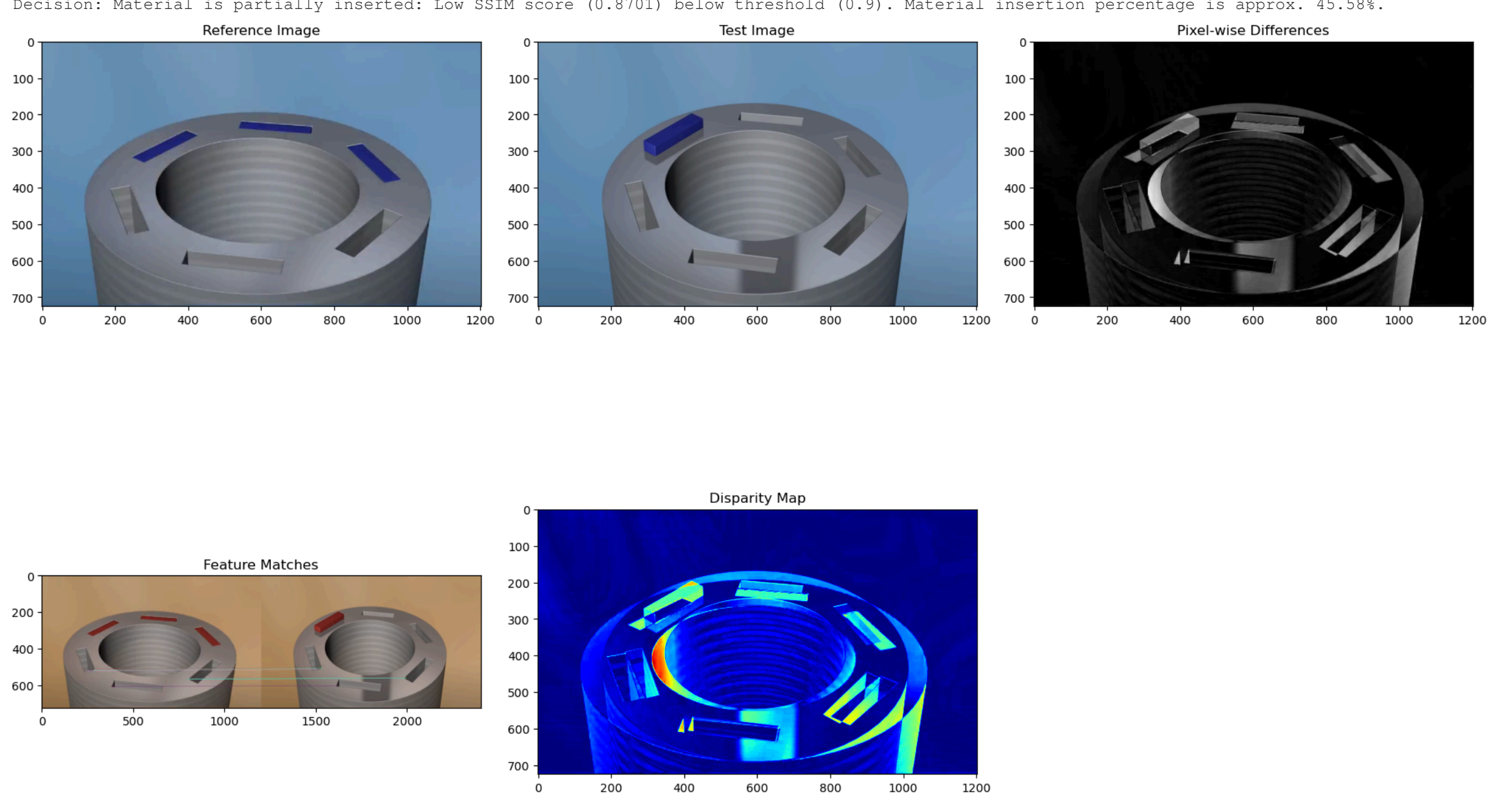
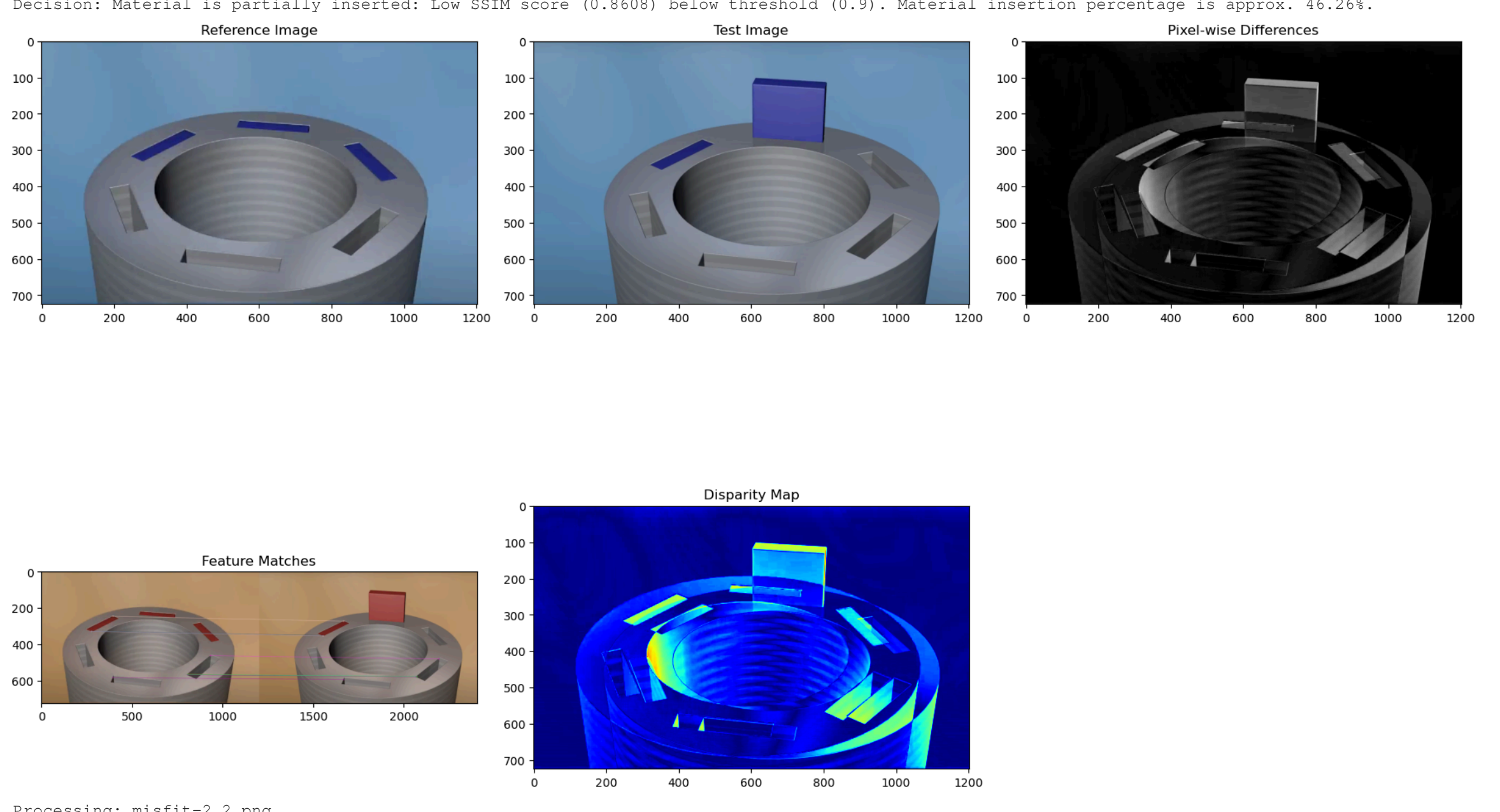
```python
import os
import cv2
import numpy as np
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt
from scipy.stats import pearsonr, spearmanr

# Load and preprocess images
def load_and_preprocess_images(ref_path, test_path):
    ref_image = cv2.imread(ref_path)
    test_image = cv2.imread(test_path)
    if ref_image is None or test_image is None:
        raise ValueError(f"Error: Unable to load images from {ref_path} or {test_path}.")

    # Resize test image to match reference image dimensions
    test_image_resized = cv2.resize(test_image, (ref_image.shape[1], ref_image.shape[0]))

    # Convert to grayscale
    ref_gray = cv2.cvtColor(ref_image, cv2.COLOR_BGR2GRAY)
    test_gray = cv2.cvtColor(test_image_resized, cv2.COLOR_BGR2GRAY)

    return ref_image, test_image_resized, ref_gray, test_gray

# Compare images using SSIM and pixel-wise differences
def compare_images(ref_gray, test_gray):
    similarity_index, _ = ssim(ref_gray, test_gray, full=True)
    diff_image = cv2.absdiff(ref_gray, test_gray)
    diff_image_normalized = cv2.normalize(diff_image, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    return diff_image_normalized, similarity_index

# Calculate SSIM threshold using statistical method (mean + std deviation)
def determine_ssim_threshold(ssim_scores):
    mean_ssim = np.mean(ssim_scores)
    std_ssim = np.std(ssim_scores)
    threshold = mean_ssim + std_ssim
    return threshold

# Feature matching
def feature_matching(ref_gray, test_gray):
    orb = cv2.ORB_create()
    keypoints_ref, descriptors_ref = orb.detectAndCompute(ref_gray, None)
    keypoints_test, descriptors_test = orb.detectAndCompute(test_gray, None)
    bf_matcher = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf_matcher.match(descriptors_ref, descriptors_test)
    matches = sorted(matches, key=lambda x: x.distance)
    return matches, keypoints_ref, keypoints_test

# Surface roughness analysis
def analyze_surface(depth_map):
    surface_roughness = np.std(depth_map)
    return surface_roughness

# Depth analysis
def depth_analysis(ref_gray, test_gray):
    disparity_map = cv2.absdiff(ref_gray, test_gray)
    return disparity_map

# Magnet insertion percentage
def determine_magnet_insertion(test_gray):
    threshold = 130  # Threshold to identify inserted magnets
    inserted_pixels = np.sum(test_gray < threshold)
    total_pixels = test_gray.size
    insertion_percentage = (inserted_pixels / total_pixels) * 100
    return insertion_percentage

# Calculate alignment score
def calculate_alignment_score(ssim_index):
    alignment_score = ssim_index * 100
    return alignment_score

# Decision-making system
def decision_making(ssim_index, insertion_percentage, surface_roughness):
    # Define thresholds
    threshold_ssim = 0.9
    threshold_insertion = 0.4326  # Example threshold for full insertion percentage
    threshold_roughness = 22.3971

    # Decision logic based on SSIM, insertion percentage, and surface roughness
    if ssim_index < threshold_ssim and insertion_percentage < threshold_insertion:
        decision = (
            f"Assembly issue detected:Material is partially inserted Low SSIM score ({ssim_index:.4f}) below threshold ({threshold_ssim}), "
            f"and material is partially inserted (approx. {insertion_percentage:.2f}% of required insertion)."
        )
    elif ssim_index < threshold_ssim:
        decision = (
            f"Material is partially inserted: Low SSIM score ({ssim_index:.4f}) below threshold ({threshold_ssim}). "
            f"Material insertion percentage is approx. {insertion_percentage:.2f}%."
        )
    elif insertion_percentage < threshold_insertion:
        decision = (
            f"Material is partially inserted: Insertion percentage is approx. {insertion_percentage:.2f}%, "
            f"below the full insertion threshold of {threshold_insertion * 100:.2f}%."
        )
    elif surface_roughness > threshold_roughness:
        decision = (
            f"Surface quality issue detected: Surface roughness ({surface_roughness:.4f}) exceeds acceptable threshold "
            f"({threshold_roughness}). Material insertion is approx. {insertion_percentage:.2f}%."
        )
    else:
        decision = (
            f"Assembly is acceptable: Material is fully inserted with insertion percentage of approx. {insertion_percentage:.2f}% "
            f"(meets or exceeds {threshold_insertion * 100:.2f}% threshold), SSIM score is {ssim_index:.4f}, "
            f"and surface roughness is within acceptable limits ({surface_roughness:.4f} <= {threshold_roughness})."
        )

    return decision

# Main execution function
def main():
    ref_path = input("Enter the reference image path (fully inserted magnet): ")
    input_folder = input("Enter the folder path containing test images: ")

    # Lists to store SSIM scores
    full_insertion_ssims = []
    partial_insertion_ssims = []

    # Get list of images in the input folder
    test_images = [f for f in os.listdir(input_folder) if f.endswith(('.jpg', '.png'))]

    # Process each image
    for test_image_name in test_images:
        test_path = os.path.join(input_folder, test_image_name)

        # Load and preprocess images
        ref_image, test_image, ref_gray, test_gray = load_and_preprocess_images(ref_path, test_path)

        # Compare images
        diff_image, ssim_index = compare_images(ref_gray, test_gray)

        # Collect SSIM score for further analysis
        if 'full' in test_image_name.lower():
            full_insertion_ssims.append(ssim_index)
        else:
            partial_insertion_ssims.append(ssim_index)

        # Calculate and display the threshold for the current image
        current_threshold = determine_ssim_threshold(partial_insertion_ssims)

        # Feature matching
        matches, keypoints_ref, keypoints_test = feature_matching(ref_gray, test_gray)

        # Depth analysis
        disparity_map = depth_analysis(ref_gray, test_gray)

        # Surface roughness analysis
        surface_roughness = analyze_surface(disparity_map)

        # Magnet insertion percentage
        insertion_percentage = determine_magnet_insertion(test_gray)

        # Alignment score
        alignment_score = calculate_alignment_score(ssim_index)

        # Decision-making
        decision = decision_making(ssim_index, insertion_percentage, surface_roughness)

        # Output the results in the specified format
        print(f"\nProcessing: {test_image_name}")
        #print(f"SSIM Score: {ssim_index:.4f}")
        #print(f"Current SSIM Threshold: {current_threshold:.4f}")
        #print(f"Magnet Insertion Percentage: {insertion_percentage:.2f}%")
        #print(f"Surface Roughness (Std Dev): {surface_roughness:.4f}")
        #print(f"Alignment Score: {alignment_score:.2f}%")
        print(f"Decision: {decision}")

        # Visualize results
        plt.figure(figsize=(18, 12))
        plt.subplot(2, 3, 1)
        plt.title('Reference Image')
        plt.imshow(cv2.cvtColor(ref_image, cv2.COLOR_BGR2RGB))
        plt.subplot(2, 3, 2)
        plt.title('Test Image')
        plt.imshow(cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB))
        plt.subplot(2, 3, 3)
        plt.title('Pixel-wise Differences')
        plt.imshow(diff_image, cmap='gray')
        matched_image = cv2.drawMatches(ref_image, keypoints_ref, test_image, keypoints_test, matches[:10], None, flags=cv2.DrawMatchesFlags_NOT_DR
        plt.subplot(2, 3, 4)
        plt.title('Feature Matches')
        plt.imshow(matched_image)
        plt.subplot(2, 3, 5)
        plt.title('Disparity Map')
        plt.imshow(disparity_map, cmap='jet')
        plt.tight_layout()
        plt.show()

    # Determine the threshold for SSIM classification
    ssim_threshold = determine_ssim_threshold(partial_insertion_ssims)

    # Compute SSIM score for fully inserted magnet (reference image)
    fully_inserted_image = load_and_preprocess_images(ref_path, ref_path)[2]
    full_ssim_score = compare_images(fully_inserted_image, fully_inserted_image)[1]

# Run the main function
if __name__ == "__main__":
    main()
```

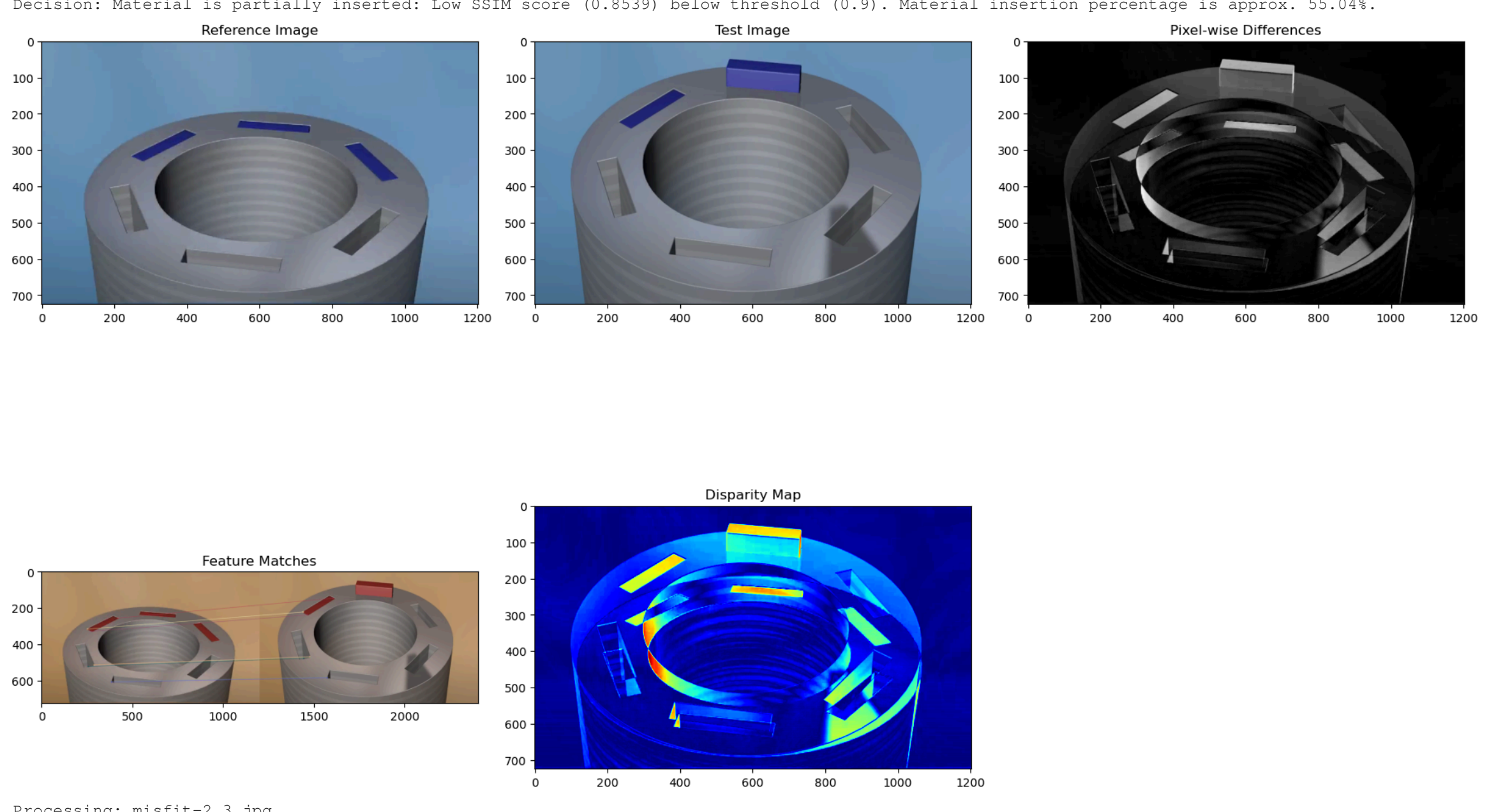
```
Processing: misfit-1.jpg
Decision: Material is partially inserted: Low SSIM score (0.8701) below threshold (0.9). Material insertion percentage is approx. 45.58%.
```



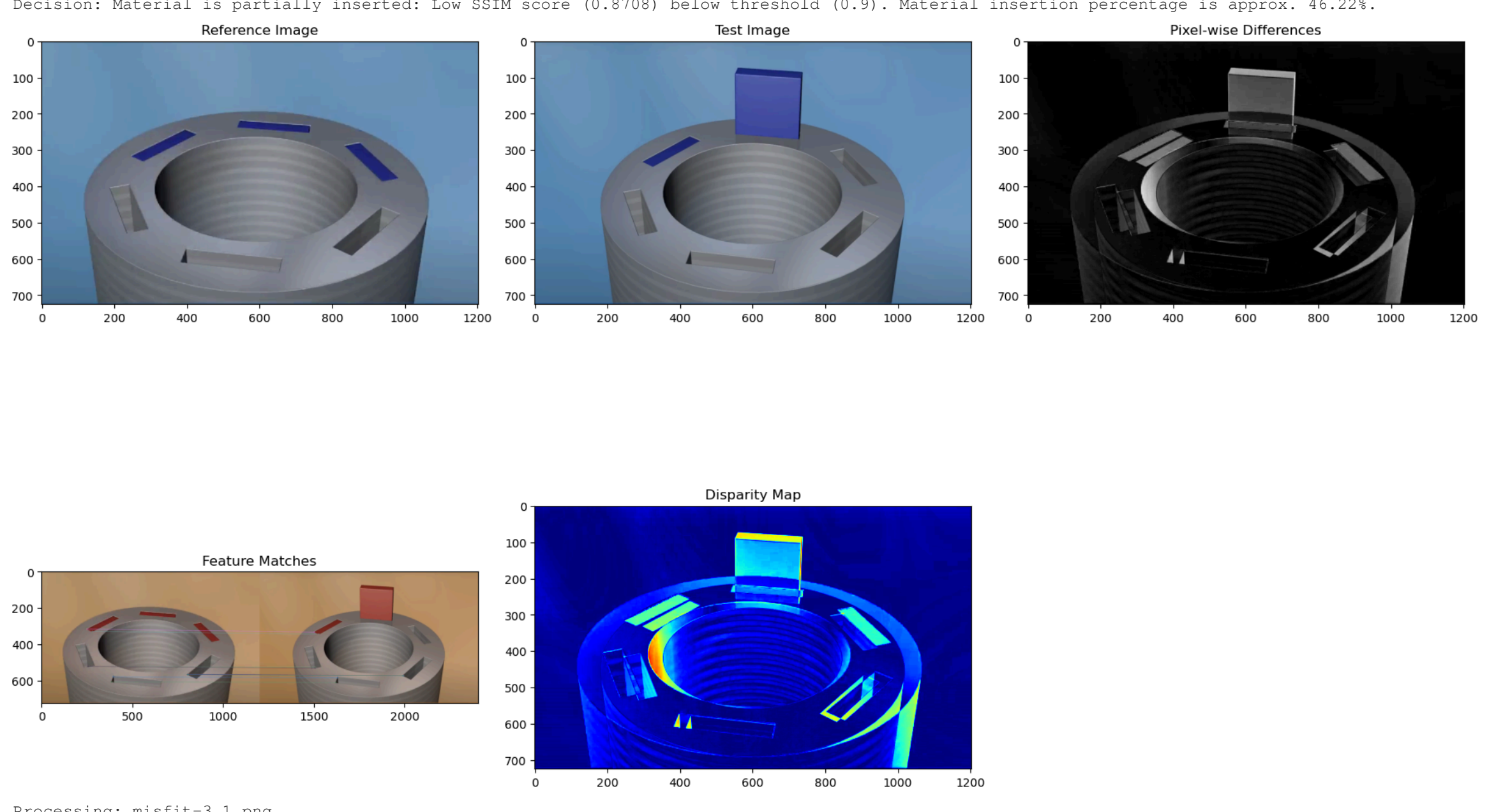

```
Processing: misfit-2_1.jpg
Decision: Material is partially inserted: Low SSIM score (0.8608) below threshold (0.9). Material insertion percentage is approx. 46.26%.
```



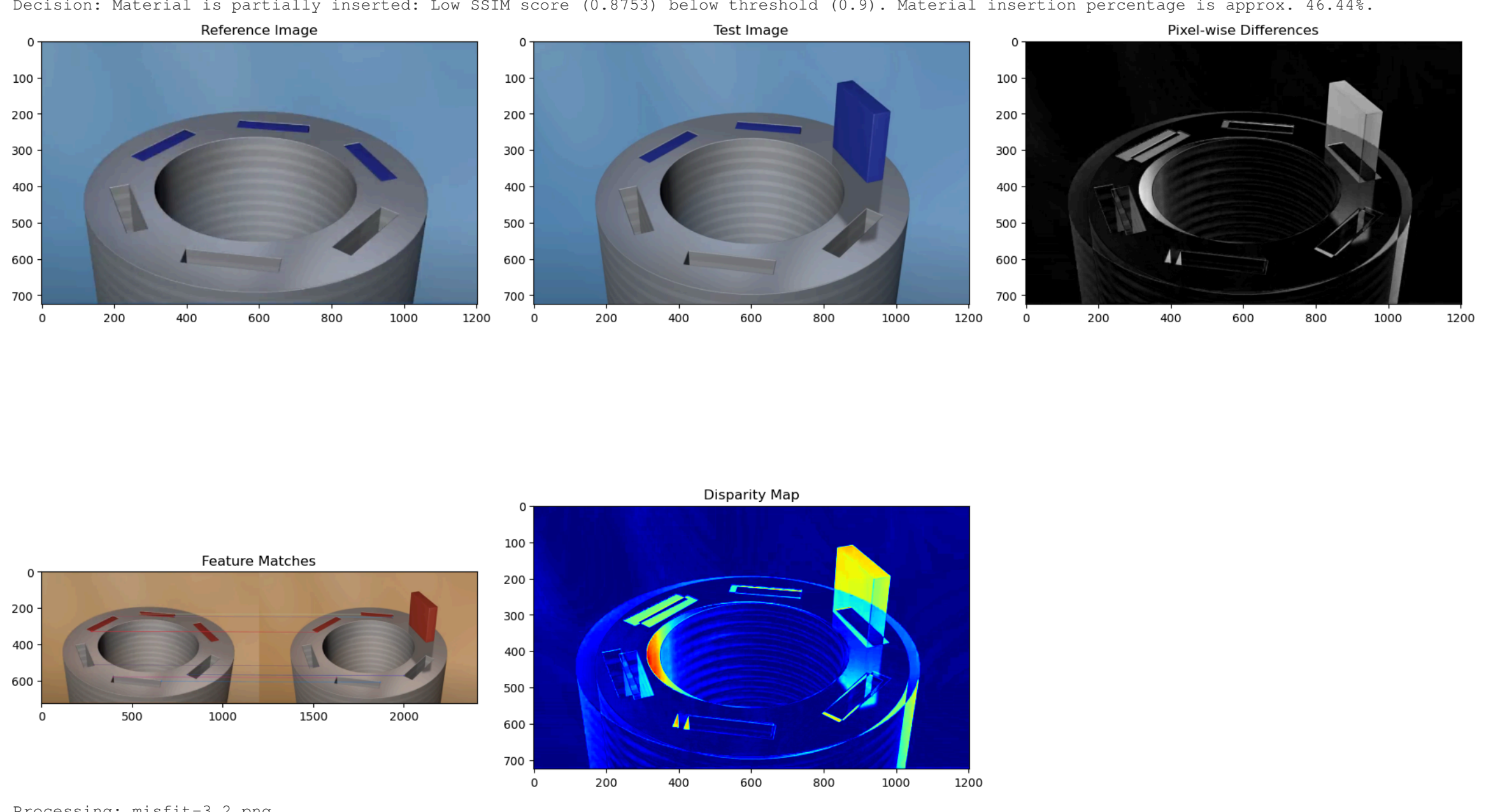

```
Processing: misfit-2_2.png
Decision: Material is partially inserted: Low SSIM score (0.8539) below threshold (0.9). Material insertion percentage is approx. 55.04%.
```



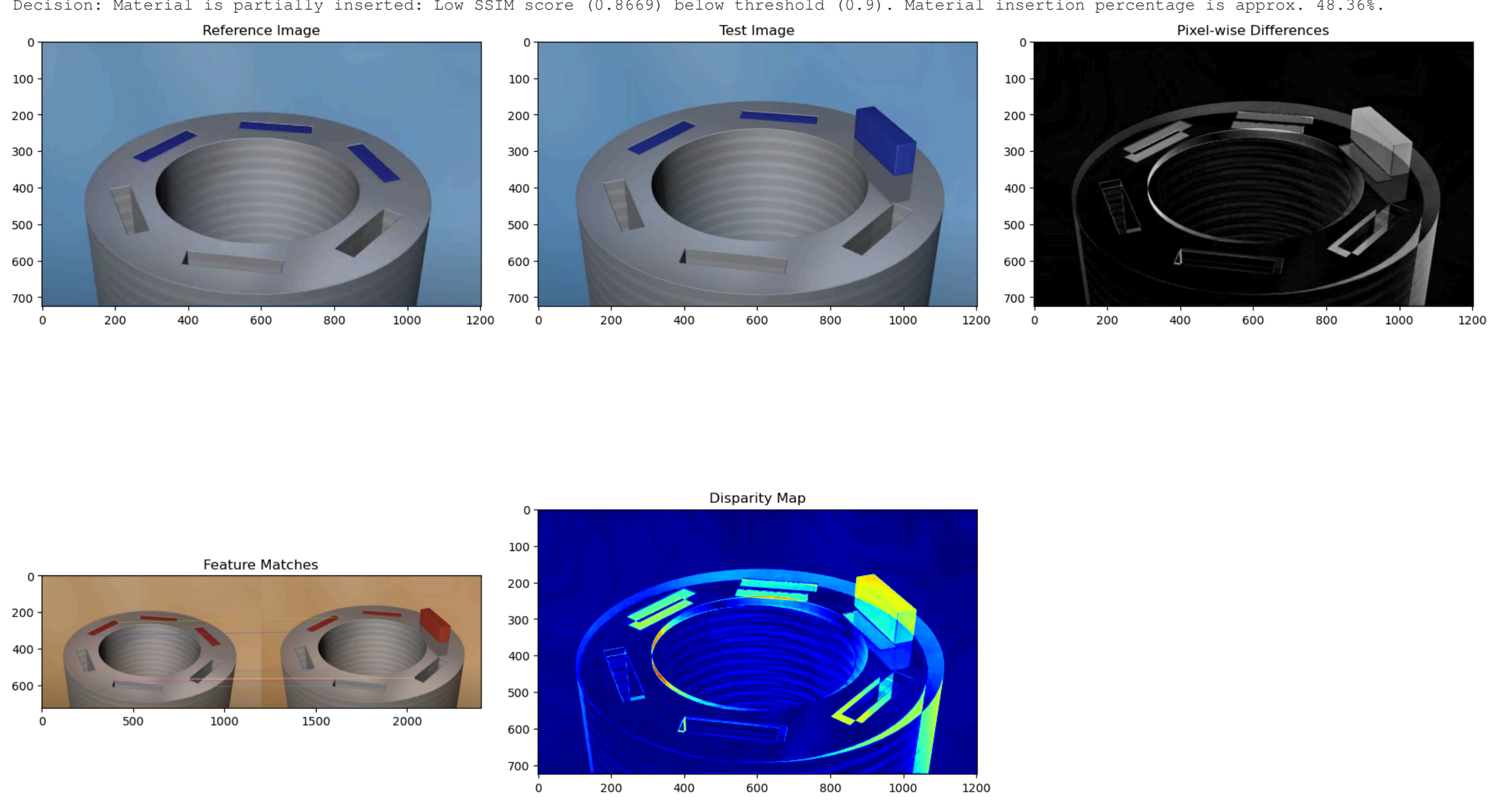

```
Processing: misfit-2_3.jpg
Decision: Material is partially inserted: Low SSIM score (0.8708) below threshold (0.9). Material insertion percentage is approx. 46.22%.
```



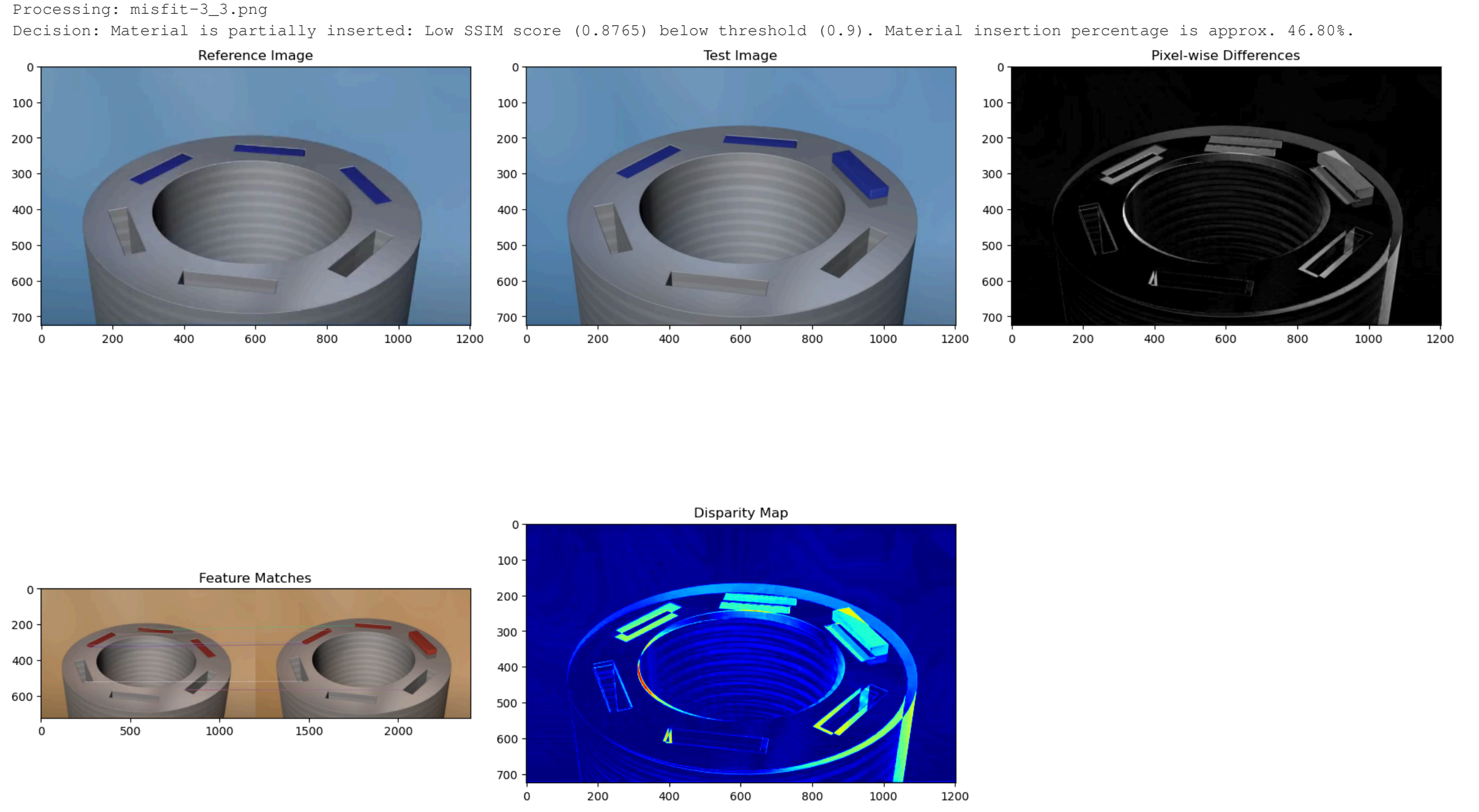

```
Processing: misfit-3_1.png
Decision: Material is partially inserted: Low SSIM score (0.8753) below threshold (0.9). Material insertion percentage is approx. 46.44%.
```



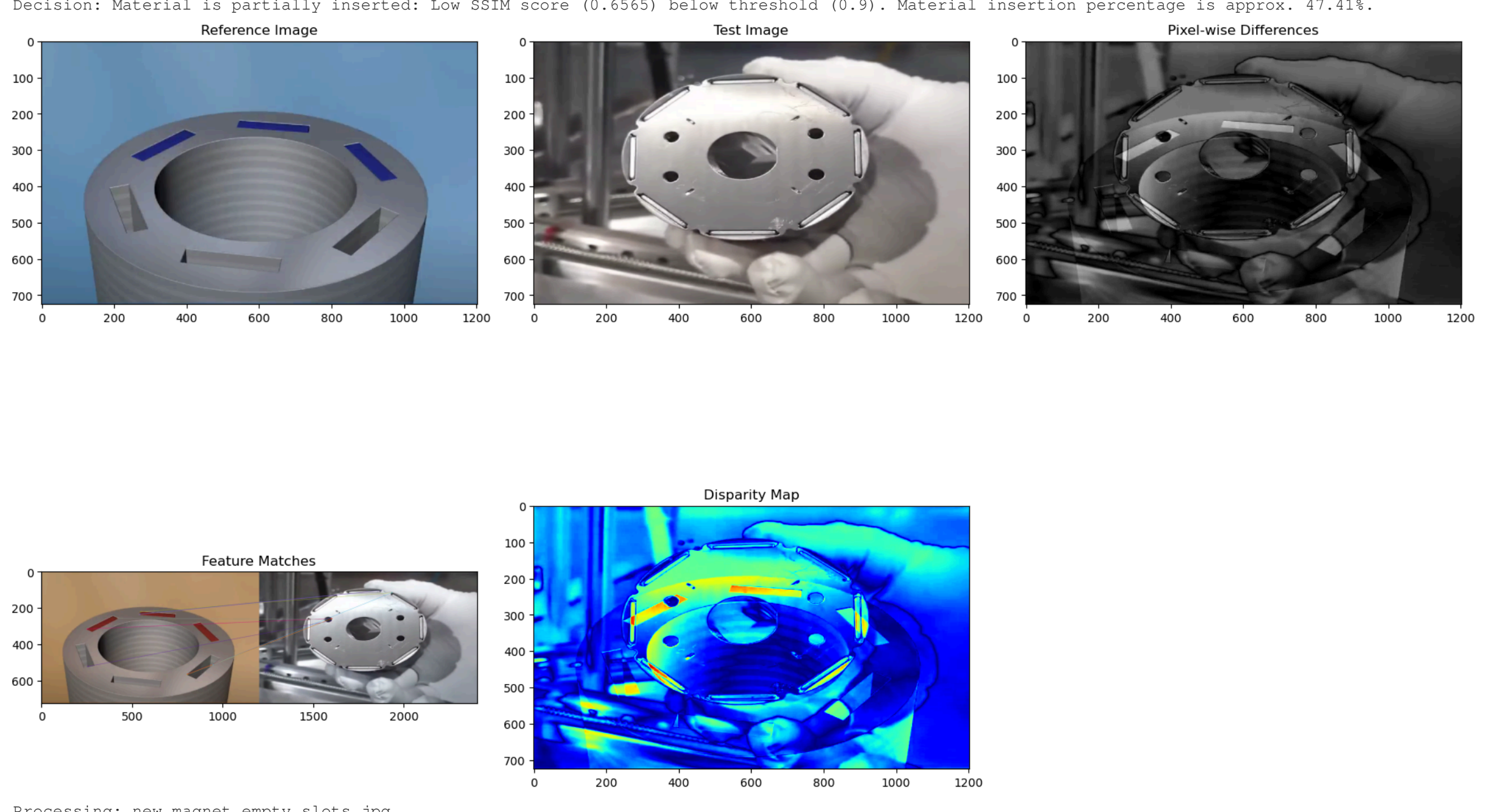

```
Processing: misfit-3_2.png
Decision: Material is partially inserted: Low SSIM score (0.8669) below threshold (0.9). Material insertion percentage is approx. 48.36%.
```




```
Processing: misfit-3_3.png
Decision: Material is partially inserted: Low SSIM score (0.8765) below threshold (0.9). Material insertion percentage is approx. 46.80%.
```




```
Processing: new magnet-proper.jpg
Decision: Material is partially inserted: Low SSIM score (0.6565) below threshold (0.9). Material insertion percentage is approx. 47.41%.
```




```
Processing: new magnet_empty slots.jpg
Decision: Material is partially inserted: Low SSIM score (0.6597) below threshold (0.9). Material insertion percentage is approx. 51.96%.
```