated which significantly affected the output quality and even distorted the color mapping.
- It was difficult to generate PSF of each separate star especially since star clusters generally has multiple stars in close proximity.
- Color mapping done previously, produced distorted images implying it was not done correctly.

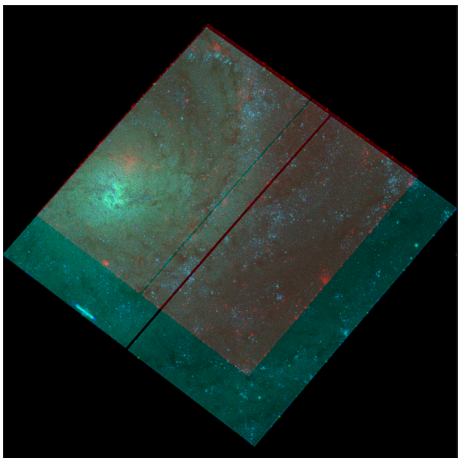

Figure 5: Differing Sizes of Images

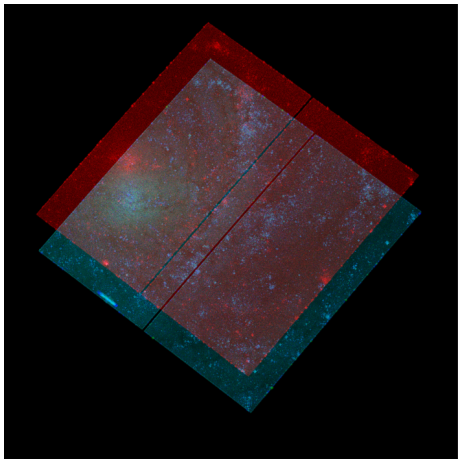

Figure 6: Misalign Images

5 Next Set of Work:

In the upcoming week, we plan to:

- Expand the data set to obtain a more refined output
- Quantitatively measure the PSF applied.
- Apply deconvolution and filtering on the image to deblur it.

6 Conclusion

For this week, we changed our PSF extraction methodology and decided to model the PSF of the image as the average of the individual PSFs of the star in the image. Next, we will implement deconvolution and filtering on this image and after that is achieved, we will focus on expanding the dataset to make the output more robust.

References

[1] E. J. Groth, "A pattern recognition program for cosmic ray rejection on CCD images," *The Astronomical Journal*, vol. 118, no. 4, pp. 1764–1776, 2006. [Online]. Available: https://iopscience.iop.org/article/10.1086/510117/pdf)

[2] Mikulski Archive for Space Telescopes (MAST), "MAST Portal." [Online]. Available: https://mast.stsci.edu/portal/Mashup/Clients/Mast/Portal.html

[3] NASA, "How are Webb's full-color images made?," [Online]. Available: https://science.nasa.gov/mission/webb/science-overview/science-explainers/how-are-webbs-full-color-images-made/