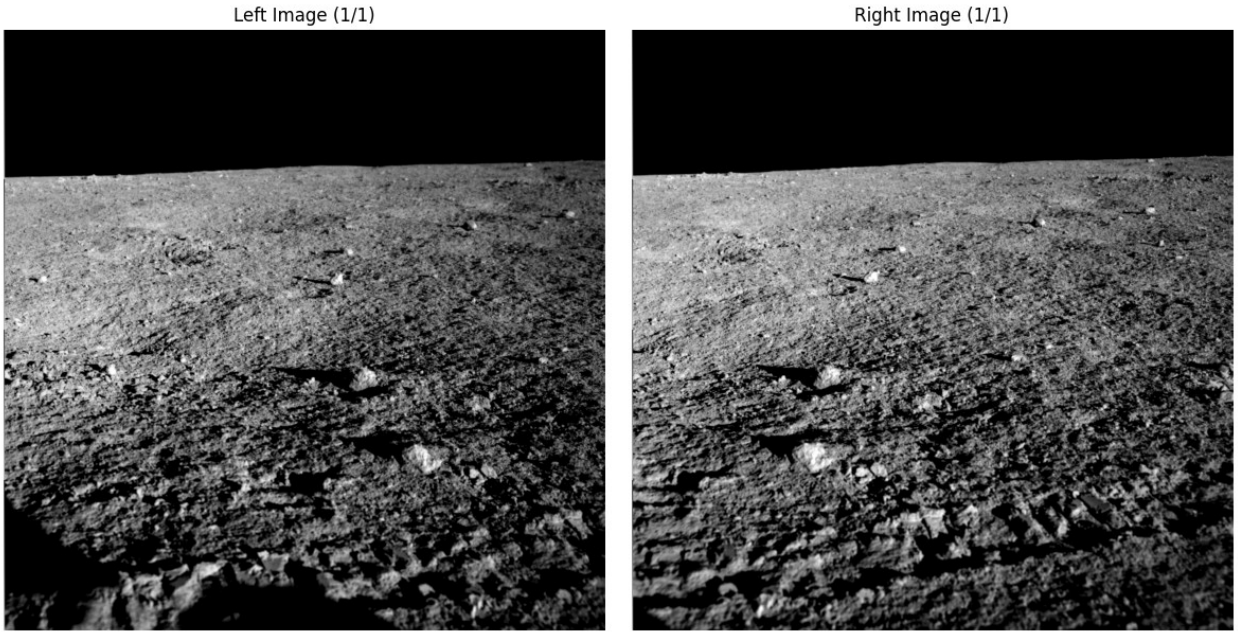
```python
import os
import zipfile
import shutil

# Paths
zip_path = r"D:\ISRO\navcam_2025Jan10T104956978.zip"
extract_to = "extracted_data"
output_folder = "filtered_images"

# Ensure output folder exists
os.makedirs(output_folder, exist_ok=True)

# Extract ZIP file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_to)

# Filter and copy _nrl_ and _nrr_ images
for root, _, files in os.walk(extract_to):
    for file in files:
        if file.endswith('.png') and ('_nrl_' in file or '_nrr_' in
file):
            src = os.path.join(root, file)
            dest = os.path.join(output_folder, file)
            shutil.copy2(src, dest)

print(f"Filtered images have been copied to: {output_folder}")
```

```
Filtered images have been copied to: filtered_images
```

```python
import os
import random
from PIL import Image
import matplotlib.pyplot as plt

def display_random_image_pairs(folder_path, num_pairs=2):
    """
    Displays a specified number of random left-right image pairs.

    Args:
        folder_path (str): Path to the folder containing images.
        num_pairs (int): Number of random pairs to display (default:
2).
    """
    # Collect all _nrl_ (left) and _nrr_ (right) images
    left_images = [file for file in os.listdir(folder_path) if "_nrl_"
in file]
    right_images = [file for file in os.listdir(folder_path) if
"_nrr_" in file]

    # Check if there are left and right images
```

```
    if not left_images or not right_images:
        print("No left or right images found in the folder.")
        return

    # Create mappings for left and right images based on their common
prefix
    left_map = {file.split("_nrl_")[0]: file for file in left_images}
    right_map = {file.split("_nrr_")[0]: file for file in
right_images}

    # Find common prefixes between left and right images
    common_keys = list(left_map.keys() & right_map.keys())

    if not common_keys:
        print("No matching left-right image pairs found.")
        return

    # Limit the number of pairs to display
    num_pairs = min(num_pairs, len(common_keys))

    # Randomly select keys for the pairs
    selected_keys = random.sample(common_keys, num_pairs)

    # Display each selected pair
    for idx, key in enumerate(selected_keys, start=1):
        left_file = os.path.join(folder_path, left_map[key])
        right_file = os.path.join(folder_path, right_map[key])

        # Open images safely
        try:
            left_img = Image.open(left_file).convert("RGB")
            right_img = Image.open(right_file).convert("RGB")
        except Exception as e:
            print(f"Error loading images for key {key}: {e}")
            continue

        # Plot the images
        fig, axs = plt.subplots(1, 2, figsize=(12, 6))

        # Left image
        axs[0].imshow(left_img)
        axs[0].set_title(f"Left Image ({idx}/{num_pairs})")
        axs[0].axis("off")

        # Right image
        axs[1].imshow(right_img)
        axs[1].set_title(f"Right Image ({idx}/{num_pairs})")
        axs[1].axis("off")

        # Display the pair
```

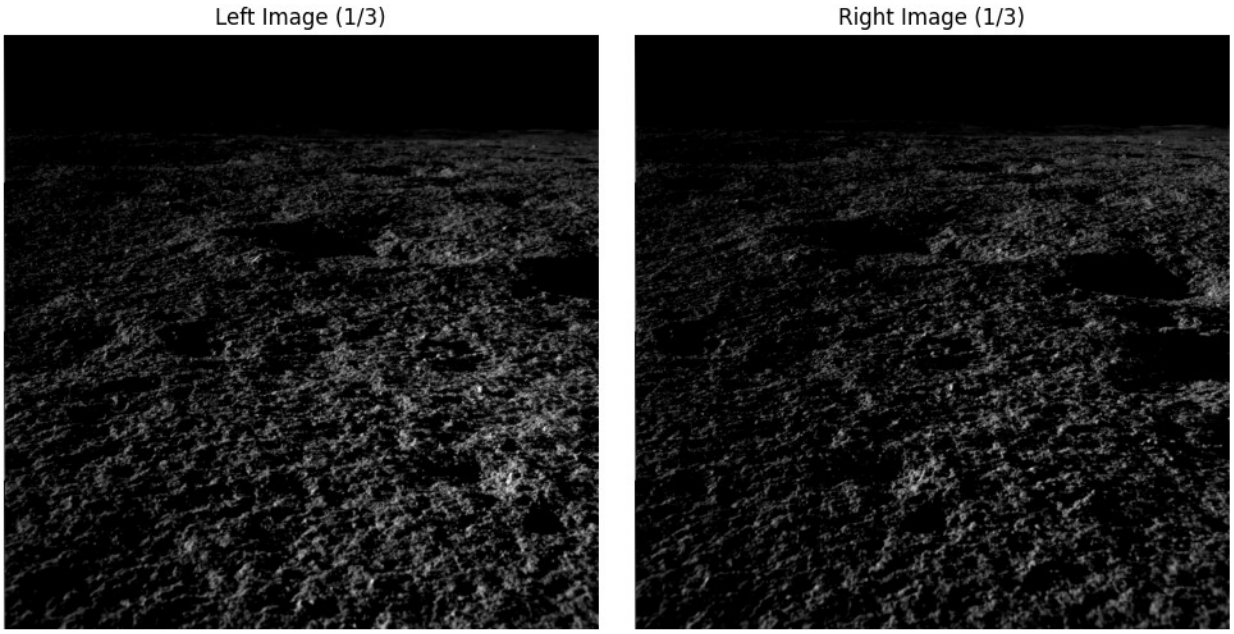
```
        plt.tight_layout()
        plt.show()

# Example Usage
display_random_image_pairs(r"D:\ISRO\filtered_images", num_pairs=3)
```




```
import os
import random
from PIL import Image
import matplotlib.pyplot as plt

def get_suffix(filename):
    """
    Extract the portion starting from '_d_img_' so we can match left
and right images.
    For example:
        ch3_nav_nrl_20230823T1801171121_d_img_d32_001.png
    becomes
        _d_img_d32_001.png
    """
    marker = "_d_img_"
    if marker in filename:
        # Split once on '_d_img_' and keep that plus everything after
        parts = filename.split(marker, 1)
        # parts[0] = everything before '_d_img_', parts[1] =
everything after
        return marker + parts[1]  # e.g. '_d_img_d32_001.png'
    return filename  # fallback if '_d_img_' not found
```

```
def display_random_image_pairs(folder_path, num_pairs=2):
    """
    Displays a specified number of random left-right image pairs by
matching
    on the suffix starting from '_d_img_'.
    """
    # Collect all _nrl_ (left) and _nrr_ (right) images
    left_images = [f for f in os.listdir(folder_path) if "_nrl_" in f]
    right_images = [f for f in os.listdir(folder_path) if "_nrr_" in
f]

    # Check if there are left and right images
    if not left_images or not right_images:
        print("No left or right images found in the folder.")
        return

    # Build dictionaries keyed by the '_d_img_' suffix
    left_map = {}
    for file in left_images:
        key = get_suffix(file)
        left_map[key] = file  # each suffix → last left file with that
suffix

    right_map = {}
    for file in right_images:
        key = get_suffix(file)
        right_map[key] = file  # each suffix → last right file with
that suffix

    # Find common keys (suffixes) between left and right
    common_keys = list(left_map.keys() & right_map.keys())
    if not common_keys:
        print("No matching left-right image pairs found.")
        return

    # Randomly select up to 'num_pairs' unique suffixes
    num_pairs = min(num_pairs, len(common_keys))
    selected_keys = random.sample(common_keys, num_pairs)

    # Display each selected pair
    for idx, key in enumerate(selected_keys, start=1):
        left_file = os.path.join(folder_path, left_map[key])
        right_file = os.path.join(folder_path, right_map[key])

        # Open images safely
        try:
            left_img = Image.open(left_file).convert("RGB")
            right_img = Image.open(right_file).convert("RGB")
        except Exception as e:
            print(f"Error loading images for key {key}: {e}")
```

```
            continue

        # Plot the images side by side
        fig, axs = plt.subplots(1, 2, figsize=(10, 5))

        axs[0].imshow(left_img)
        axs[0].set_title(f"Left Image ({idx}/{num_pairs})")
        axs[0].axis("off")

        axs[1].imshow(right_img)
        axs[1].set_title(f"Right Image ({idx}/{num_pairs})")
        axs[1].axis("off")

        plt.tight_layout()
        plt.show()

# Example usage
display_random_image_pairs(r"D:\ISRO\filtered_images", num_pairs=3)
```



Left Image (2/3)

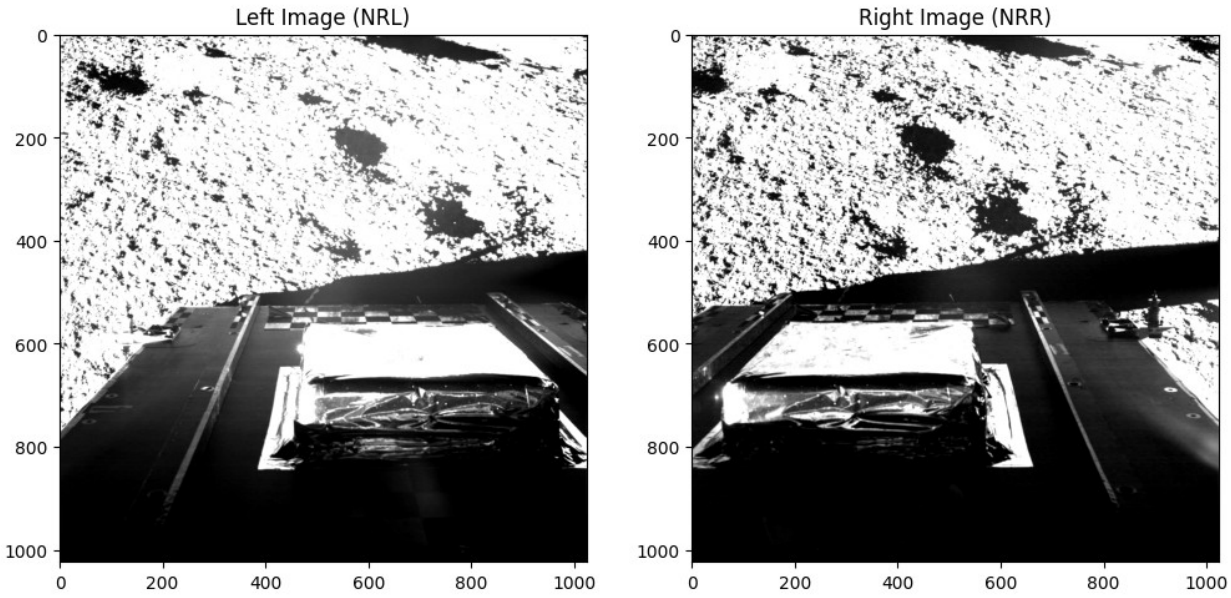
Right Image (2/3)

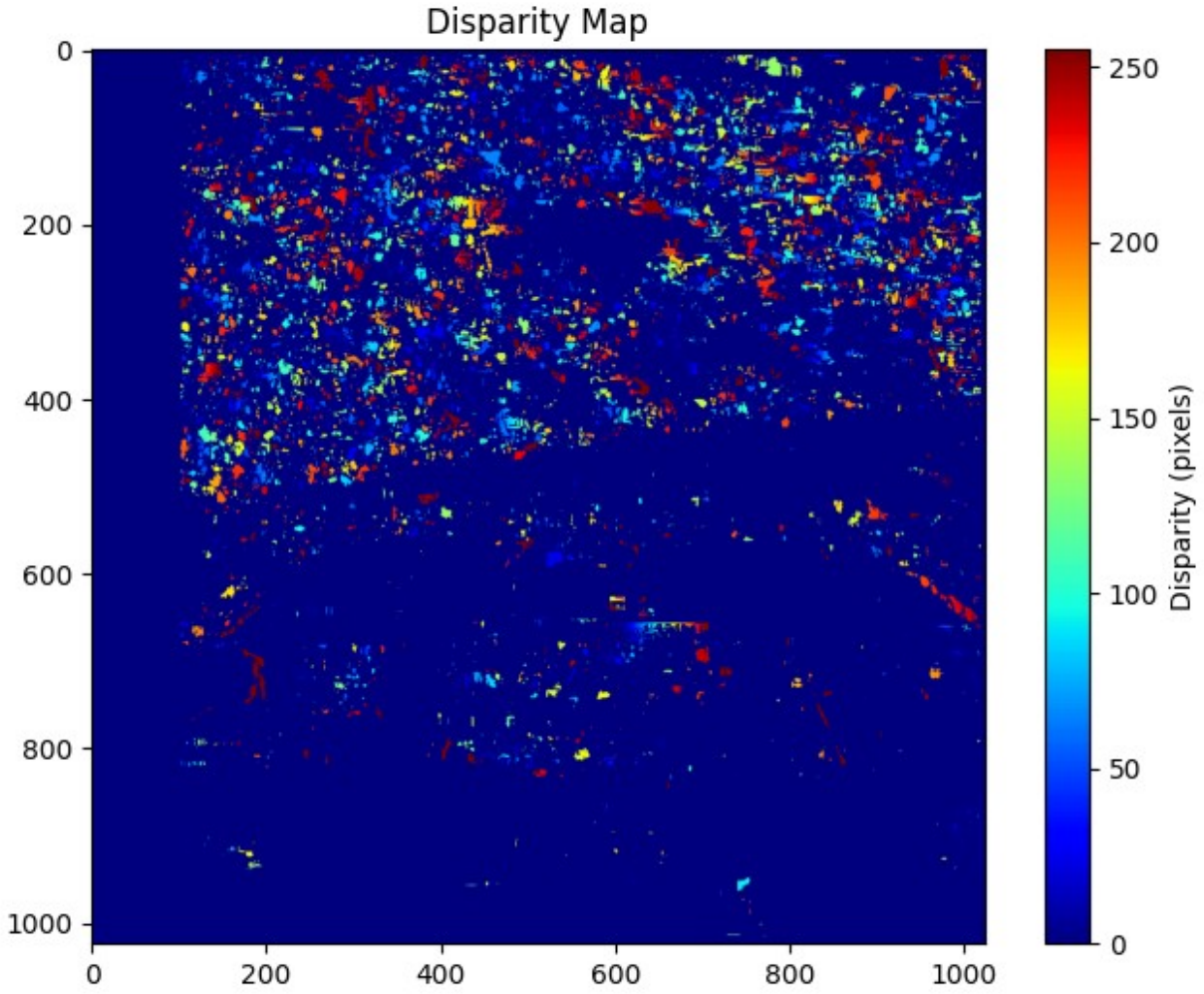
Left Image (3/3)

Right Image (3/3)

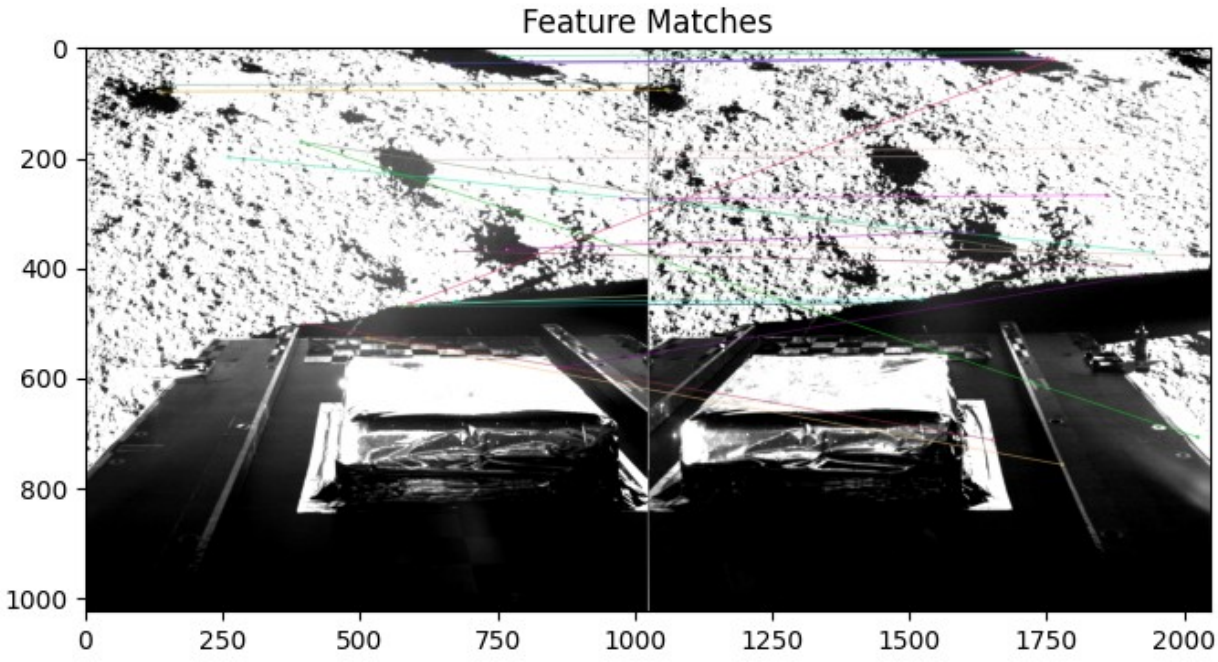
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# File paths for the left-right pair
left_image_path = r"D:\ISRO\filtered_images\
ch3_nav_nrl_20230823T1801171121_d_img_d32_001.png"
right_image_path = r"D:\ISRO\filtered_images\
ch3_nav_nrr_20230823T1801572401_d_img_d32_001.png"
```

```
# Load images as grayscale
left_image = cv2.imread(left_image_path, cv2.IMREAD_GRAYSCALE)
right_image = cv2.imread(right_image_path, cv2.IMREAD_GRAYSCALE)

# Check if images were loaded correctly
if left_image is None or right_image is None:
    raise FileNotFoundError("One or both images could not be loaded.
Check file paths.")

# Resize images to ensure same dimensions
if left_image.shape != right_image.shape:
    height = min(left_image.shape[0], right_image.shape[0])
    width = min(left_image.shape[1], right_image.shape[1])
    left_image = cv2.resize(left_image, (width, height))
    right_image = cv2.resize(right_image, (width, height))

# Ensure images are in uint8 format
left_image = left_image.astype('uint8')
right_image = right_image.astype('uint8')

# Display the images
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1), plt.imshow(left_image, cmap='gray'),
plt.title("Left Image (NRL)")
plt.subplot(1, 2, 2), plt.imshow(right_image, cmap='gray'),
plt.title("Right Image (NRR)")
plt.show()

# Initialize the StereoBM matcher
stereo = cv2.StereoBM_create(numDisparities=16*6, blockSize=15)

# Compute the disparity map
disparity_map = stereo.compute(left_image, right_image)

# Normalize for visualization
disparity_map_normalized = cv2.normalize(disparity_map, None, alpha=0,
beta=255,
                                         norm_type=cv2.NORM_MINMAX,
dtype=cv2.CV_8U)

# Display the disparity map
plt.figure(figsize=(8, 6))
plt.imshow(disparity_map_normalized, cmap='jet')
plt.colorbar(label="Disparity (pixels)")
plt.title("Disparity Map")
plt.show()
```

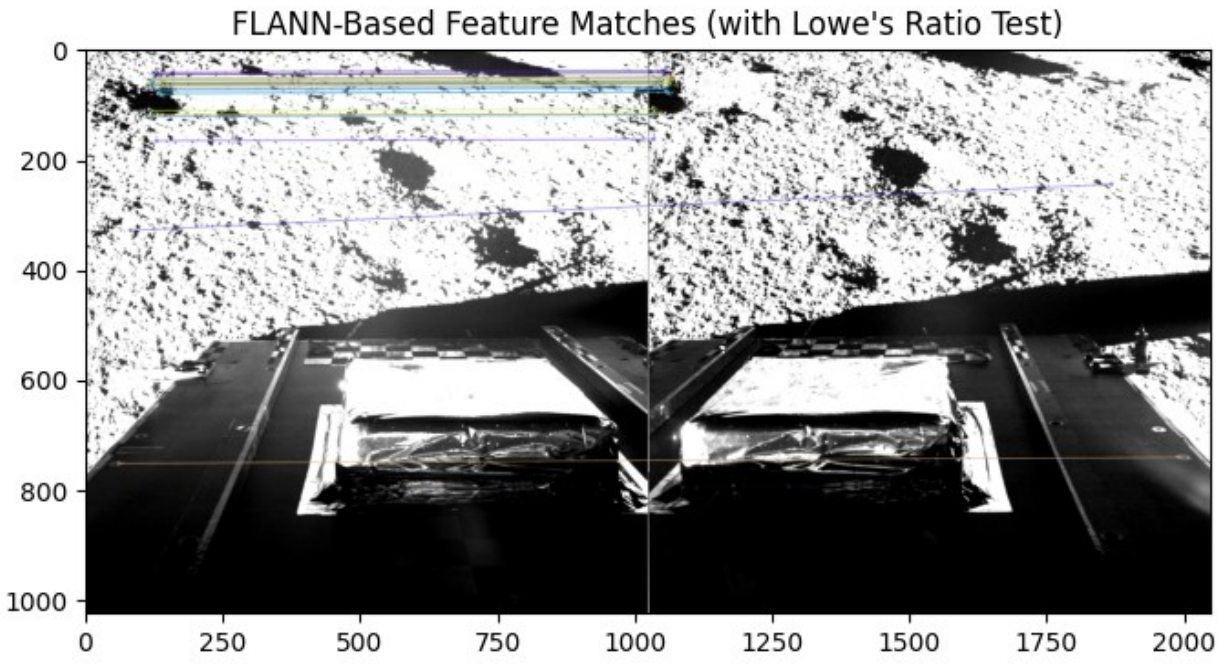

Left Image (NRL)
Right Image (NRR)
0
200
400
600
800
1000


Disparity Map
0
200
400
600
800
1000
250
200
150
100
50
0
Disparity (pixels)

```
# Use SIFT for feature detection and descriptor extraction
sift = cv2.SIFT_create()
keypoints_left, descriptors_left = sift.detectAndCompute(left_image,
None)
keypoints_right, descriptors_right =
sift.detectAndCompute(right_image, None)

# Initialize the Brute-Force matcher with L2 norm (for SIFT)
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

# Match descriptors
matches = bf.match(descriptors_left, descriptors_right)

# Sort matches by distance (best matches first)
matches = sorted(matches, key=lambda x: x.distance)

# Draw the matches
matched_image = cv2.drawMatches(left_image, keypoints_left,
                                right_image, keypoints_right,
                                matches[:25], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

# Display the matched features
plt.figure(figsize=(10,4))
plt.imshow(matched_image)
plt.title("Feature Matches")
plt.show()
```

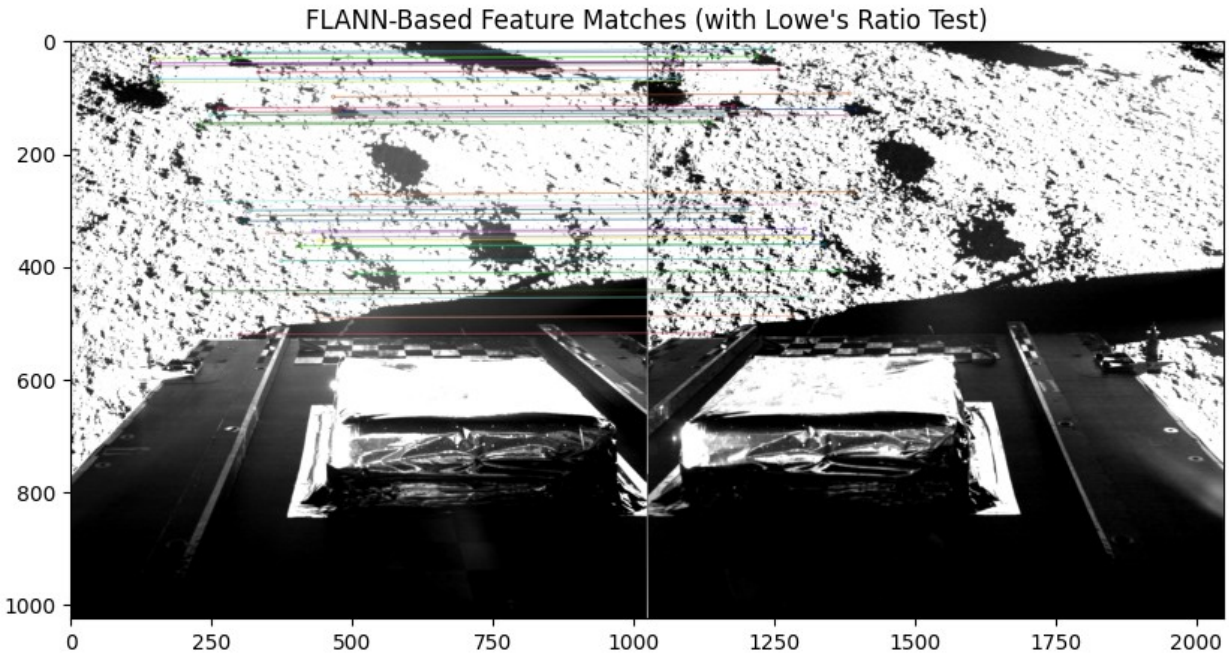


```
# Use SIFT for feature detection and descriptor extraction
sift = cv2.SIFT_create()
keypoints_left, descriptors_left = sift.detectAndCompute(left_image,
None)
keypoints_right, descriptors_right =
sift.detectAndCompute(right_image, None)

# FLANN-based matcher
index_params = dict(algorithm=1, trees=5)  # KDTree for SIFT
search_params = dict(checks=50)  # Number of times to check neighbors
flann = cv2.FlannBasedMatcher(index_params, search_params)

# Match descriptors using KNN
matches = flann.knnMatch(descriptors_left, descriptors_right, k=2)

# Apply Lowe's ratio test
good_matches = []
for m, n in matches:
    if m.distance < 0.6 * n.distance:
        good_matches.append(m)

# Draw the matches
matched_image = cv2.drawMatches(left_image, keypoints_left,
                                right_image, keypoints_right,
                                good_matches[:30], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

# Display the matched features
plt.figure(figsize=(10, 4))
plt.imshow(matched_image)
plt.title("FLANN-Based Feature Matches (with Lowe's Ratio Test)")
plt.show()
```




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

# File paths for the left-right pair
left_image_path = r"D:\ISRO\filtered_images\
ch3_nav_nrl_20230823T1801171121_d_img_d32_001.png"
right_image_path = r"D:\ISRO\filtered_images\
ch3_nav_nrr_20230823T1801572401_d_img_d32_001.png"

# Load images as grayscale
left_image = cv2.imread(left_image_path, cv2.IMREAD_GRAYSCALE)
right_image = cv2.imread(right_image_path, cv2.IMREAD_GRAYSCALE)

# Check if images were loaded correctly
if left_image is None or right_image is None:
    raise FileNotFoundError("One or both images could not be loaded.
Check file paths.")

# Equalize histograms to enhance contrast
left_image_eq = cv2.equalizeHist(left_image)
right_image_eq = cv2.equalizeHist(right_image)

# Sharpen the images
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
left_image_sharp = cv2.filter2D(left_image_eq, -1, kernel)
right_image_sharp = cv2.filter2D(right_image_eq, -1, kernel)

# Use SIFT for feature detection and descriptor extraction
```

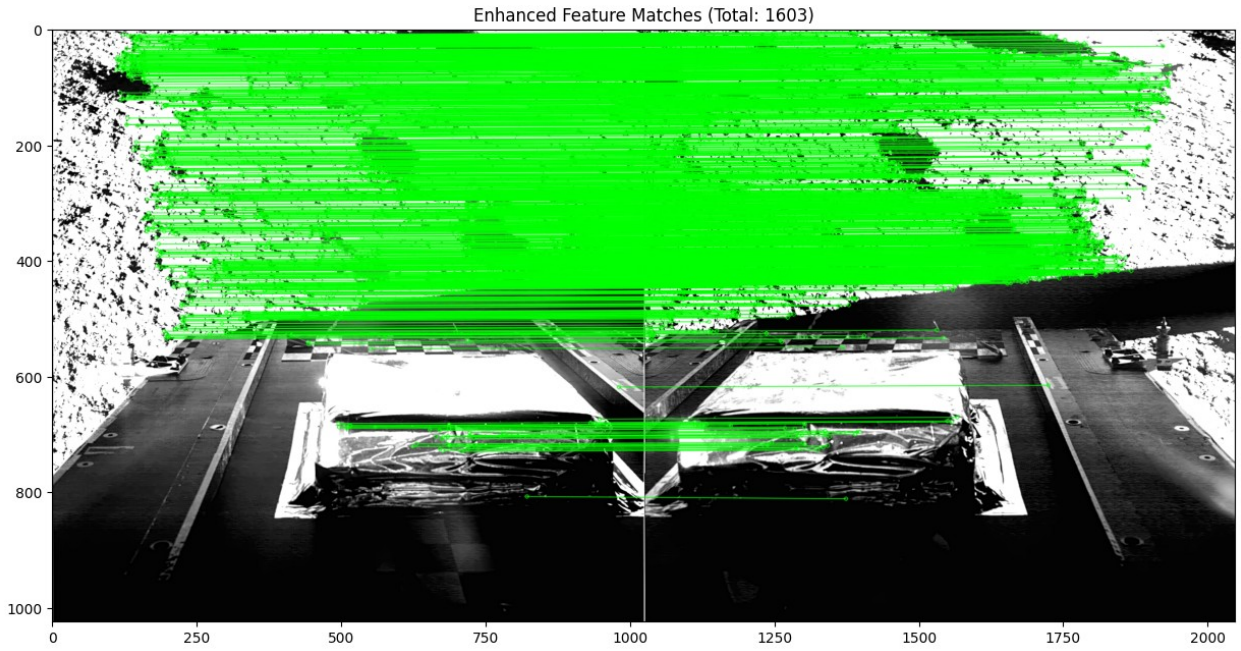
```
sift = cv2.SIFT_create()
keypoints_left, descriptors_left =
sift.detectAndCompute(left_image_sharp, None)
keypoints_right, descriptors_right =
sift.detectAndCompute(right_image_sharp, None)


# FLANN-based matcher
index_params = dict(algorithm=1, trees=5)  # KDTree for SIFT
search_params = dict(checks=50)  # Number of checks
flann = cv2.FlannBasedMatcher(index_params, search_params)

# Match descriptors using KNN
matches = flann.knnMatch(descriptors_left, descriptors_right, k=2)

# Apply Lowe's ratio test
good_matches = []
for m, n in matches:
    if m.distance < 0.5 * n.distance:
        good_matches.append(m)

# Draw the matches
matched_image = cv2.drawMatches(left_image, keypoints_left,
                                right_image, keypoints_right,
                                good_matches[:40], None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

# Display the matched features
plt.figure(figsize=(10,5))
plt.imshow(matched_image)
plt.title("FLANN-Based Feature Matches (with Lowe's Ratio Test)")
plt.show()
```




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

def preprocess_images(left_img, right_img):
    # Convert to float and normalize
    left_float = cv2.normalize(left_img.astype('float32'), None, 0.0,
1.0, cv2.NORM_MINMAX)
    right_float = cv2.normalize(right_img.astype('float32'), None,
0.0, 1.0, cv2.NORM_MINMAX)

    # Apply CLAHE (Contrast Limited Adaptive Histogram Equalization)
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
    left_clahe = clahe.apply(cv2.convertScaleAbs(left_float*255))
    right_clahe = clahe.apply(cv2.convertScaleAbs(right_float*255))

    # Denoise
    left_denoised = cv2.fastNlMeansDenoising(left_clahe)
    right_denoised = cv2.fastNlMeansDenoising(right_clahe)

    # Apply unsharp masking for edge enhancement
    gaussian = cv2.GaussianBlur(left_denoised, (0, 0), 2.0)
    left_enhanced = cv2.addWeighted(left_denoised, 1.5, gaussian, -
0.5, 0)
    gaussian = cv2.GaussianBlur(right_denoised, (0, 0), 2.0)
    right_enhanced = cv2.addWeighted(right_denoised, 1.5, gaussian, -
0.5, 0)

    return left_enhanced, right_enhanced
```

```
def match_features(left_img, right_img):
    # Create SIFT detector with adjusted parameters
    sift = cv2.SIFT_create(
        nfeatures=0,  # no limit on number of features
        nOctaveLayers=5,  # increase number of scale layers
        contrastThreshold=0.04,  # lower threshold to detect more
features
        edgeThreshold=10,  # increase edge threshold
        sigma=1.6
    )

    # Detect keypoints and compute descriptors
    kp1, des1 = sift.detectAndCompute(left_img, None)
    kp2, des2 = sift.detectAndCompute(right_img, None)

    # Configure FLANN matcher
    index_params = {
        'algorithm': 1,  # KDTREE
        'trees': 8       # number of parallel kd-trees
    }
    search_params = {
        'checks': 100    # increase number of searches
    }
    flann = cv2.FlannBasedMatcher(index_params, search_params)

    # Perform matching with k=2
    matches = flann.knnMatch(des1, des2, k=2)

    # Apply ratio test with adjusted threshold
    good_matches = []
    for m, n in matches:
        # Adjust ratio threshold (0.75-0.8 is typical range)
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)

    # Filter matches based on epipolar constraints
    if len(good_matches) > 8:
        src_pts = np.float32([kp1[m.queryIdx].pt for m in
good_matches]).reshape(-1, 1, 2)
        dst_pts = np.float32([kp2[m.trainIdx].pt for m in
good_matches]).reshape(-1, 1, 2)

        # Use RANSAC to find fundamental matrix and filter outliers
        F, mask = cv2.findFundamentalMat(src_pts, dst_pts,
cv2.FM_RANSAC, 3, 0.99)

        # Keep only inlier matches
        good_matches = [good_matches[i] for i in
```

```
range(len(good_matches)) if mask[i]]

    return kp1, kp2, good_matches

# Main execution
left_image = cv2.imread(left_image_path, cv2.IMREAD_GRAYSCALE)
right_image = cv2.imread(right_image_path, cv2.IMREAD_GRAYSCALE)

# Preprocess images
left_processed, right_processed = preprocess_images(left_image,
right_image)

# Match features
keypoints_left, keypoints_right, good_matches =
match_features(left_processed, right_processed)

# Draw matches with more options
matched_image = cv2.drawMatches(left_processed, keypoints_left,
                                right_processed, keypoints_right,
                                good_matches, None,

flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS,
                                matchColor=(0, 255, 0),
                                singlePointColor=(255, 0, 0))

plt.figure(figsize=(15,8))
plt.imshow(matched_image)
plt.title(f"Enhanced Feature Matches (Total: {len(good_matches)})")
plt.show()

# Print statistics
print(f"Number of keypoints in left image: {len(keypoints_left)}")
print(f"Number of keypoints in right image: {len(keypoints_right)}")
print(f"Number of good matches: {len(good_matches)}")
```




```
Number of keypoints in left image: 11939
Number of keypoints in right image: 13421
Number of good matches: 1603

def visualize_matches(left_img, right_img, kp1, kp2, good_matches):
    # Get dimensions for image stitching
    h1, w1 = left_img.shape
    h2, w2 = right_img.shape

    # Create color visualization
    vis = np.zeros((max(h1, h2), w1 + w2, 3), np.uint8)
    vis[:h1, :w1] = cv2.cvtColor(left_img, cv2.COLOR_GRAY2BGR)
    vis[:h2, w1:w1+w2] = cv2.cvtColor(right_img, cv2.COLOR_GRAY2BGR)

    # Calculate disparities and distances
    disparities = []
    pts1 = []
    pts2 = []

    for match in good_matches:
        pt1 = kp1[match.queryIdx].pt
        pt2 = kp2[match.trainIdx].pt
        pts1.append(pt1)
        pts2.append((pt2[0] + w1, pt2[1]))  # Adjust x-coordinate for
right image
        disparity = abs(pt1[0] - pt2[0])
        disparities.append(disparity)

    # Normalize disparities to 0-1 range
    if disparities:
```

```
        min_disp = min(disparities)
        max_disp = max(disparities)
        norm_disparities = [(d - min_disp)/(max_disp - min_disp) if
max_disp != min_disp else 0.5 for d in disparities]

        # Draw matches with color coding
        for i in range(len(pts1)):
            # Color interpolation: blue (small disparity) to red
(large disparity)
            color = (int(255*norm_disparities[i]), 0, int(255*(1-
norm_disparities[i])))
            cv2.line(vis, (int(pts1[i][0]), int(pts1[i][1])),
                     (int(pts2[i][0]), int(pts2[i][1])), color, 1)
            cv2.circle(vis, (int(pts1[i][0]), int(pts1[i][1])), 3,
color, -1)
            cv2.circle(vis, (int(pts2[i][0]), int(pts2[i][1])), 3,
color, -1)

        # Add color bar
        cbar_height = 30
        cbar_width = w1 + w2
        cbar = np.zeros((cbar_height, cbar_width, 3), np.uint8)
        for x in range(cbar_width):
            norm_x = x / cbar_width
            color = (int(255*norm_x), 0, int(255*(1-norm_x)))
            cv2.line(cbar, (x, 0), (x, cbar_height), color, 1)

        # Add text annotations for disparity range
        vis = np.vstack([vis, cbar])
        cv2.putText(vis, f'Min disparity: {min_disp:.1f}px', (10, h1 +
20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
        cv2.putText(vis, f'Max disparity: {max_disp:.1f}px', (w1 + w2
- 200, h1 + 20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

        # Calculate and display statistics
        mean_disp = np.mean(disparities)
        std_disp = np.std(disparities)
        cv2.putText(vis, f'Mean disparity: {mean_disp:.1f}px', (w1//2
- 100, h1 + 20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

    return vis

# After feature matching, use the visualization function
visualization = visualize_matches(left_image, right_image,
                                  keypoints_left, keypoints_right,
good_matches)
```

```
plt.figure(figsize=(15,10))
plt.imshow(cv2.cvtColor(visualization, cv2.COLOR_BGR2RGB))
plt.title("Feature Matches with Disparity Visualization")
plt.axis('off')
plt.show()

# Print some additional analysis
disparities = [abs(keypoints_left[m.queryIdx].pt[0] -
keypoints_right[m.trainIdx].pt[0])
               for m in good_matches]
print(f"\nDisparity Statistics:")
print(f"Mean disparity: {np.mean(disparities):.2f} pixels")
print(f"Std deviation: {np.std(disparities):.2f} pixels")
print(f"Min disparity: {min(disparities):.2f} pixels")
print(f"Max disparity: {max(disparities):.2f} pixels")

# Calculate approximate depths (if baseline and focal length are
known)
baseline = 0.25  # meters
focal_length_pixels = 1000  # This needs to be calibrated for your
camera
depths = [(baseline * focal_length_pixels) / d if d > 0 else
float('inf') for d in disparities]
valid_depths = [d for d in depths if d != float('inf')]

if valid_depths:
    print(f"\nDepth Statistics (approximate):")
    print(f"Mean depth: {np.mean(valid_depths):.2f} meters")
    print(f"Min depth: {min(valid_depths):.2f} meters")
    print(f"Max depth: {max(valid_depths):.2f} meters")
```

Feature Matches with Disparity Visualization


```
Disparity Statistics:
Mean disparity: 142.85 pixels
Std deviation: 50.17 pixels
Min disparity: 80.14 pixels
Max disparity: 472.02 pixels

Depth Statistics (approximate):
Mean depth: 1.89 meters
Min depth: 0.53 meters
Max depth: 3.12 meters
```