main

November 14, 2024

[105]:
```
#LAB-5 EECE-5554-ROBOT SENSING AND NAVIGATION - CAMERA MOSAIC - PROFESSOR SINGH          SHRIMAN RAGHAV SRINIVASAN
```

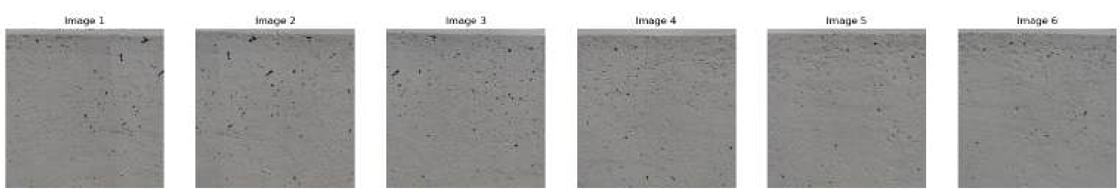
[107]:
```
#I have first loaded the white cinder block dataset in the purpose of showcasing the gradual reduction in RANSAC threshold value
#to improve homography accuracy and increase number of keypoints
```

[109]:
```
#loading 1st data of white cinder blocks
```

[71]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from utils import *
from harris import *
```

[73]:
```
path = "C:\\Users\\ragsh\\Desktop\\FALL 24\\RSN\\LAB5\\LAB5\\data\\cinder"
images = load_images(path)
```

```
Images loaded successfully!
```



[74]:
```
def detect_and_compute_keypoints(img,show_corners = False,harris_params={}):
    """
    Detect keypoints using Harris Corner Detector and compute descriptors using SIFT.
    """
    # Initialize SIFT and convert image to grayscale
    sift = cv2.SIFT_create()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    y_coords, x_coords, response = harris(gray, **harris_params)

    keypoints = [cv2.KeyPoint(float(x), float(y), 1) for y, x in zip(y_coords,␣
↪x_coords)]

    # Compute descriptors for detected keypoints using SIFT
    keypoints, descriptors = sift.compute(img, keypoints)

    ## displaying harris corner detection output
    if show_corners:
        img_with_corners = img.copy()
        for y, x in zip(y_coords, x_coords):
            img_with_corners[int(y), int(x)] = [0, 0, 255]  # Mark corners in␣
↪red
        plt.figure(figsize=(8, 8))
        plt.imshow(cv2.cvtColor(img_with_corners, cv2.COLOR_BGR2RGB))
        plt.title("Custom Harris Corners Detection")
        plt.axis("off")
        plt.show()



    return keypoints, descriptors
```

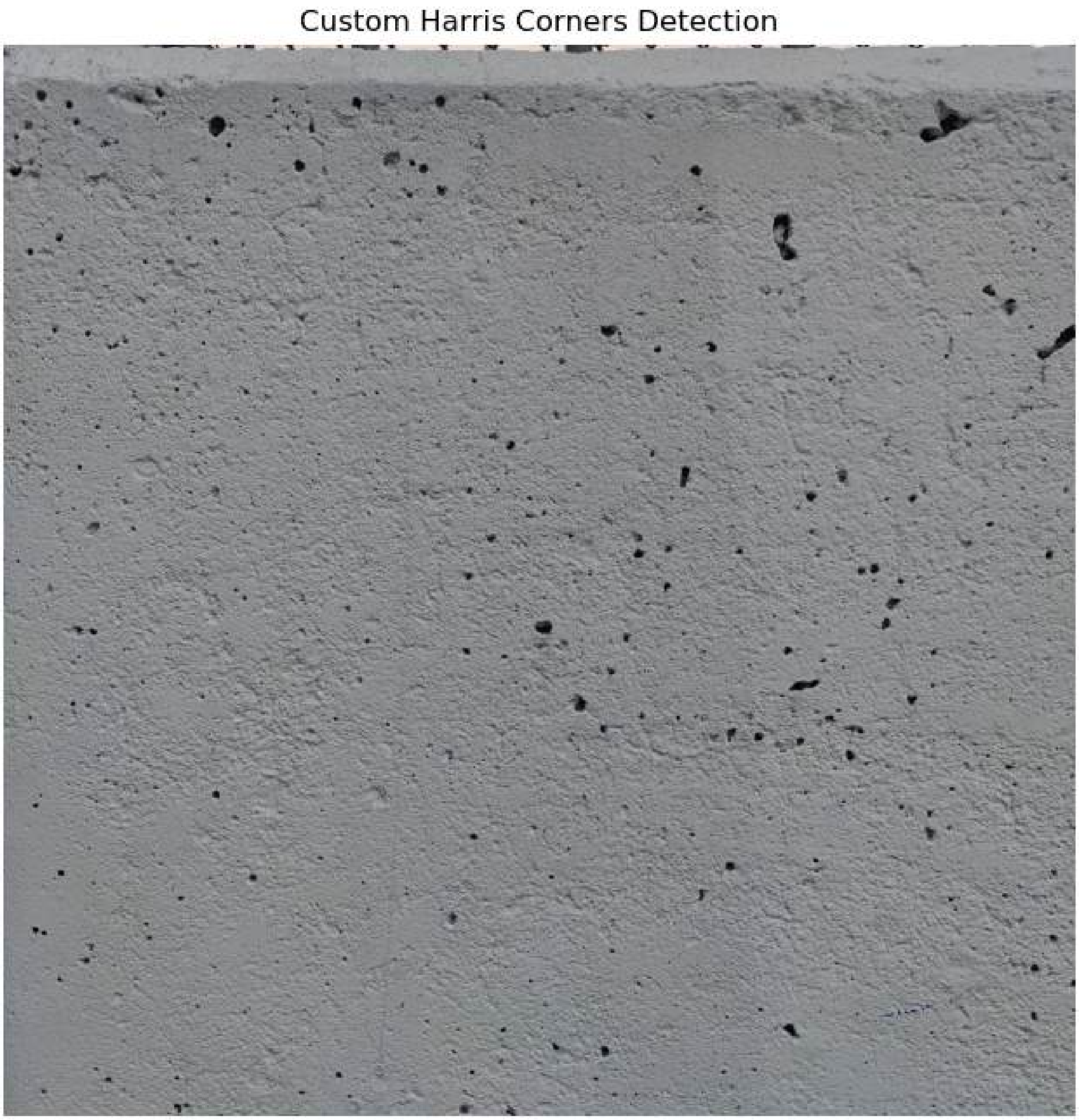
[77]:
```
def stitch_images(img1, img2, kp1, kp2, matches):
    """
    Stitch two images together using matched keypoints and RANSAC to find␣
↪homography.
    This version adds padding based on detected alignment (left or right).
    """
    # Extract matched keypoints
    img1_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1,␣
↪2)
    img2_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1,␣
↪2)

    # Compute homography matrix using RANSAC
    H, mask = cv2.findHomography(img1_pts, img2_pts, cv2.RANSAC, 8.0) #RANSAC␣
↪Value kept high
    img1_warped, img2_padded = warpPerspectivePadded(img1, img2, H)
    stitched_image = masking(img1_warped, img2_padded, 0.1)
    result = np.uint8(stitched_image)

    return result
```

[79]:
```python
def create_panorama(images, show_corners=False, harris_params={}):
    """
    Create a panorama from a list of images.
    """
    stitched_image = images[0]  # Start with the first image
    for i in range(1, len(images)):
        print(f"Stitching image {i-1} and {i}...")

        # Detect keypoints and descriptors with custom Harris and SIFT
        keypoints1, descriptors1 = detect_and_compute_keypoints(stitched_image,␣
↪show_corners=show_corners, harris_params=harris_params)
        keypoints2, descriptors2 = detect_and_compute_keypoints(images[i],␣
↪show_corners=show_corners, harris_params=harris_params)

        # Match features between descriptors
        matches = match_features(descriptors1, descriptors2)
        print(f"Number of good matches between stitched image and image {i}:␣
↪{len(matches)}")

        # Optionally draw matches
        draw_matches(stitched_image, images[i], keypoints1, keypoints2,␣
↪matches, num_matches=50)

        # Stitch the current image with the next in sequence
        stitched_image = stitch_images(stitched_image, images[i], keypoints1,␣
↪keypoints2, matches)


    return stitched_image
```

[81]:
```python
harris_params = {

        'disp': False,
        'N': 250,
        'thresh': 0.01,
        'hsize': 2,
        'sigma': 0.6,
        'eig': True,
        'tile': [1,1],
        'mask': None,
        'fft': False,
    }
panorama = create_panorama(images, show_corners=True,␣
↪harris_params=harris_params)


# Display the final panorama
```

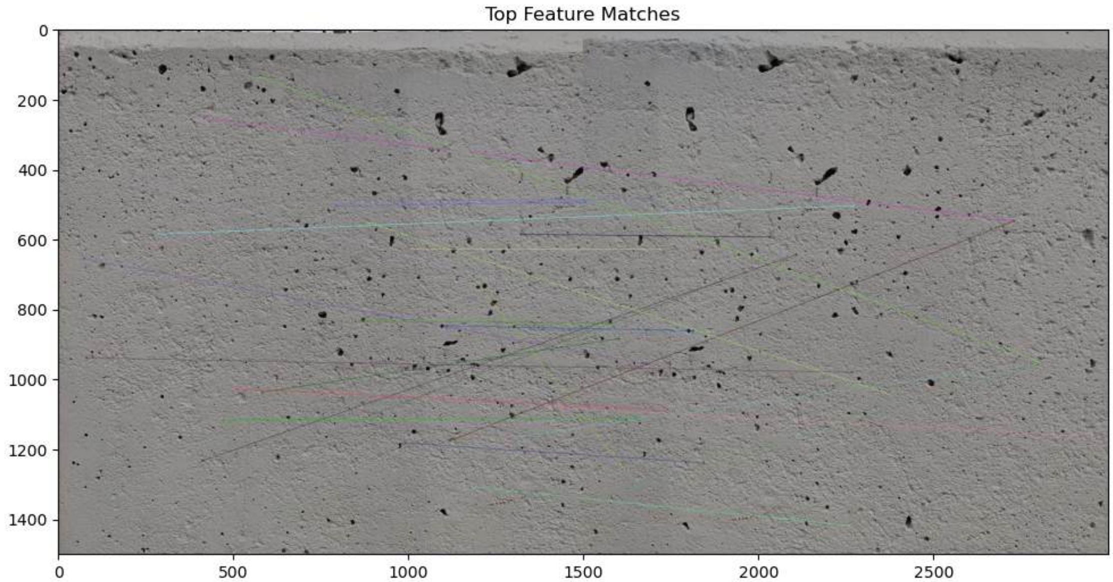
```python
plt.figure(figsize=(30, 20))
plt.imshow(panorama)
plt.title("Panorama Image")
plt.axis("off")
plt.show()
```

Stitching image 0 and 1…

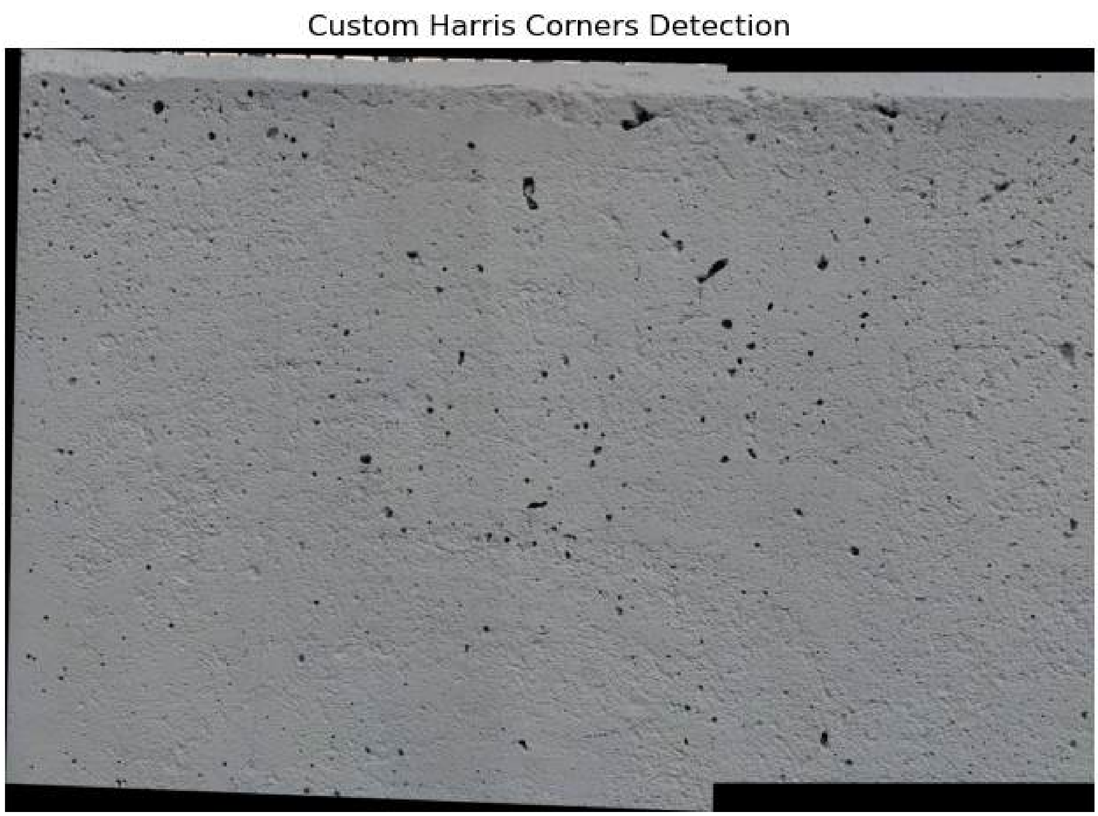


Custom Harris Corners Detection

```
Number of good matches between stitched image and image 1: 27
```

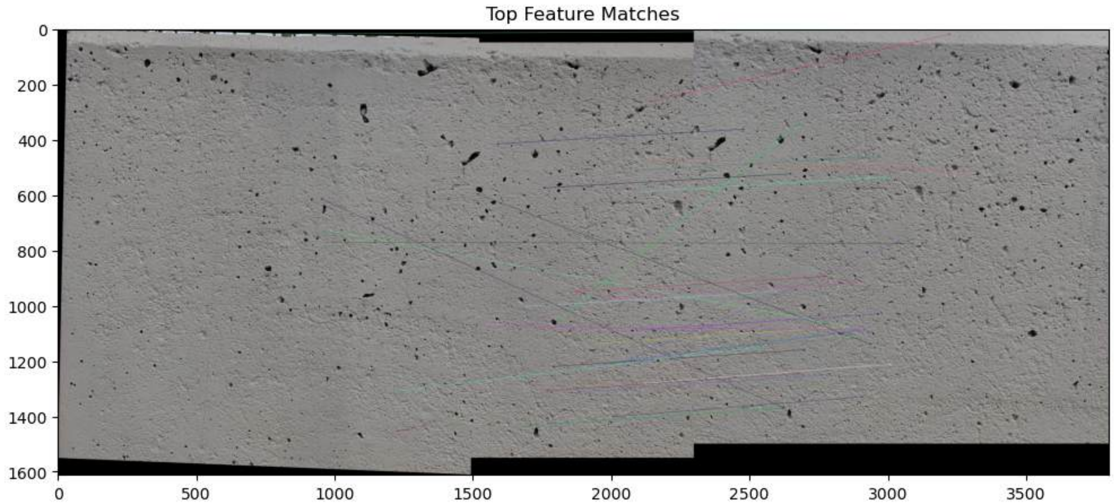


Stitching image 1 and 2…

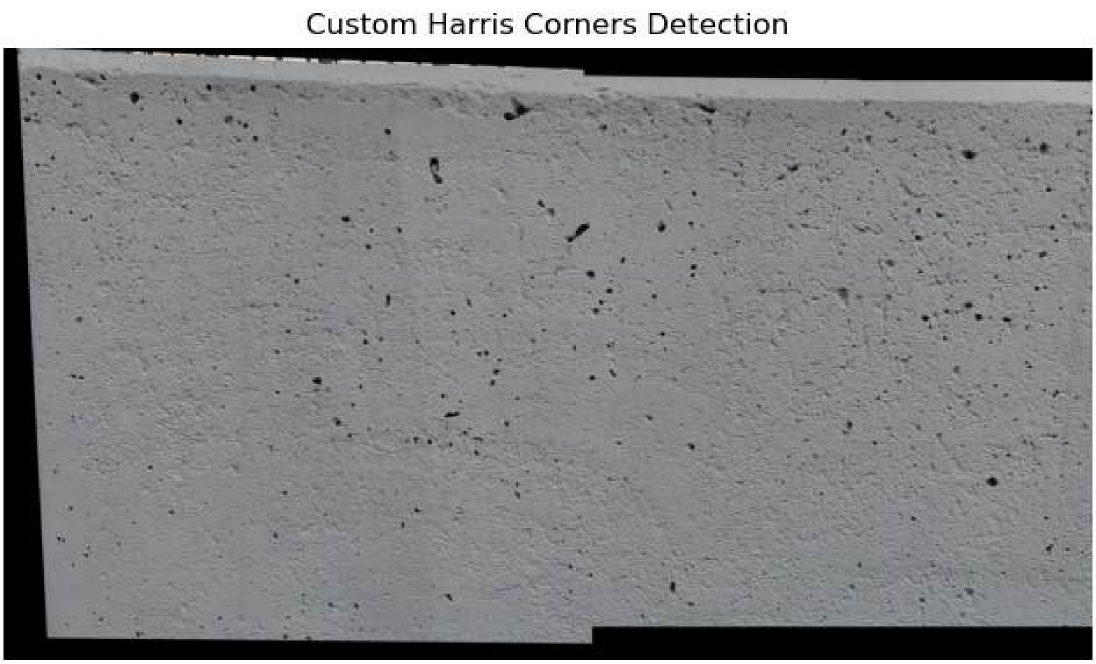


Custom Harris Corners Detection

```
Number of good matches between stitched image and image 2: 31
```

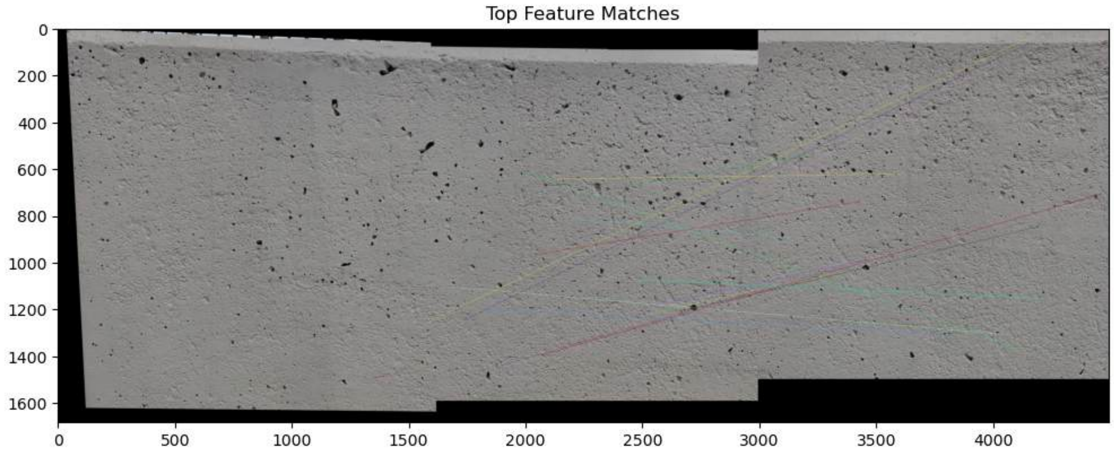


Stitching image 2 and 3…



Custom Harris Corners Detection

```
Number of good matches between stitched image and image 3: 19
```

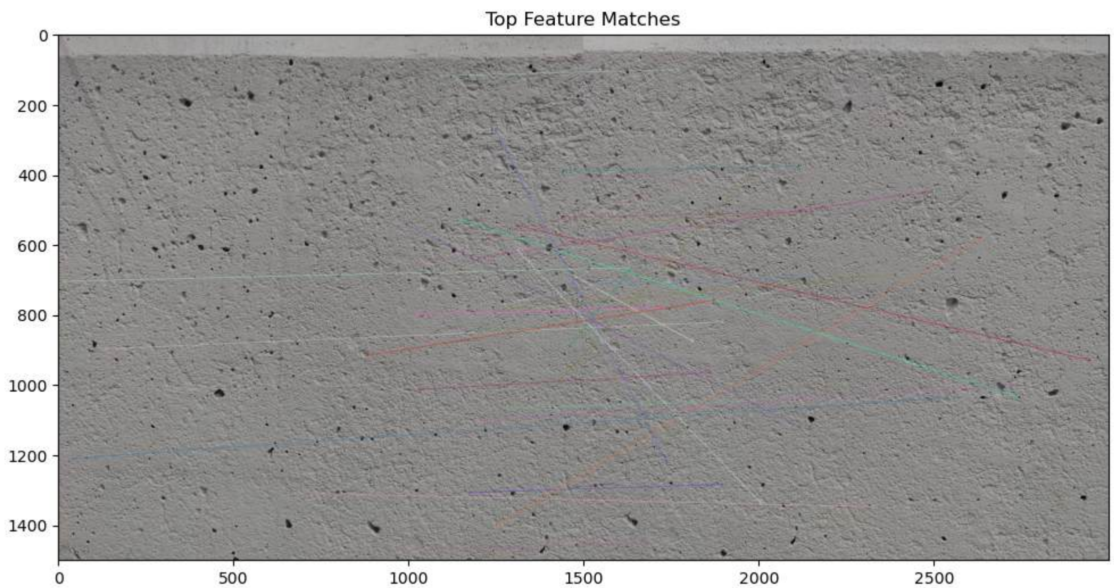



```
Stitching image 3 and 4…
```

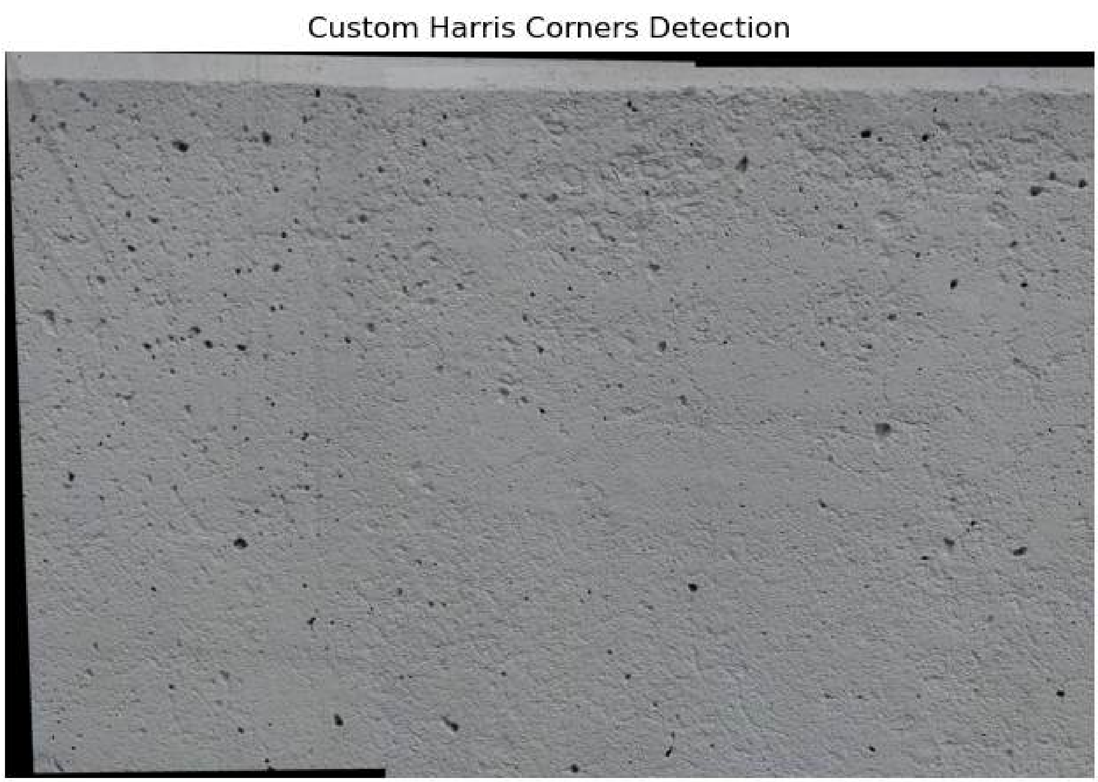
Custom Harris Corners Detection

Custom Harris Corners Detection

Number of good matches between stitched image and image 4: 31

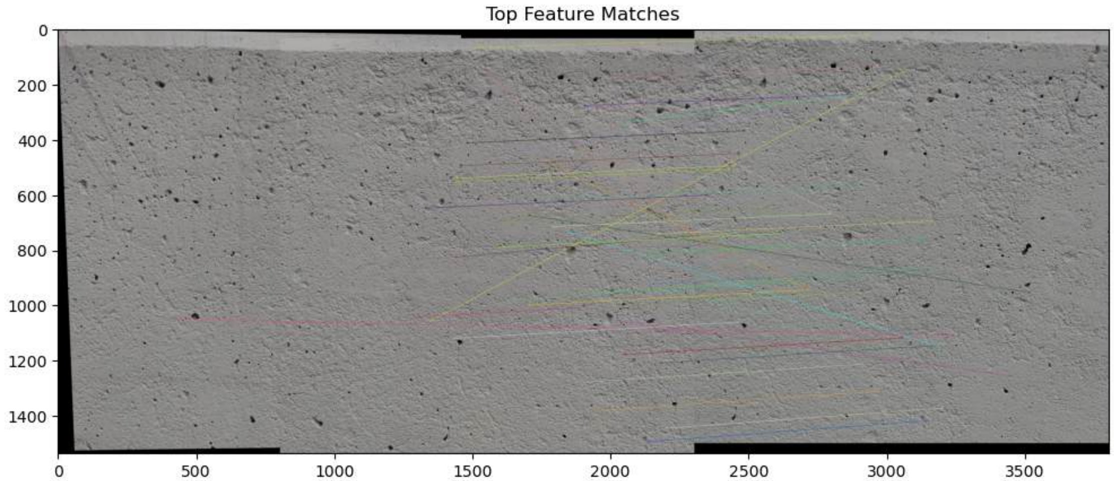


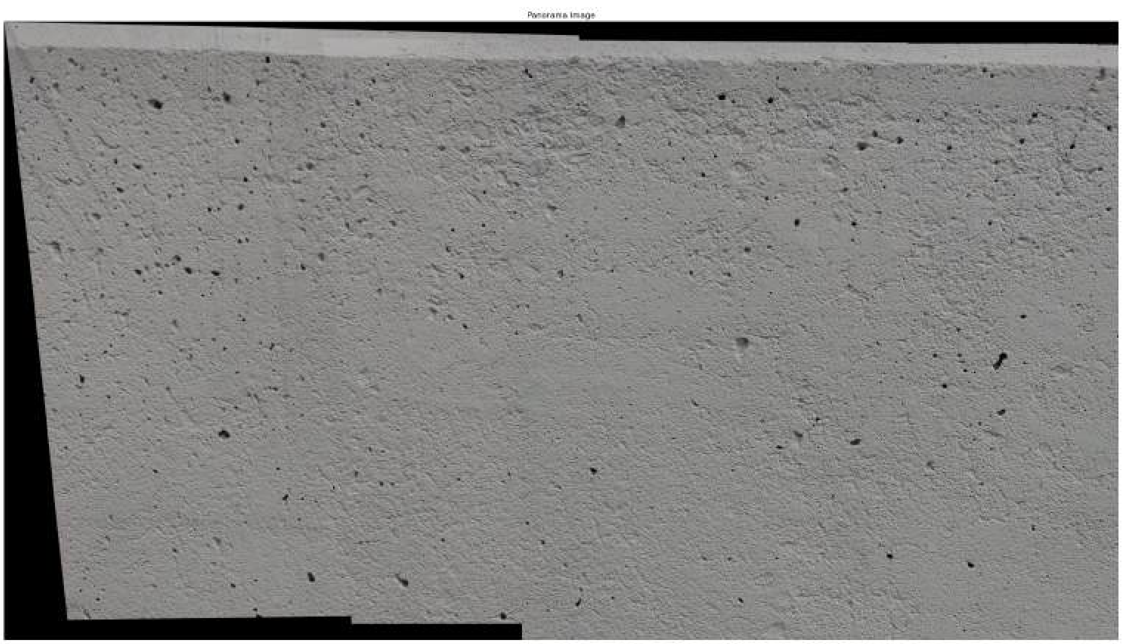
Stitching image 4 and 5…



Custom Harris Corners Detection

Number of good matches between stitched image and image 5: 41





[111]:
```
"""
White cinder blocks present a unique challenge due to their repetitive, uniform␣
↪texture.
Such patterns can make it difficult for traditional feature detection␣
↪algorithms to identify unique keypoints
that stand out distinctly across multiple images.Therefore ,  couple of changes␣
↪has been made to improve the panaroma formation in case of white cinder
block images,
```

```
1. Harris Corner detection value(N) - N has been set to a low value of 250 since lower number of keypoints is suitable
for surfaces with distict and repetitive patterns like cinder blocks.
2. RANSAC Threshold : It has been set to a high value of 8.0 which allows more flexiblity while matching since matching surfaces
with very few distinct features can been difficult and which may lead to alignment issues.
"""
```

[111]:

```
' \nWhite cinder blocks present a unique challenge due to their repetitive,
uniform texture.\nSuch patterns can make it difficult for traditional feature
detection algorithms to identify unique keypoints \nthat stand out distinctly
across multiple images.Therefore ,  couple of changes has been made to improve
the panaroma formation in case of white cinder \nblock images,\n1. Harris Corner
detection value(N) - N has been set to a low value of 250 since lower number of
keypoints is suitable \nfor surfaces with distict and repetitive patterns like
cinder blocks. \n2. RANSAC Threshold : It has been set to a high value of 8.0
which allows more flexiblity while matching since matching surfaces\nwith very
few distinct features can been difficult and which may lead to alignment issues.
\n'
```

[14]:

```python
#Loading second set of images with 50% overlapping
```

[83]:

```python
path = "C:\\Users\\ragsh\\Desktop\\FALL 24\\RSN\\LAB5\\LAB5\\data\\grafiti-50per"
images = load_images(path)
```

```
Images loaded successfully!
```

[85]:

```python
#The detect and compute keypoints function is kept same as before
def detect_and_compute_keypoints(img,show_corners = False,harris_params={}):
    """
    Detect keypoints using Harris Corner Detector and compute descriptors using SIFT.
    """
    # Initialize SIFT and convert image to grayscale
    sift = cv2.SIFT_create()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
    y_coords, x_coords, response = harris(gray, **harris_params)

    keypoints = [cv2.KeyPoint(float(x), float(y), 1) for y, x in zip(y_coords,␣
↪x_coords)]

    # Compute descriptors for detected keypoints using SIFT
    keypoints, descriptors = sift.compute(img, keypoints)

    ## displaying harris corner detection output
    if show_corners:
        img_with_corners = img.copy()
        for y, x in zip(y_coords, x_coords):
            img_with_corners[int(y), int(x)] = [0, 0, 255]  # Mark corners in␣
↪red
        plt.figure(figsize=(8, 8))
        plt.imshow(cv2.cvtColor(img_with_corners, cv2.COLOR_BGR2RGB))
        plt.title("Custom Harris Corners Detection")
        plt.axis("off")
        plt.show()



    return keypoints, descriptors
```

[87]:
```
def stitch_images(img1, img2, kp1, kp2, matches):
    """
    Stitch two images together using matched keypoints and RANSAC to find␣
↪homography.
    This version adds padding based on detected alignment (left or right).
    """
    # Extract matched keypoints
    img1_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1,␣
↪2)
    img2_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1,␣
↪2)

    # Compute homography matrix using RANSAC
    H, mask = cv2.findHomography(img1_pts, img2_pts, cv2.RANSAC, 4.0) #RANSAC␣
↪Value reduced to improve accuracy in homography
    img1_warped, img2_padded = warpPerspectivePadded(img1, img2, H)
    stitched_image = masking(img1_warped, img2_padded, 0.1)
    result = np.uint8(stitched_image)

    return result
```

[89]:
```
def create_panorama(images, show_corners=False, harris_params={}):
    """
    Create a panorama from a list of images.
    """
    stitched_image = images[0]  # Start with the first image
    for i in range(1, len(images)):
        print(f"Stitching image {i-1} and {i}...")

        # Detect keypoints and descriptors with custom Harris and SIFT
        keypoints1, descriptors1 = detect_and_compute_keypoints(stitched_image,␣
↪show_corners=show_corners, harris_params=harris_params)
        keypoints2, descriptors2 = detect_and_compute_keypoints(images[i],␣
↪show_corners=show_corners, harris_params=harris_params)

        # Match features between descriptors
        matches = match_features(descriptors1, descriptors2)
        print(f"Number of good matches between stitched image and image {i}:␣
↪{len(matches)}")

        # Optionally draw matches
        draw_matches(stitched_image, images[i], keypoints1, keypoints2,␣
↪matches, num_matches=50)

        # Stitch the current image with the next in sequence
        stitched_image = stitch_images(stitched_image, images[i], keypoints1,␣
↪keypoints2, matches)


    return stitched_image
```

[91]:
```
#Harris corner parameters for 50% overlap images
harris_params = {

        'disp': False,
        'N': 1500,
        'thresh': 0.01,
        'hsize': 2,
        'sigma': 0.6,
        'eig': True,
        'tile': [1,1],
        'mask': None,
        'fft': False,
    }
panorama = create_panorama(images, show_corners=True,␣
↪harris_params=harris_params)
```

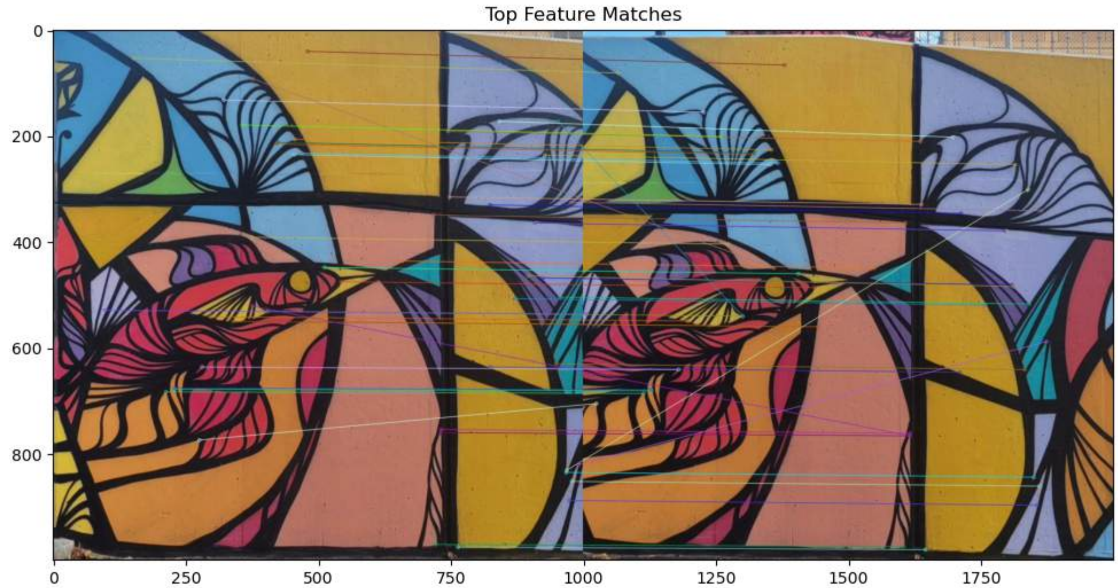
```python
# Display the final panorama
plt.figure(figsize=(50, 30))
plt.imshow(panorama)
plt.title("Panorama Image")
plt.axis("off")
plt.show()
```

Stitching image 0 and 1…

Custom Harris Corners Detection

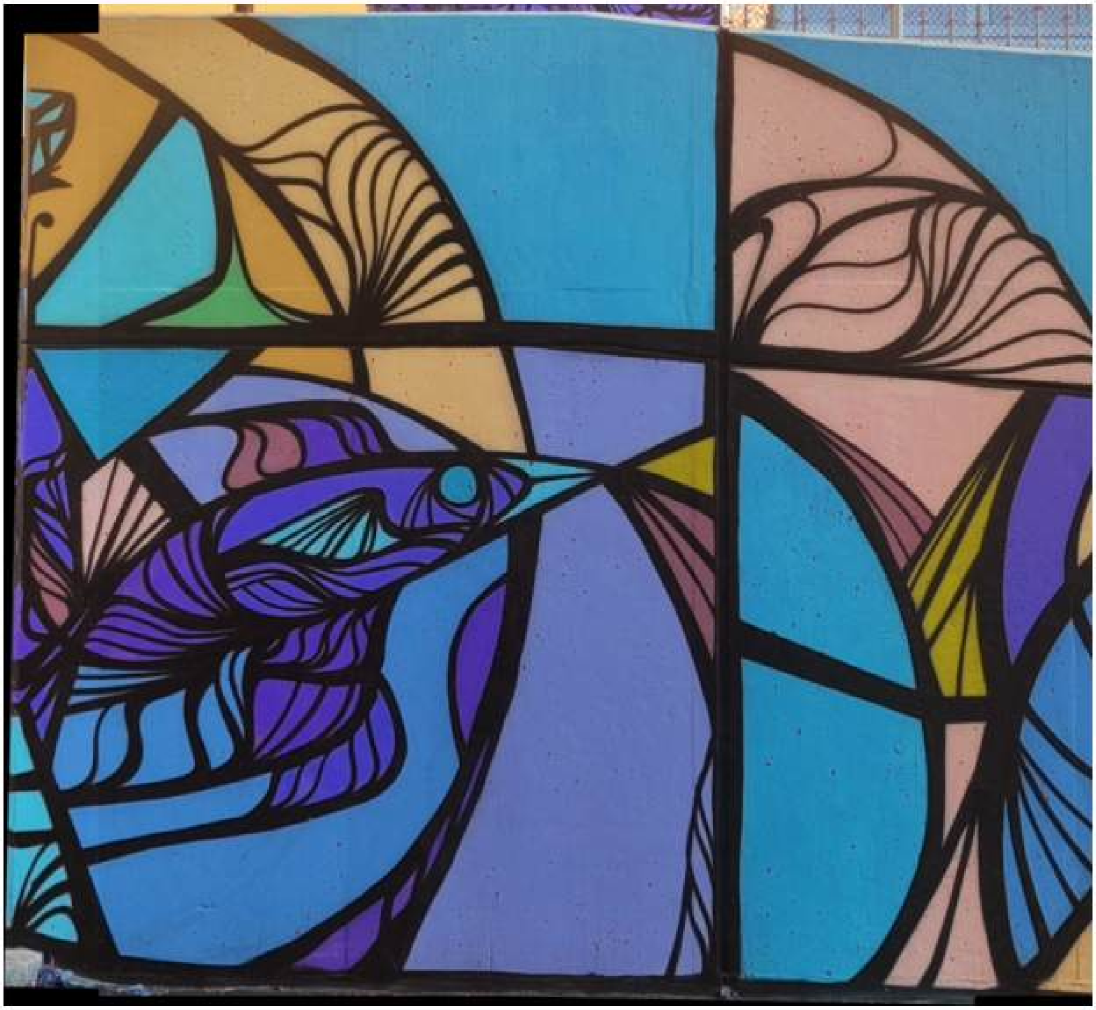
Custom Harris Corners Detection

Number of good matches between stitched image and image 1: 392



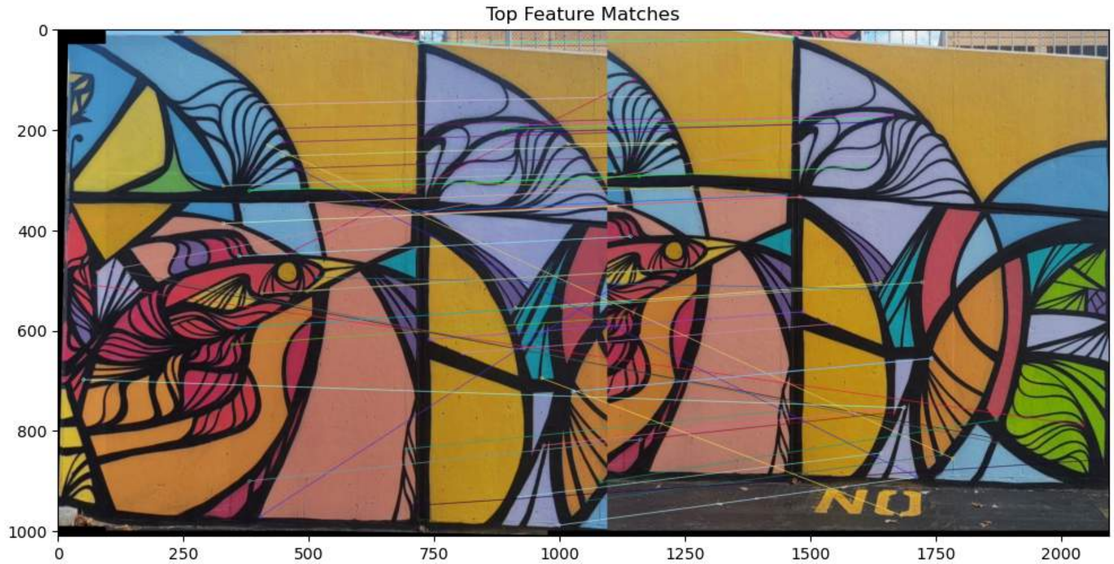
Stitching image 1 and 2…

Custom Harris Corners Detection

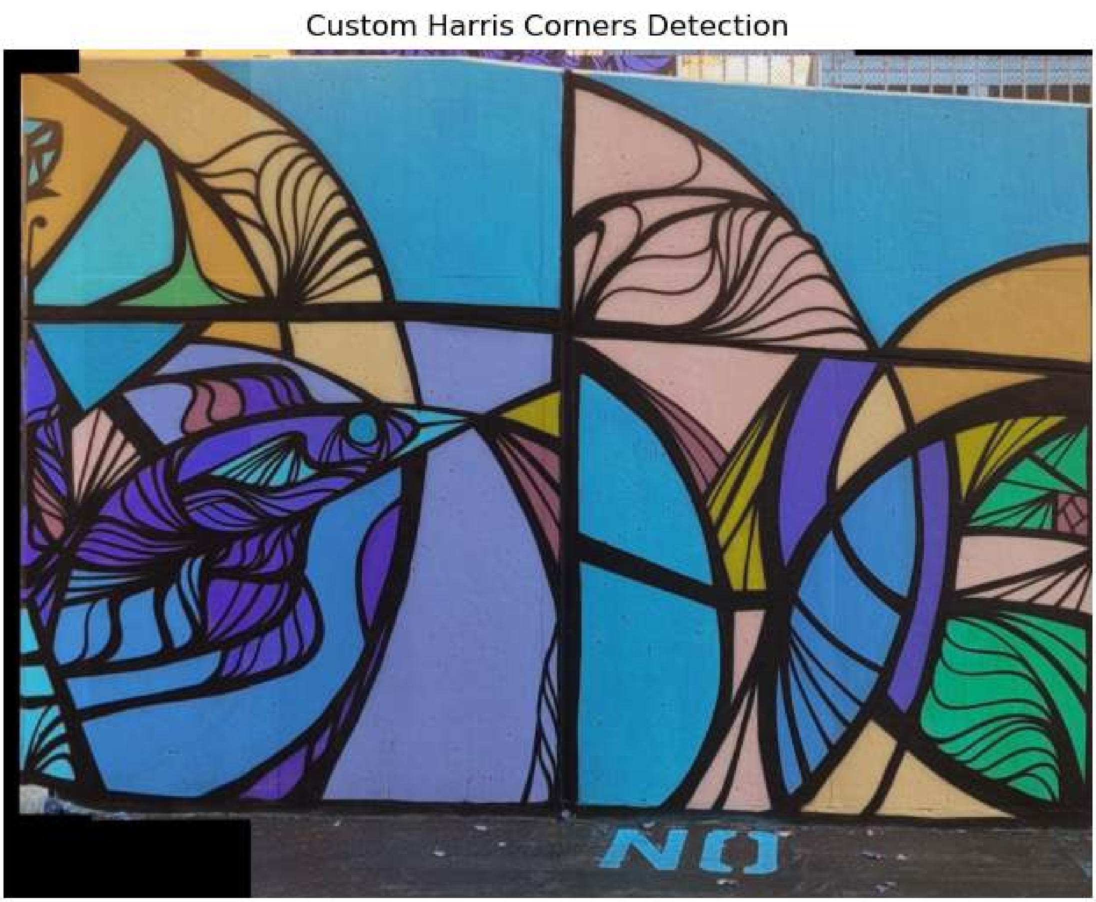


Custom Harris Corners Detection


```
Number of good matches between stitched image and image 2: 308
```

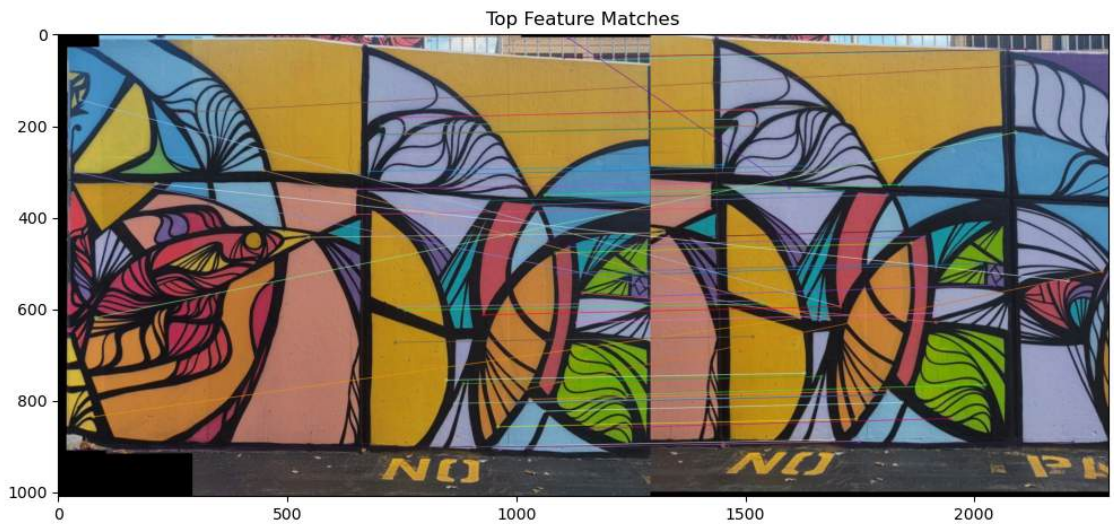


Stitching image 2 and 3…



Custom Harris Corners Detection

```
Number of good matches between stitched image and image 3: 307
```

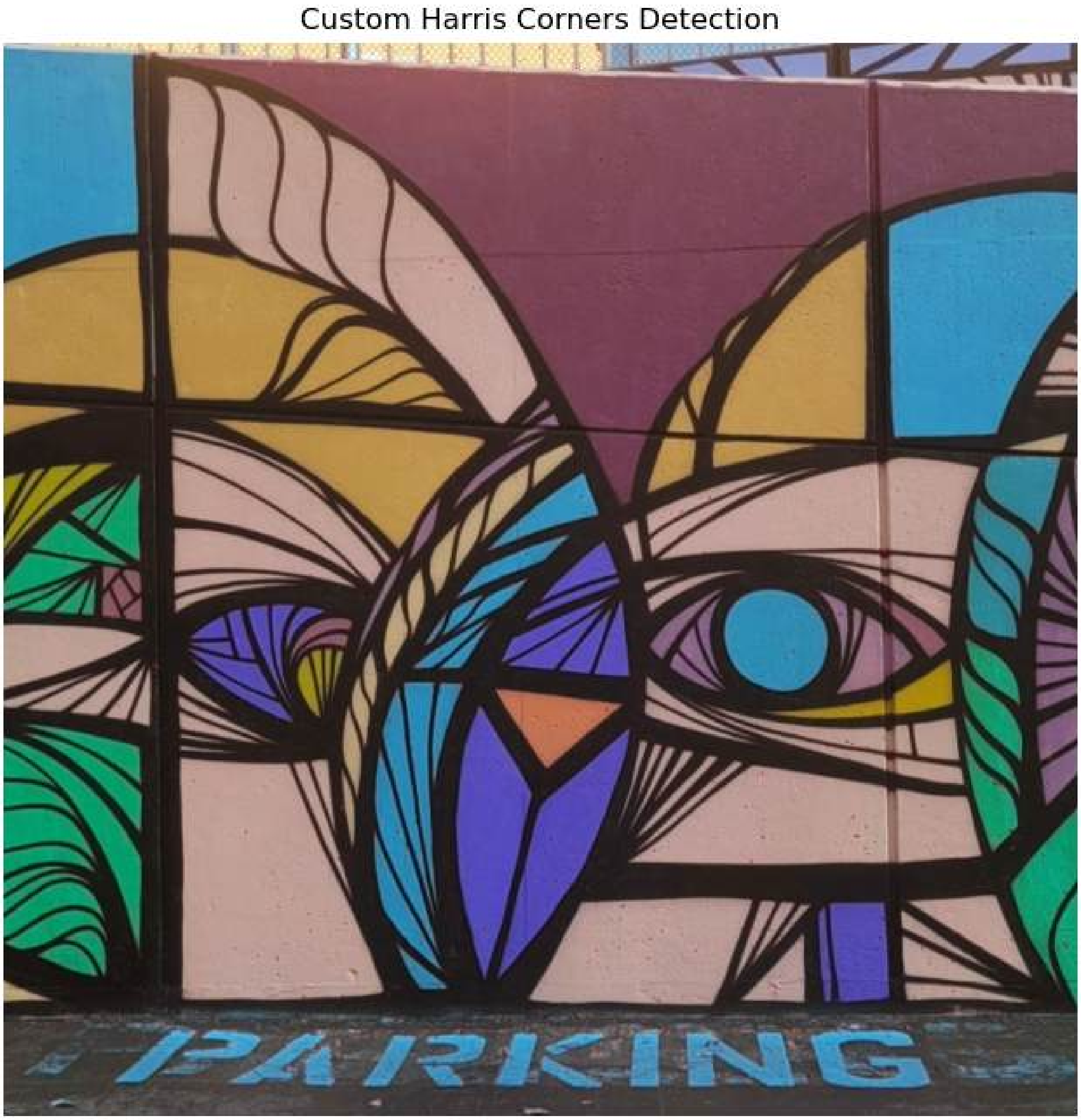


Stitching image 3 and 4…

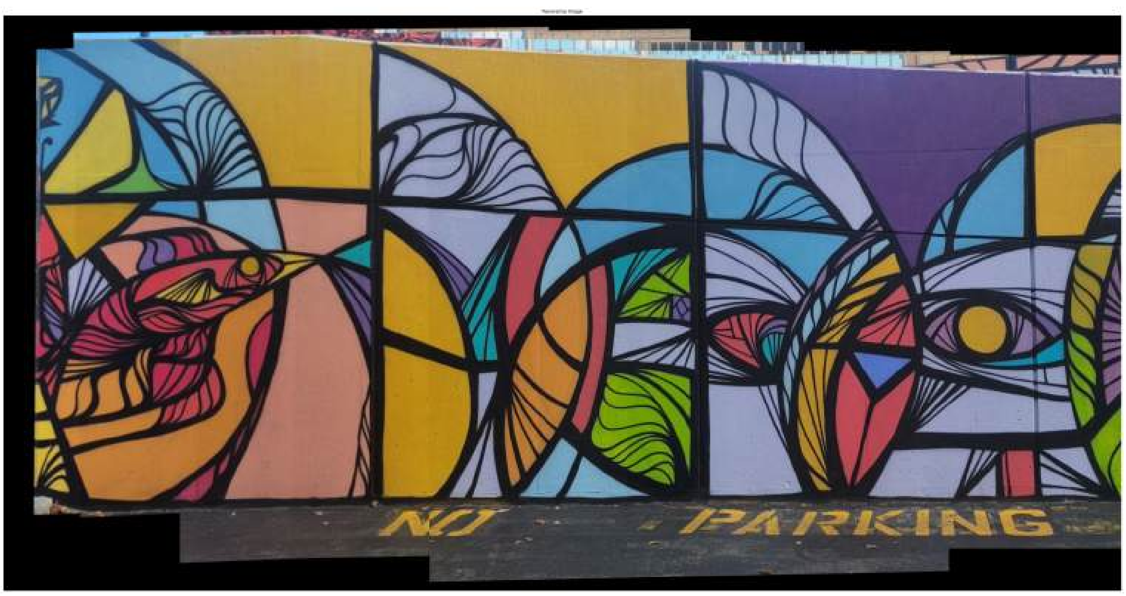
Custom Harris Corners Detection


```
Number of good matches between stitched image and image 4: 288
```

Stitching image 4 and 5…




```
Number of good matches between stitched image and image 5: 302
```



[113]:
```
"""
Images with 50% overlap offer a moderate level of common content between each pair of images.
This overlap provides a good number of shared features for the algorithm to match,
making it feasible to use a larger keypoint count and a more precise homography calculation for stitching.
Therefore ,couple of changes has been made to improve the panaroma formation in case of grafiti image with 50% image
overlapping.
```

```
1. Harris Corner detection value(N) - N has been set to a moderate value of␣
↪1500 allowing greater number of keypoints to be
detected. This keypoint density also ensures that the harris corner algorithm␣
↪will be able to detect more number of features
and thus ensures reliable matches which is crucial for proper stiching of the␣
↪image.
2. RANSAC Threshold : It has been reduced to value of 4.0 which restricts the␣
↪matching criteria further. With the dataset having
a 50% overlapping images, there has been strong matches and this threshold␣
↪values eliminates the noise and reduces alignment
errors,resulting in higher precision stitching.
"""
```

[113]:

```
'\nImages with 50% overlap offer a moderate level of common content between each
pair of images. \nThis overlap provides a good number of shared features for the
algorithm to match, \nmaking it feasible to use a larger keypoint count and a
more precise homography calculation for stitching. \nTherefore ,couple of
changes has been made to improve the panaroma formation in case of grafiti image
with 50% image \noverlapping.\n1. Harris Corner detection value(N) - N has been
set to a moderate value of 1500 allowing greater number of keypoints to be
\ndetected. This keypoint density also ensures that the harris corner algorithm
will be able to detect more number of features\nand thus ensures reliable
matches which is crucial for proper stiching of the image.\n2. RANSAC Threshold
: It has been reduced to value of 4.0 which restricts the matching criteria
further. With the dataset having\na 50% overlapping images, there has been
strong matches and this threshold values eliminates the noise and reduces
alignment\nerrors,resulting in higher precision stitching.\n'
```

[9]:

```
#Loading third dataset of images with less(15%) overlapping
```

[93]:

```
path = "C:\\Users\\ragsh\\Desktop\\FALL␣
↪24\\RSN\\LAB5\\LAB5\\data\\grafiti-15per"
images = load_images(path)
```

Images loaded successfully!

[94]:

```
#The detect and compute keypoints function is kept same as before
def detect_and_compute_keypoints(img,show_corners = False,harris_params={}):
    """
```

```
    Detect keypoints using Harris Corner Detector and compute descriptors using␣
↪SIFT.
    """
    # Initialize SIFT and convert image to grayscale
    sift = cv2.SIFT_create()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


    y_coords, x_coords, response = harris(gray, **harris_params)

    keypoints = [cv2.KeyPoint(float(x), float(y), 1) for y, x in zip(y_coords,␣
↪x_coords)]

    # Compute descriptors for detected keypoints using SIFT
    keypoints, descriptors = sift.compute(img, keypoints)

    ## displaying harris corner detection output
    if show_corners:
        img_with_corners = img.copy()
        for y, x in zip(y_coords, x_coords):
            img_with_corners[int(y), int(x)] = [0, 0, 255]  # Mark corners in␣
↪red
        plt.figure(figsize=(8, 8))
        plt.imshow(cv2.cvtColor(img_with_corners, cv2.COLOR_BGR2RGB))
        plt.title("Custom Harris Corners Detection")
        plt.axis("off")
        plt.show()



    return keypoints, descriptors
```

[97]:

```
def stitch_images(img1, img2, kp1, kp2, matches):
    """
    Stitch two images together using matched keypoints and RANSAC to find␣
↪homography.
    This version adds padding based on detected alignment (left or right).
    """
    # Extract matched keypoints
    img1_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1,␣
↪2)
    img2_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1,␣
↪2)

    # Compute homography matrix using RANSAC
```

```
    H, mask = cv2.findHomography(img1_pts, img2_pts, cv2.RANSAC, 1.0) #RANSAC␣
↪value reduced further to improve accuracy in homography
    img1_warped, img2_padded = warpPerspectivePadded(img1, img2, H)
    stitched_image = masking(img1_warped, img2_padded, 0.1)
    result = np.uint8(stitched_image)

    return result
```

[99]:
```
def create_panorama(images, show_corners=False, harris_params={}):
    """
    Create a panorama from a list of images.
    """
    stitched_image = images[0]  # Start with the first image
    for i in range(1, len(images)):
        print(f"Stitching image {i-1} and {i}...")

        # Detect keypoints and descriptors with custom Harris and SIFT
        keypoints1, descriptors1 = detect_and_compute_keypoints(stitched_image,␣
↪show_corners=show_corners, harris_params=harris_params)
        keypoints2, descriptors2 = detect_and_compute_keypoints(images[i],␣
↪show_corners=show_corners, harris_params=harris_params)

        # Match features between descriptors
        matches = match_features(descriptors1, descriptors2)
        print(f"Number of good matches between stitched image and image {i}:␣
↪{len(matches)}")

        # Optionally draw matches
        draw_matches(stitched_image, images[i], keypoints1, keypoints2,␣
↪matches, num_matches=50)

        # Stitch the current image with the next in sequence
        stitched_image = stitch_images(stitched_image, images[i], keypoints1,␣
↪keypoints2, matches)


    return stitched_image
```

[101]:
```
#Harris corner parameters for 50% overlap images
harris_params = {

        'disp': False,
        'N': 5000,
        'thresh': 0.01,
        'hsize': 2,
        'sigma': 0.6,
```

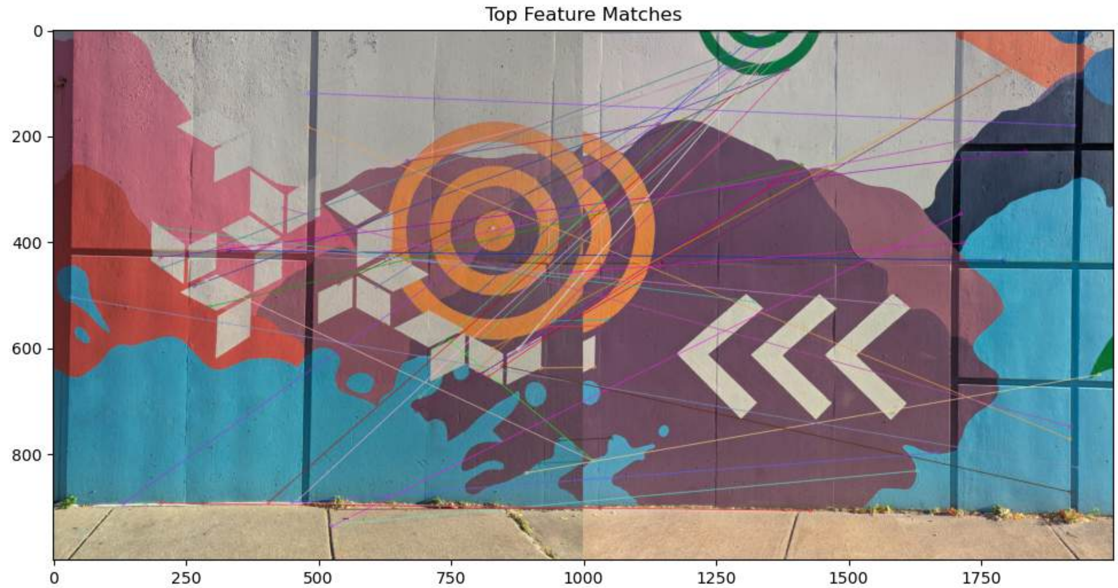
```
        'eig': True,
        'tile': [1,1],
        'mask': None,
        'fft': False,
    }
panorama = create_panorama(images, show_corners=True,␣
 ↪harris_params=harris_params)


# Display the final panorama
plt.figure(figsize=(50,30))
plt.imshow(panorama)
plt.title("Panorama Image")
plt.axis("off")
plt.show()
```

```
Stitching image 0 and 1…
```

Custom Harris Corners Detection

Custom Harris Corners Detection

Number of good matches between stitched image and image 1: 272

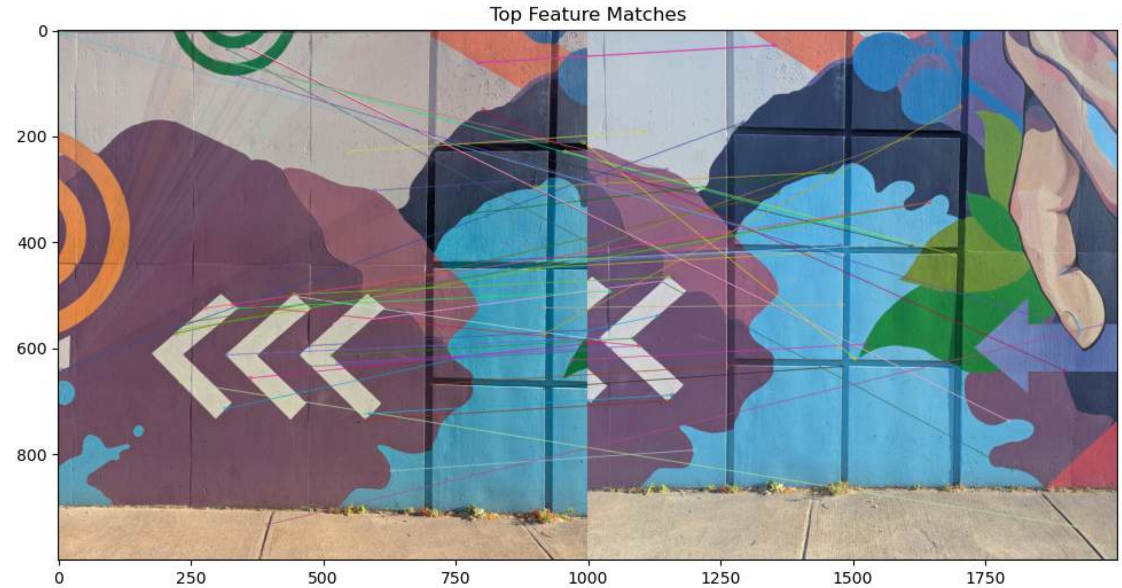


Stitching image 1 and 2…

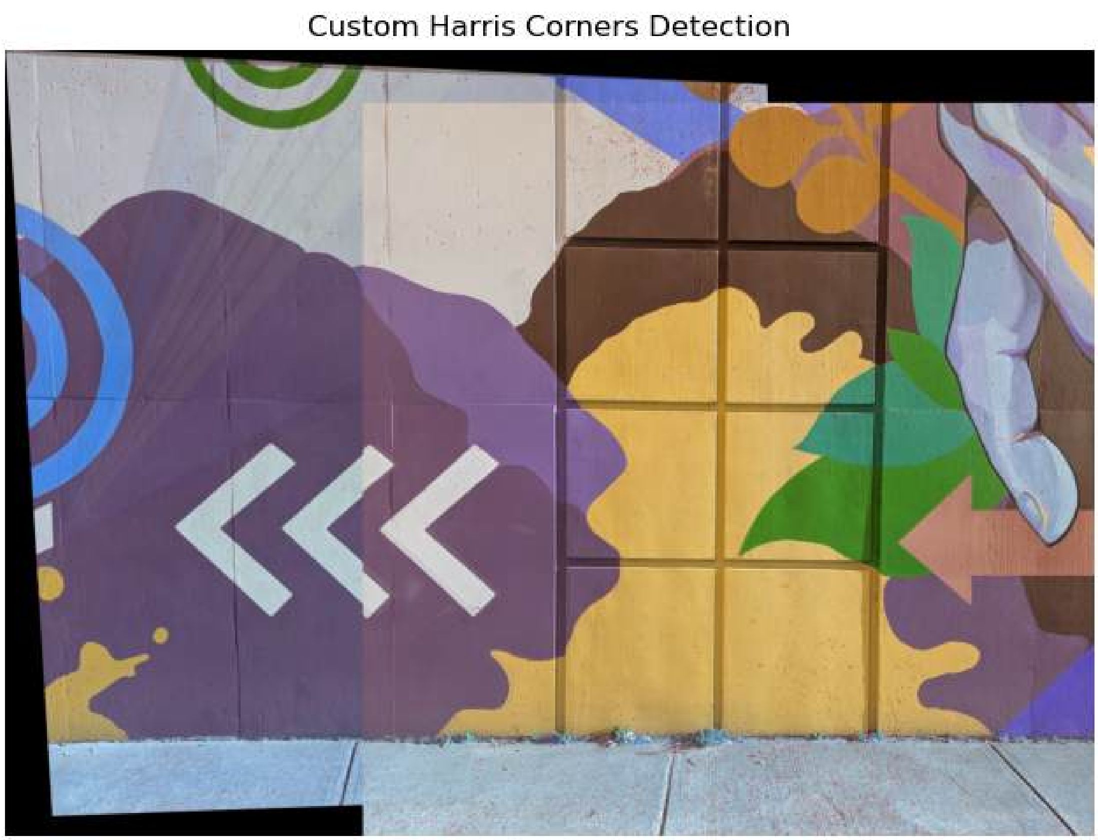
Custom Harris Corners Detection

Custom Harris Corners Detection

```
Number of good matches between stitched image and image 2: 479
```

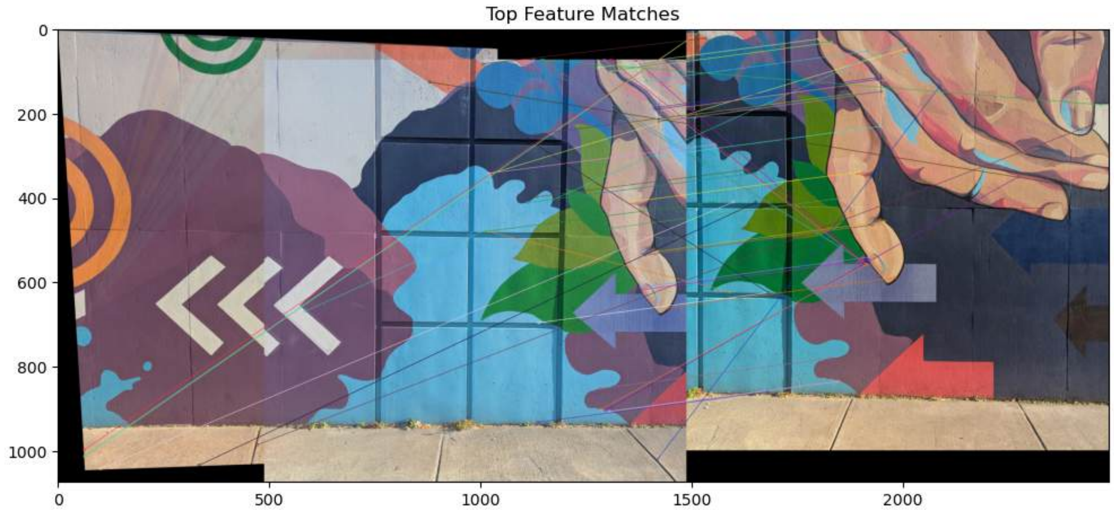


Stitching image 2 and 3…

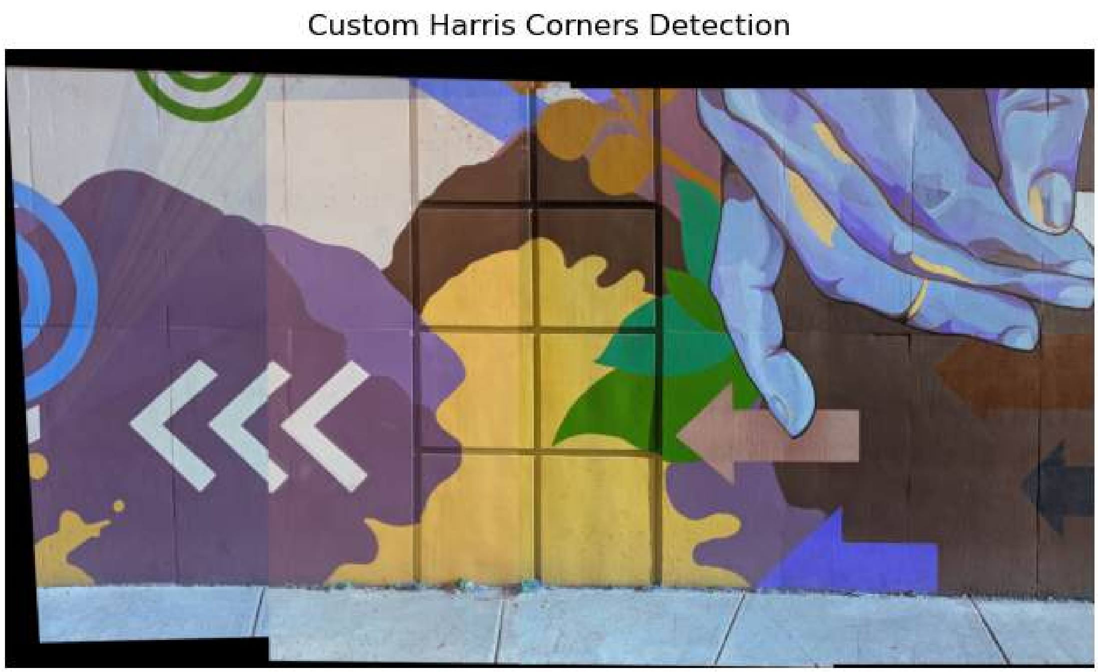


Custom Harris Corners Detection

```
Number of good matches between stitched image and image 3: 389
```

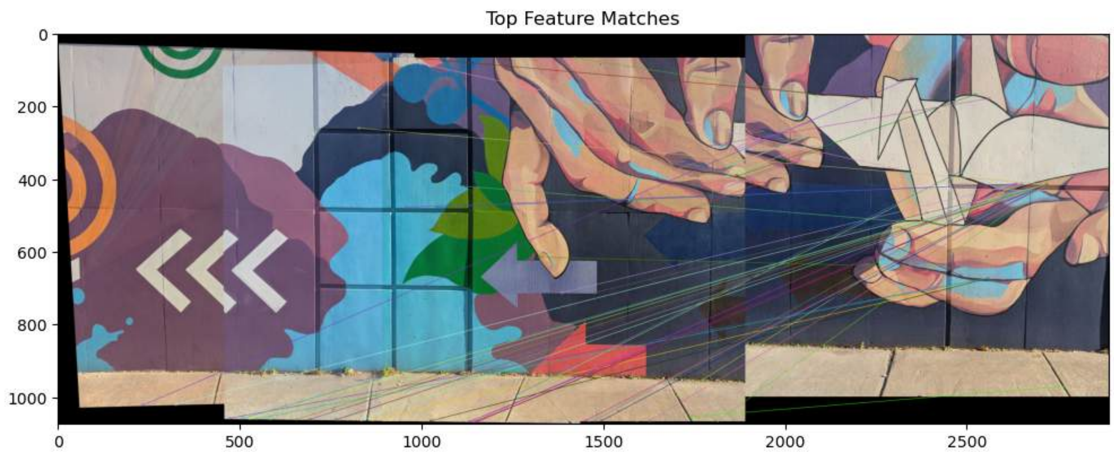


Stitching image 3 and 4…



Custom Harris Corners Detection

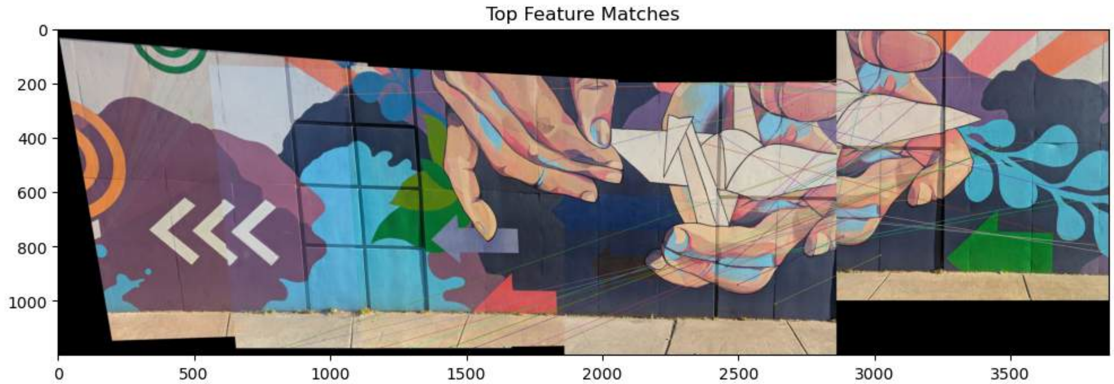
Number of good matches between stitched image and image 4: 394



Stitching image 4 and 5…

Custom Harris Corners Detection

```
Number of good matches between stitched image and image 5: 508
```



[115]:

```
"""
Low-overlap images, with only 15% of shared regions, pose a significant␣
↪challenge for panorama creation
because there are limited common features that are commonly shared between the␣
↪images. Therefore, the parameters have
been futher tuned to improve the image stitching.
1. Harris Corner detection value(N) - N has been set to a high value of 5000␣
↪allowing more number of keypoints to be
detected. This keypoint density also ensures that the harris corner algorithm␣
↪will be able to detect more and more number of features
between the images since they have only 15% in overlap between them.
2. RANSAC Threshold : It has been reduced to a strict value of 1.0 which␣
↪restricts the matching criteria further. With the dataset having
 only 15% overlapping images, A low threshold value filters out weaker matches␣
↪and retains only the accurate ones which is
 essential for proper image panaroma.

"""
```

[115]: '\nLow-overlap images, with only 15% of shared regions, pose a significant challenge for panorama creation \nbecause there are limited common features that are commonly shared between the images. Therefore, the parameters have \nbeen futher tuned to improve the image stitching. \n1. Harris Corner detection value(N) - N has been set to a high value of 5000 allowing more number of keypoints to be \ndetected. This keypoint density also ensures that the harris corner algorithm will be able to detect more and more number of features\nbetween the images since they have only 15% in overlap between them. \n2. RANSAC Threshold : It has been reduced to a strict value of 1.0 which restricts the matching criteria further. With the dataset having\n only 15% overlapping images, A low threshold value filters out weaker matches and retains only the accurate ones which is \n essential for proper image panaroma.\n\n'

[117]:
```
"""
Yes, the mosaicing algorithm continues to work,
but its effectiveness and accuracy vary based on the overlap and type of images␣
 ↪used in each dataset.

The varying overlapping percentages in each dataset required a different "N"␣
 ↪value and RANSAC threshold value to optimize
the stitching process.For cinder blocks, which have repetitive patterns,
fewer keypoints and a higher RANSAC threshold allow for better flexibility. For␣
 ↪50% overlap images,
a balanced approach with a moderate RANSAC threshold and a higher keypoint␣
 ↪count provides accurate matching.
For 15% overlap images, maximizing keypoints and enforcing strict matching are␣
 ↪essential to
identify the limited shared features, ensuring effective alignment despite␣
 ↪minimal overlap.
Each set of parameters is fine-tuned for the dataset characteristics, allowing␣
 ↪for successful
panorama creation across different scenarios.


"""
```

[117]: '\nYes, the mosaicing algorithm continues to work, \nbut its effectiveness and accuracy vary based on the overlap and type of images used in each dataset.\n\nThe varying overlapping percentages in each dataset required a different "N" value and RANSAC threshold value to optimize \nthe stitching process.For cinder blocks, which have repetitive patterns, \nfewer keypoints and a higher RANSAC threshold allow for better flexibility. For 50% overlap images, \na balanced approach with a moderate RANSAC threshold and a higher keypoint count provides accurate matching. \nFor 15% overlap images, maximizing keypoints and enforcing strict matching are essential to \nidentify the limited shared features, ensuring effective alignment despite minimal overlap.\nEach set of parameters is fine-tuned for the dataset characteristics, allowing for

```
successful \npanorama creation across different scenarios.\n\n\n'
```

[ ]: