

# Deblurring and Enhancing Astrophotographs Using PSF Based Digital Image Processing

Group Number: 19

Weekly Report 5

## Group Members:

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

## 1 Introduction

Until last week, we had implemented stacking and also processed RGB wavelength intensity data separately and then combined the data into RGP channels of the colored image. However, the output obtained was of poor quality implying that the stacking was misaligned and color mapping was not implemented correctly. Also, the dataset images had a blank strip across the image, due to lack of information in that region. This was due to the imaging system, hence the dataset had to be changed such that the dataset should contain pre-processed/calibrated images, so as to generate better quality output. So, this week we implement color mapping again using a slightly different methodology.

## 2 Methodology

The methodology used so far includes using an Ideal Gaussian PSF to deconvolve the PSF using RL deconvolution after applying the Wiener Filter to remove noise. Contrast stretching is also applied for image enhancement purposes. This method is applied to three individual images, each with data at 428, 555, and 841 nanometer wavelengths, corresponding to the blue, green, and red channels, respectively. Thus, a fully processed RGB image is formed.

Below is the code for the image processing pipeline, which includes contrast stretching, Wiener filtering, and Richard-Lucy deconvolution for individual blue, green, and red images, and stacking them to create the final colored image.

### 2.1 Code:

```
import numpy as np
import cv2
```

```

import matplotlib.pyplot as plt
from skimage import img_as_float, restoration, exposure,
↳ transform
from numpy.fft import fft2, ifft2
from astropy.io import fits

def make_preview(img, max_dim=2048):
    h, w = img.shape[:2]
    scale = min(1.0, float(max_dim) / max(h, w))
    if scale < 1.0:
        return transform.resize(img, (int(h * scale), int(w *
↳ scale)), anti_aliasing=True)
    return img.copy()

# ===== Utility Functions =====

def read_fits(path):
    with fits.open(path) as hdul:
        data = None
        header = None
        for hdu in hdul:
            if getattr(hdu, 'data', None) is not None:
                data = hdu.data
                header = hdu.header
                break

        if data is None:
            raise ValueError(f"No image data found in any HDU of
↳ {path}")

    if hasattr(data, 'filled'):
        data = data.filled(np.nan)

    data = np.asarray(data)

    if data.dtype == object:
        try:
            data = data.astype(np.float64)
        except Exception:
            flat = []
            for v in data.flat:
                try:
                    flat.append(float(v))
                except Exception:
                    flat.append(0.0)

```

```

        data = np.array(flat,
            ↪ dtype=np.float64).reshape(data.shape)

data = np.nan_to_num(data, nan=0.0, posinf=0.0, neginf=0.0)

data = data.astype(np.float64, copy=False)
return img_as_float(data)

def gaussian_psf(size=7, sigma=1.5):
    ax = np.linspace(-(size-1)/2, (size-1)/2, size)
    xx, yy = np.meshgrid(ax, ax)
    psf = np.exp(-(xx**2 + yy**2) / (2 * sigma**2))
    psf /= psf.sum()
    return psf

def wiener_filter(img, kernel, K=0.0001):
    kernel /= np.sum(kernel)
    dummy = fft2(img)
    kernel_fft = fft2(kernel, s=img.shape)
    wiener = np.conj(kernel_fft) / (np.abs(kernel_fft)**2 + K)
    result = np.abs(iff2(dummy * wiener))
    return result

def preprocess_channel(img, psf, K=0.0001, num_iter=30):
    wiener_out = wiener_filter(img, psf, K)
    p2, p98 = np.percentile(wiener_out, (60, 98))
    stretched = exposure.rescale_intensity(wiener_out,
        ↪ in_range=(p2, p98))

    deconv = restoration.richardson_lucy(stretched, psf,
        ↪ num_iter=num_iter, clip=True)
    p2, p98 = np.percentile(deconv, (60, 98))
    final = exposure.rescale_intensity(deconv, in_range=(p2,
        ↪ p98))
    return final

# ===== Load FITS Images (update paths accordingly)
↪ =====
path_B = "C:/Users/pnpar/Downloads/Image_Stacking_Data/5/j96r230_
↪ 11_435_drc.fits" # Blue channel
path_G = "C:/Users/pnpar/Downloads/Image_Stacking_Data/5/j96r230_
↪ 11_555_drc.fits" # Green channel
path_R = "C:/Users/pnpar/Downloads/Image_Stacking_Data/5/j96r230_
↪ 11_658_drc.fits" # Red channel
img_B = read_fits(path_B)
img_G = read_fits(path_G)

```

```

img_R = read_fits(path_R)

# ===== PSF and Processing =====
psf = gaussian_psf(size=7, sigma=1.5)

B_proc = preprocess_channel(img_B, psf)
G_proc = preprocess_channel(img_G, psf)
R_proc = preprocess_channel(img_R, psf)

# ===== Stack into RGB =====
rgb = cv2.merge((B_proc, G_proc, R_proc))

# ===== Display =====
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(make_preview(img_R), cmap='gray')
plt.title("Raw HST (F814W - Red)")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(make_preview(img_G), cmap='gray')
plt.title("Raw HST (F555W - Green)")
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(make_preview(img_B), cmap='gray')
plt.title("Raw HST (F435W - Blue)")
plt.axis('off')
plt.tight_layout()
plt.show()

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(make_preview(R_proc), cmap='gray')
plt.title("Processed HST (F814W - Red)")
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(make_preview(G_proc), cmap='gray')
plt.title("Processed HST (F555W - Green)")
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(make_preview(B_proc), cmap='gray')
plt.title("Raw HST (F435W - Blue)")
plt.axis('off')

```

```

plt.tight_layout()
plt.show()

plt.figure(figsize=(10, 8))
plt.imshow(make_preview(rgb))
plt.title("True-Color HST Composite (Wiener + RL)")
plt.axis('off')
plt.show()

# ===== Save =====
cv2.imwrite("HST_ColorComposite.png", cv2.cvtColor((rgb *
→ 255).astype(np.uint8), cv2.COLOR_RGB2BGR))

```

### 3 Results

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

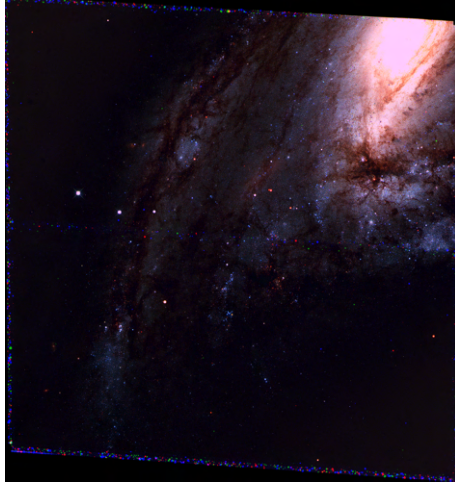


Figure 1: Edge Star Misalignments

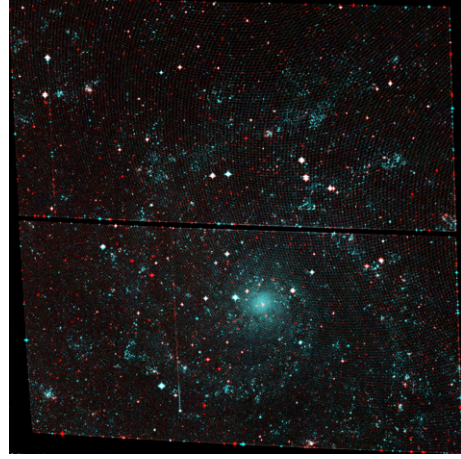


Figure 2: Red Channle Misaligned Image

The shown results do not uphold the integrity of the image as the misaligned stars and colour channels reduce the image quality. The resizing option to align the images while stacking leads to some stars being out of order or the entire channel being out of alignment with the other images.

### 4 Problems Faced:

- The output quality worsened such that the color mapping distorted the image.
- The problem of misalignment in stacking persisted which also affected the output quality.
- The same methodology could not be implemented to a larger dataset, since it proved to be unsuccessful.

### 5 Next Set of Work:

In the upcoming week, we plan to:

- Find a better dataset with the same dimension for the entire set of three images.

- Implement a different PSF based on the instrument used.

## 6 Conclusion

For this work, we implemented color mapping again in order to produce better quality image with distinguishable features, however this method was unsuccessful. Next, we intend to change the methodology of extracting PSF (instrument ased PSF) as well as the dataset to achieve the desired results.

## 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/>