ESSE 2220 – Lab 7 Report

Detecting Features in Space, SIFT and ORB in Action

Course: ESSE 2220
Lab Title: Detecting Features in Space, SIFT and ORB in Action **Date Performed:** 2025-11-07
Date Submitted: 2025-11-07

Names & Group Info

Group Number: GROUP 5
Members: Yathharthha Kaushal · Owen Oliver

---

1. Image Loading and Channels:

1.1 Code output for IMGREAD_GRAYSCALE:

```
img = cv2.imread('space.jpg', cv2.IMREAD_GRAYSCALE)
```

1.2 Code output for IMGREAD_COLOR:

```
img = cv2.imread('space.jpg', cv2.IMREAD_COLOR)
```

1.3 Printed shapes:

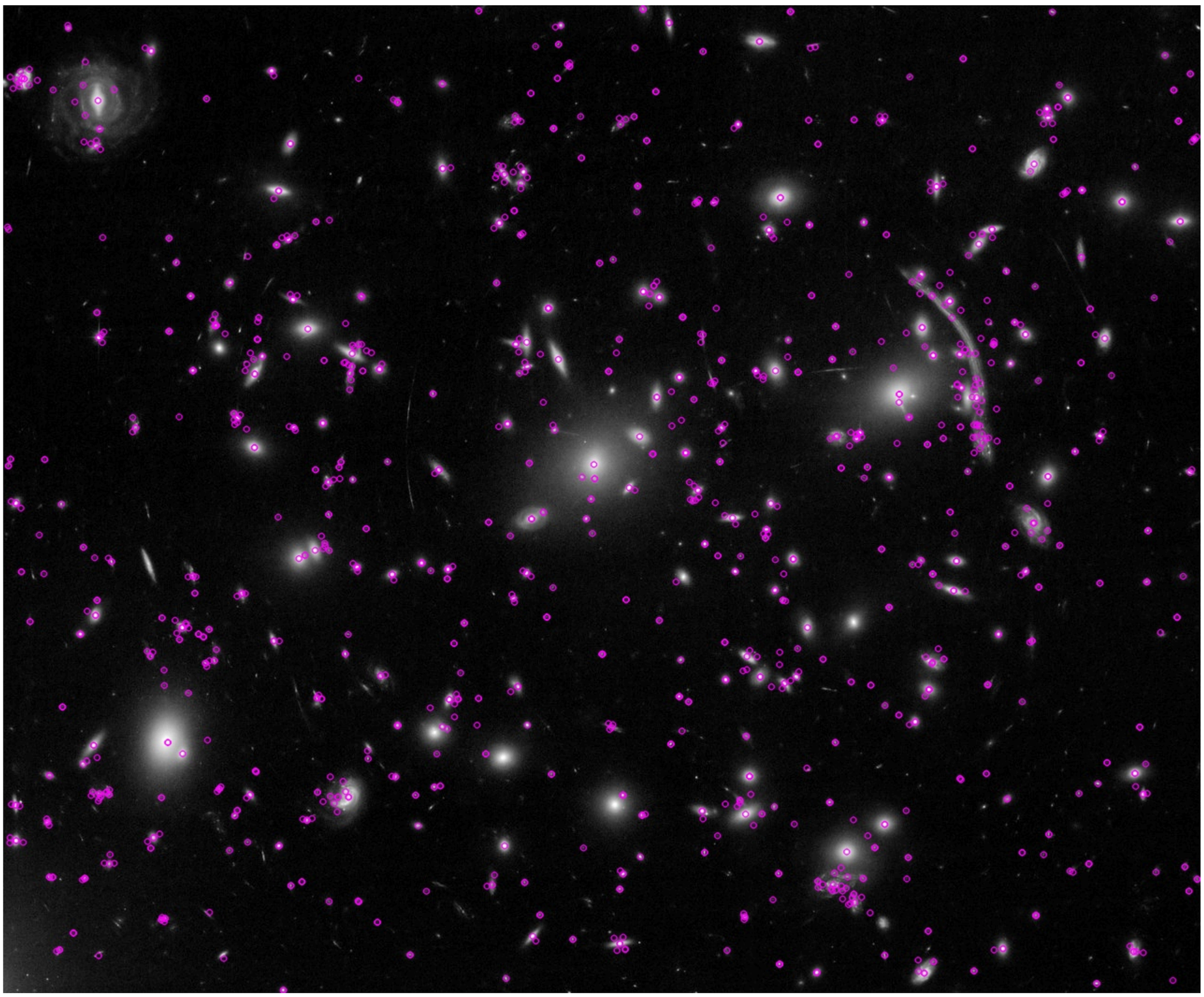
> Grayscale image shape: (1056, 1280) Color image shape: (1056, 1280, 3)

1.4 What changed and shape differences:

> The grayscale image has a shape of (1056, 1280), indicating it has only two dimensions (height and width). The color image has a shape of (1056, 1280, 3), indicating it has three dimensions - the first two can be implied as the height and width, and the third dimension represents the number of the color channels (Red, Green, Blue).

2. SIFT Detection and Visualization:

2.1 Sift Keypoint image:



2.2 Color chosen for keypoints

2.2.1 Color chosen:

> Bright magenta (RGB: 255, 0, 255) was chosen because the bright it rarely occurs in raw starfield imagery, and is easily distinguishable from the dark background, and is not confused with common blue/red nebulas.

2.2.2 Explain how I designed it:

> I looked for a color that would: stand out visibly against the dark background of space and not be confused with colors commonly found in starts, nebulae, other space objects. Picking the color was rather trivial, as it was just full red and full blue, with no green.

2.3 Number of keypoints detected and descriptor shape:

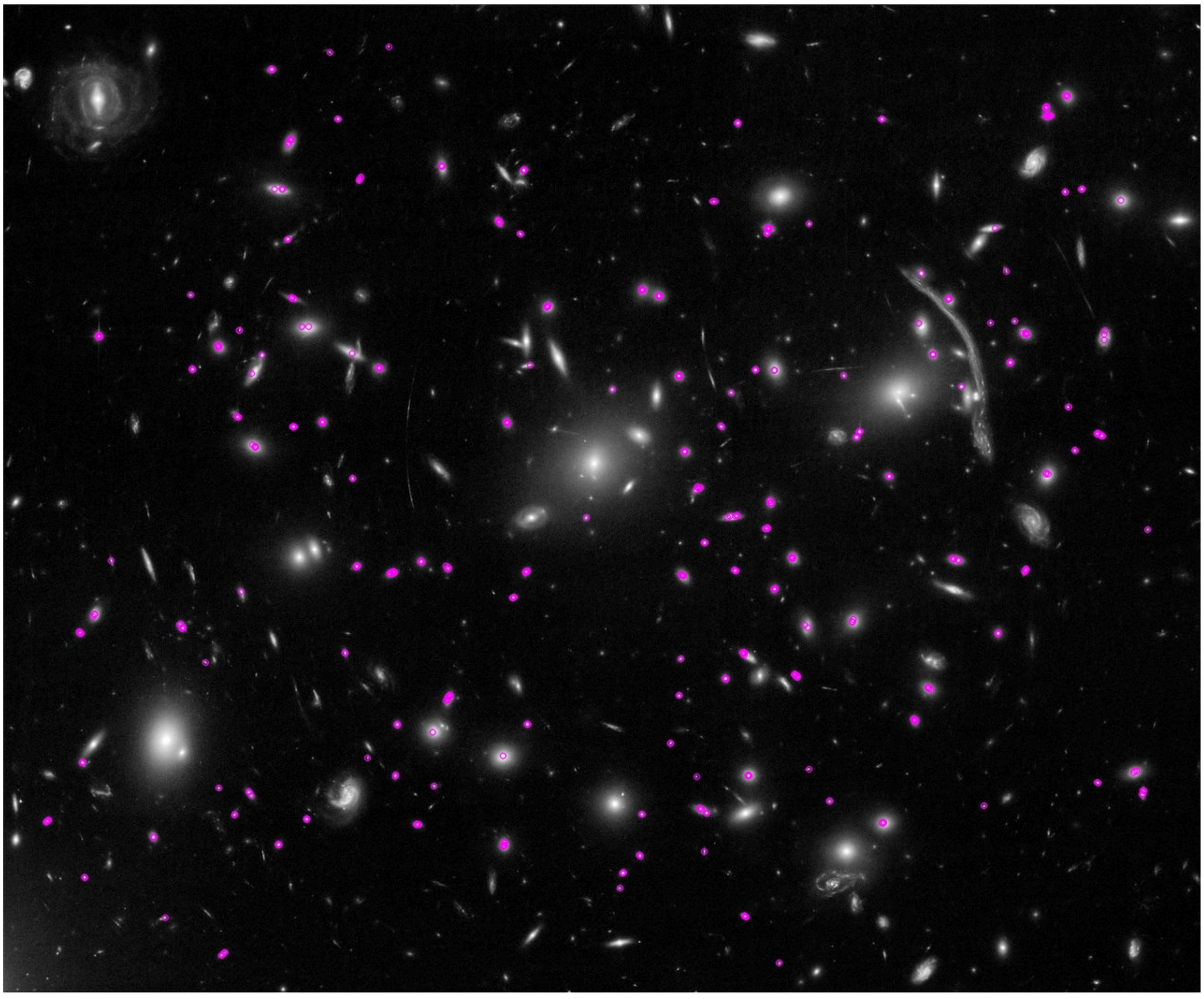
> Number of keypoints (SIFT): 1391
> Descriptor shape (SIFT): (1391, 128)

2.4 Explanation of keypoints:

> A keypoint is a certain feature in the image that is considered interesting or important for analysis. In the context of SIFT (Scale-Invariant Feature Transform), keypoints are specific locations in the image that are invariant to scale, rotation, and illumination changes. These keypoints are typically corners, edges, or blobs in the image that can be reliably detected and used for tasks such as image matching, object recognition, and 3D reconstruction.

3. ORB Detection and Visualization:

3.1 ORB Keypoint image:



3.2 Color chosen for keypoints

> The reasoning is the same as in 2.2

3.3 Number of keypoints detected and descriptor shape:

> Number of keypoints (ORB): 500 Descriptor shape (ORB): (500, 32)

3.4 What values did I use for nfeatures.

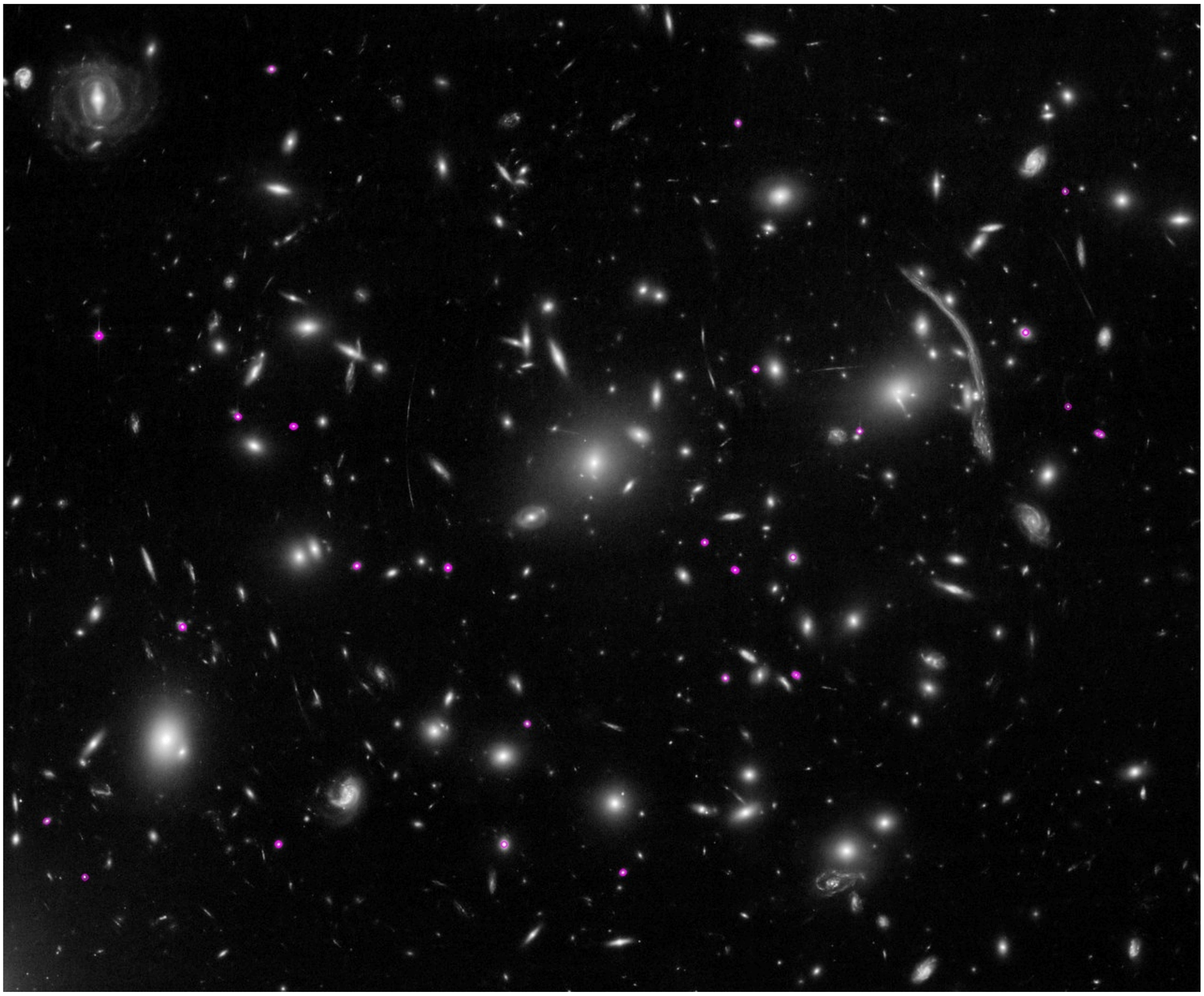
> For the first iteration, I set the value as 500, and the number of detection points was relatively high For the second iteration, I set the value as 5, and the number of detection points was much smaller.

4. Strongest Keypoint

4.1 Top 50 Keypoint images:

4.1.1 SIFT Top 50 Keypoints:

4.1.2 ORB Top 50 Keypoints:



4.2 What does the response for the keypoint represent?

> The response for a keypoint represents the strength of a keypoint (i.e. how unique the feature is), the higher the value, the easier it is to recognize that keypoint in different images or under different conditions.

4.3 Why might strong keypoints be useful in later matching steps?

> Strong keypoints will be useful in matching steps because they are more likely to be easily identifiable/distinguisable between multiple images, which would make them more reliable for matching features across images.

5. Descriptor Values:

5.1 First few descriptor values for SIFT and ORB:

> First two SIFT descriptor values:

In [ ]:
```
[ 11.   2.   0.   0.   0.   1.   2.  10.  25.  13.   1.   0.   0.   0.
  17.  32.  12.   3.   0.   0.   0.  34.  79.  28.   4.   1.   0.   0.
   0.  55.  29.   6.   8.   3.   2.   2.   2.   1.   1.   8.  84.  47.
  11.   2.   1.   0.   2.  12. 150.  18.   0.   0.   3.  55.  84. 137.
  12.   1.   0.   0.  79. 150.  66.  24.   2.   1.   0.   2.   6.   3.
   2.   3.  93.  20.   1.   1.   2.   1.   4.  22. 150. 150.  11.   5.
  13.  13.   8.  35.  42.  51.  22.  52. 150. 107.   9.   9.   0.   0.
   0.   0.   0.   0.   0.   0.  21.  23.   0.   0.   0.   0.   0.   0.
  80. 150.  25.   4.   1.   0.   0.   0.   7.  68.  61. 112.  58.   1.
   0.   0.]
```

In [ ]:
```
[ 14.   1.   1.   0.   0.   5.   7.  13. 139.   0.   0.   0.   3.   2.
   3. 139.  50.   0.   0.  27. 139.  16.   5.  65.   0.   0.   0.  35.
 112.  13.   0.   0.  61.   9.   1.   0.   0.   0.   0.   8. 139.  30.
   5.  11.  20.   1.   1.  82.  52.   8.   5. 131. 139.   4.   1.  23.
   0.   0.   3.  62.  97.   0.   0.   0.  42.  23.   1.   0.   0.   0.
   0.   0. 138. 139.  92.  30.   6.   0.   0.   2.  11.  29.  87. 139.
 110.   0.   0.   1.   0.   0.   1.  32.  37.   7.   5.   1.   2.  10.
   2.   0.   0.   0.   0.   0.   2.  29.  48.   4.   0.   0.   0.   1.
   4.   5.  39.  24.   1.   0.   0.   3.   5.   2.   1.   3.   1.   6.
   5.   4.]
```

> First two ORB descriptor values:

In [ ]:
```
[244  96 145 100   8  41  65   8 162 178 160  24 215 211  73  26  17 220
 104  24  10  71 218   2 217 132   5  60 224  49   2  32]
```

In [ ]:
```
[252 108 148 117 168  42  69 187 242 186 174  88 211  49 193  27   1  70
 108  58 121  97 203  32 232 226 117  52 232 241 194  32]
```

5.2 Explanation of descriptor vectors:

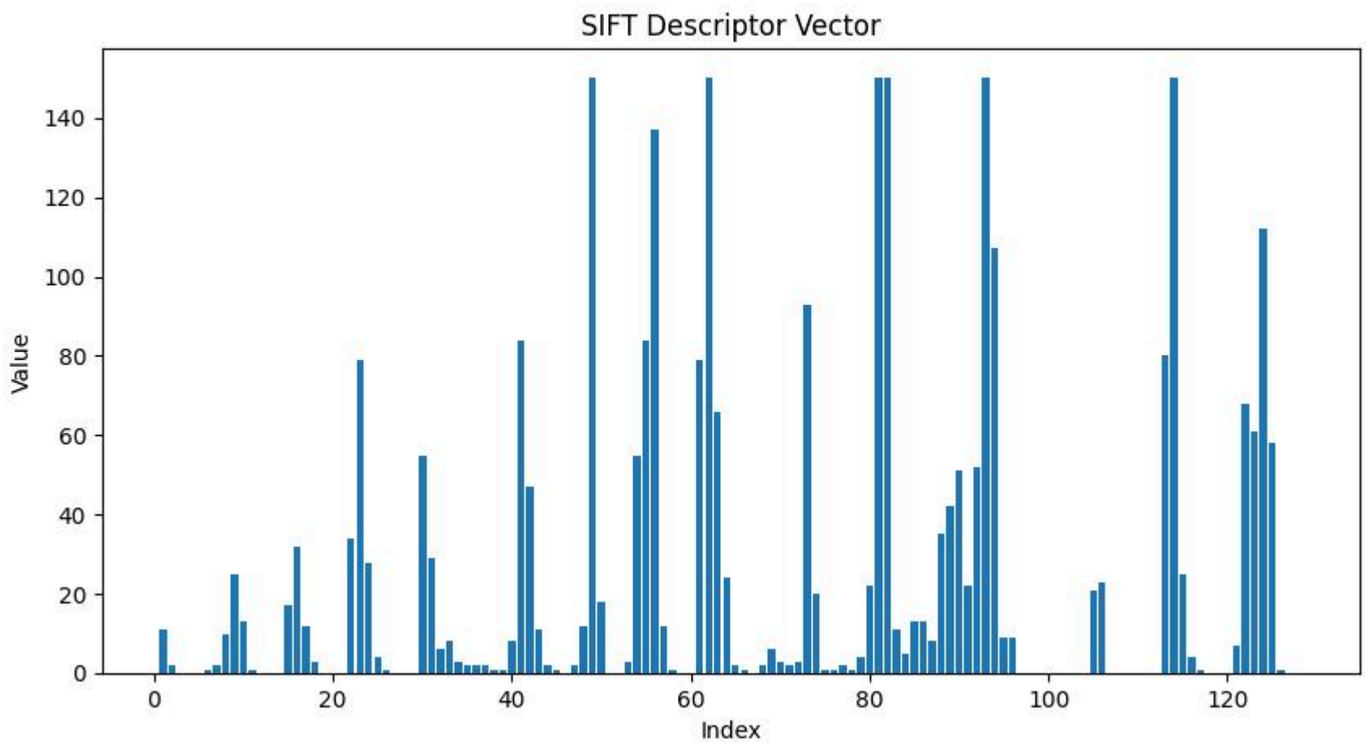
> A descriptor vector represents the local image patch around a keypoint in a compact numerical format. It captures the essential features of the image patch, such as texture, intensity gradients, and orientation. Descriptor vectors are used to compare and match keypoints between different images.

5.3 Describe one difference between SIFT and ORB descriptors:

> One difference between SIFT and ORB descriptors is that SIFT descriptors are 128-dimensional floating-point vectors, while ORB descriptors are 32-dimensional binary vectors. This means that SIFT descriptors provide a more detailed representation of the local image patch, while ORB descriptors are more compact and faster for matching (more suitable for real-time applications).

6. Descriptor Visualization:

6.1 SIFT Descriptor Plot:



6.2 Matplotlib functions used for plotting:

`figure()` - This was used to create a new blank figure for the plot.
`bar()` - This was used for defining that the plot type is a bar chart, and provide the raw data to be plotted.
`title()`, `xlabel()`, `ylabel()` - Pretty self-explanatory, these functions were used to set the title and axis labels for the plot.
`savefig()` - Saved the generated plot as an image file.
`close()` - Prevents memory leaks by closing the plot after saving.

6.3 Explanation of the plot:

> The plot represents the values of the SIFT descriptor for the first keypoint. The x-acis represents the index of each element in the descriptor vector (starting from 1 and ending at 128), while the y-axis represents the corresponding value of each element in the descriptor vector. The height of each bar indicates the direction of the feature in that dimension, with taller bars indicating stronger features.

7. Program:

In [ ]:
```
"""
Lab 7: Feature Detection in Space Images (SIFT & ORB)
------------------------------------------------------
Instructions:
1. Fill in the missing parts using the OpenCV documentation.
2. You must look up how to use the functions marked with TODO.
3. Use the official docs: https://docs.opencv.org/
   or Google search like: cv2.SIFT_create site:docs.opencv.org
4. Save all your output images using cv2.imwrite().
"""

import cv2
import numpy as np
import matplotlib.pyplot as plt

# --- Step 1: Read the image ---
img = cv2.imread('space.jpg', cv2.IMREAD_GRAYSCALE)

# TODO: Try IMREAD_COLOR instead.
img_color = cv2.imread('space.jpg', cv2.IMREAD_COLOR)
# Then print img.shape for both cases and describe what changes.
print("Grayscale image shape:", img.shape)
print("Color image shape:", img_color.shape)
# What does the shape tell you about the number of channels (grayscale vs color)?
# Example: print(img.shape)
"""Grayscale image has two dimensions (height, width), whereas
the Color image has three dimensions (height, width, 3(we can hypothesize number of color channels (i.e. RGB)))"""


# --- Step 2: Create SIFT detector ---
# TODO: Find how to create a SIFT detector object in the docs.
sift = cv2.SIFT_create()

# --- Step 3: Detect keypoints and descriptors ---
# TODO: Look up how to detect and compute both at once.
keypoints_sift, descriptors_sift = sift.detectAndCompute(img, None)

# --- Step 4: Print some info ---
print("Number of keypoints (SIFT):", len(keypoints_sift))
print("Descriptor shape (SIFT):", descriptors_sift.shape)

# --- Step 5: Draw SIFT keypoints ---
# TODO: Look up drawKeypoints function in docs.
# Search what flags can be used (hint: something about “rich keypoints”).
# Design your own color for SIFT visualization using an RGB tuple, e.g., (R, G, B).
# Write in your report what color it represents and how you designed it.
img_sift = cv2.drawKeypoints(img, keypoints_sift, None, (255, 0, 255))
"""This color was picked because the bright magenta color rarely occurs in raw
starfield imagery, and is easily distinguishable from the dark background, and is
not cofused with common blue/red nebulas"""

# Save the output image.
cv2.imwrite('sift_keypoints.jpg', img_sift)

# --- Step 6: Create ORB detector ---
# TODO: Find ORB_create and its parameters (e.g., nfeatures).
# Hint: A common starting value is around 500, but try different values (like 100, 1000)
# and describe how changing this number affects the number of detected keypoints.
orb = cv2.ORB_create(nfeatures=500)

# --- Step 7: Detect and compute with ORB ---
keypoints_orb, descriptors_orb = orb.detectAndCompute(img, None)

print("Number of keypoints (ORB):", len(keypoints_orb))
print("Descriptor shape (ORB):", descriptors_orb.shape)
```

```python
# --- Step 8: Draw ORB keypoints ---
# TODO: Draw keypoints using a different RGB color than SIFT.
# You must also add appropriate flags for drawing.
# Explain what color you chose and why.
img_orb = cv2.drawKeypoints(img, keypoints_orb, None, (255, 0, 255))
cv2.imwrite('orb_keypoints.jpg', img_orb)

# --- Step 9: Top 50 strongest keypoints ---
# TODO: Use the Python function "sorted" to sort keypoints by their response value.
# You must Google what sorted() does and how to use the "key" parameter.
# "response" measures how strong or distinctive a detected feature is — higher = stronger feature.
# Then select the top 50 keypoints for both SIFT and ORB and visualize them.
# Save the resulting images as 'sift_top50.jpg' and 'orb_top50.jpg'.
def getResponse(kp):
    return kp.response

sorted_sift = sorted(keypoints_sift, key=getResponse, reverse=True)
sorted_orb = sorted(keypoints_orb, key=getResponse, reverse=True)

sorted_sift = sorted_sift[:50]
sorted_orb = sorted_orb[:50]

#for i, (kps, kpo) in enumerate(zip(sorted_sift, sorted_orb)):
#    print(i, kps.response, kpo.response)


print("Top 50 SIFT keypoints responses:", len(sorted_sift))

cv2.imwrite("sift_top50.jpg", cv2.drawKeypoints(img, sorted_sift, None, (255, 0, 255)))
cv2.imwrite("orb_top50.jpg", cv2.drawKeypoints(img, sorted_orb, None, (255, 0, 255)))

# --- Step 10: Print and explain descriptors ---
# Print the first few descriptor values for each method.
# Example: print(descriptors_sift[:2]) and print(descriptors_orb[:2])
# In your report, explain what these descriptor values represent and how they differ.
print(descriptors_sift[:2])
print(descriptors_orb[:2])

# --- Step 11: Visualize one descriptor vector ---
# TODO: Use matplotlib to create a bar chart of one descriptor vector.
# Hint: Search “matplotlib bar chart site:matplotlib.org”
# Also check how to install matplotlib if you don’t have it installed.
# (Hint: pip install matplotlib)
# Save the figure as 'sift_descriptor_plot.jpg'.
plt.figure(figsize=(10, 5))
plt.bar(list(range(1, len(descriptors_sift[0]) + 1)), descriptors_sift[0])
plt.title('SIFT Descriptor Vector')
plt.xlabel('Index')
plt.ylabel('Value')
plt.savefig('sift_descriptor_plot.jpg')
plt.close()
```