In [12]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

shitomasi edge detection

In [10]:

```python
image_path = "mom.png"
image = cv2.imread(image_path)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

corners = cv2.goodFeaturesToTrack(gray_image, maxCorners=100, qualityLevel=0.01, minDistance=10)
corners = np.int64(corners)

image1 = image.copy()
for i in corners:
    x, y = i.ravel()
    cv2.circle(image1, (x, y), 5, (0, 255, 0), -1)

corners_1000 = cv2.goodFeaturesToTrack(gray_image, maxCorners=1000, qualityLevel=0.01, minDistance=10)
corners_1000 = np.int64(corners_1000)

image2 = image.copy()
for i in corners_1000:
    x, y = i.ravel()
    cv2.circle(image2, (x, y), 5, (0, 255, 0), -1)

fig, axes = plt.subplots(1, 3, figsize=(18, 6), dpi=300)

# Original Image
axes[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axes[0].set_title('Original Image')
axes[0].axis('off')

# Shi-Tomasi Corner Detection with Corners = 100
axes[1].imshow(cv2.cvtColor(image1, cv2.COLOR_BGR2RGB))
axes[1].set_title('Shi-Tomasi Corner Detection with Corners = 100')
axes[1].axis('off')

# Shi-Tomasi Corner Detection with Corners = 1000
axes[2].imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
axes[2].set_title('Shi-Tomasi Corner Detection with Corners = 1000')
axes[2].axis('off')

plt.show()
```

Original Image

Shi-Tomasi Corner Detection with Corners = 100

Shi-Tomasi Corner Detection with Corners = 1000

Lab Task 1

Detecting tumors in medical images, such as mammograms, requires robust feature extraction techniques to identify regions of interest. Edge detection is one such technique that can be applied effectively in this context. Here's a step-by-step explanation of how edge detection can be used for tumor detection in mammograms:

1. **Preprocessing:** Begin by preprocessing the mammogram image to enhance its quality. This may involve operations like noise reduction, contrast enhancement, and resizing to ensure uniformity in image size and quality across the dataset.
2. **Grayscale Conversion:** Convert the preprocessed image to grayscale if it's not already in grayscale. This simplifies further processing, as edge detection typically works on grayscale images.
3. **Edge Detection:** Apply an edge detection algorithm to the grayscale image. Common edge detection algorithms include the Sobel operator, Canny edge detector, and Prewitt operator. These algorithms work by identifying abrupt changes in pixel intensity, which often correspond to edges in the image.
4. **Thresholding:** After applying the edge detection algorithm, you'll get an edge map, which is essentially a binary image where edges are highlighted. To isolate potential tumor regions, apply thresholding to this edge map to segment the areas of interest.
5. **Region of Interest (ROI) Extraction:** Once you have thresholded the edge map, you can identify connected components or regions in the binary image. These regions represent potential tumor areas.
6. **Feature Extraction from ROIs:** Extract relevant features from the detected ROIs, which can include shape, texture, and statistical properties. These features can be used to further classify and characterize potential tumors.
7. **Classification:** Finally, use a machine learning or deep learning model to classify the detected regions as either tumors or non-tumors based on the extracted features. This step involves training the model on labeled data to make predictions on new mammograms.

For enhancing tumor detection in medical images, apart from edge detection, another valuable technique is:

Texture Analysis: Texture analysis is suitable for tumor detection because it helps capture fine details and patterns within the image. Here's how it works:

1. **Preprocessing:** As with edge detection, begin with image preprocessing to ensure consistency and enhance image quality.
2. **Grayscale Conversion:** Convert the preprocessed image to grayscale.
3. **Texture Analysis:** Apply texture analysis techniques such as Haralick texture features, gray-level co-occurrence matrix (GLCM), or Gabor filters to extract texture information from the grayscale image. These methods capture information about variations in pixel intensity and patterns.
4. **Feature Extraction:** Extract texture features from the analyzed image regions. These features could include contrast, entropy, energy, and homogeneity, among others.
5. **Classification:** Similar to the edge detection approach, use a classification model to classify the regions as either tumors or non-tumors based on the extracted texture features. Machine learning or deep learning models can be trained to make these predictions.

Texture analysis complements edge detection by providing additional information about the fine-grained patterns and variations in the image, which can be particularly useful for detecting tumors with subtle or irregular shapes.

In summary, combining edge detection and texture analysis as feature extraction techniques can significantly enhance the accuracy of tumor detection in mammograms by capturing both structural and textural information within the medical images.

Lab Task 2

2 Harris Corner Detection

In [9]:

```python
image_path = "mom.png"
image = cv2.imread(image_path)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

block_size = 2
k = 0.04

corner_scores = cv2.cornerHarris(gray_image, block_size, 3, k)

threshold = 0.01 * corner_scores.max()
corners = np.where(corner_scores > threshold)

corner_image = image.copy()
corner_image[corners] = [0, 0, 255]

threshold = 0.0001 * corner_scores.max()
corners1 = np.where(corner_scores > threshold)

corner_image2 = image.copy()
corner_image2[corners1] = [0, 0, 255]

fig, axes = plt.subplots(1, 3, figsize=(18, 6))

axes[0].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
axes[0].set_title('Original Image')
axes[0].axis('off')

axes[1].imshow(cv2.cvtColor(corner_image, cv2.COLOR_BGR2RGB))
axes[1].set_title('Harris Corner Detection with Threshold = 0.01')
axes[1].axis('off')

axes[2].imshow(cv2.cvtColor(corner_image2, cv2.COLOR_BGR2RGB))
axes[2].set_title('Harris Corner Detection with Threshold = 0.0001')
axes[2].axis('off')

plt.show()
```

Lab Task 3: Corner Detection in Real-time Video

In [22]:

```
1 # to see output run on your pc. It will show live detection.
```

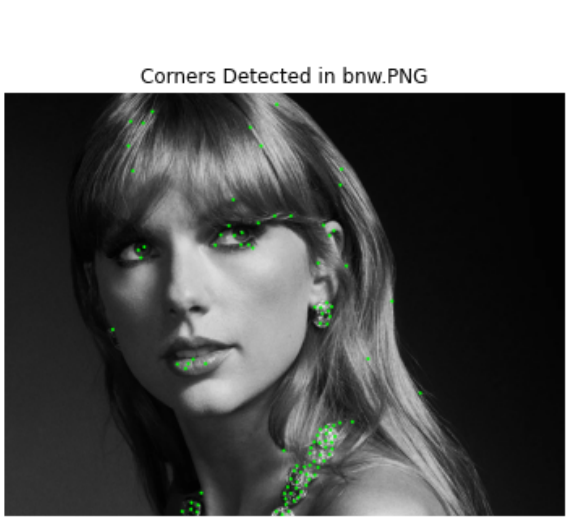
In [ ]:

```python
import cv2
import numpy as np

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Shi-Tomasi corner detection
    corners = cv2.goodFeaturesToTrack(gray, maxCorners=100, qualityLevel=0.01, minDistance=10)

    # Drawing circles at the detected corners
    if corners is not None:
        corners = np.int0(corners)
        for corner in corners:
            x, y = corner.ravel()
            cv2.circle(frame, (x, y), 5, (0, 255, 0), -1)

    cv2.imshow("Corner Detection", frame)

    if cv2.waitKey(1) & 0xFF == ord('q'): # press small q to exit
        break
cap.release()
cv2.destroyAllWindows()
```

Lab Task 4: Corner Detection for Image Stitching

In [18]:

```python
image_filenames = ['bnw.PNG', 'mom.png', 'rgb.PNG']

max_corners = 100
quality_level = 0.01
min_distance = 10

# single-row subplot for displaying images in one line
fig, axes = plt.subplots(1, len(image_filenames), figsize=(15, 5))

for i, image_filename in enumerate(image_filenames):
    image = cv2.imread(image_filename)
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    corners = cv2.goodFeaturesToTrack(gray_image, maxCorners=max_corners, qualityLevel=quality_level, minDist

    if corners is not None:
        corners = np.int0(corners)

        for corner in corners:
            x, y = corner.ravel()
            cv2.circle(image, (x, y), 3, (0, 255, 0), -1)

        axes[i].imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
        axes[i].set_title('Corners Detected in ' + image_filename)
        axes[i].axis('off')

plt.tight_layout()

plt.show()
```



Lab Task 5: Feature Detection and Matching using ORB Detector

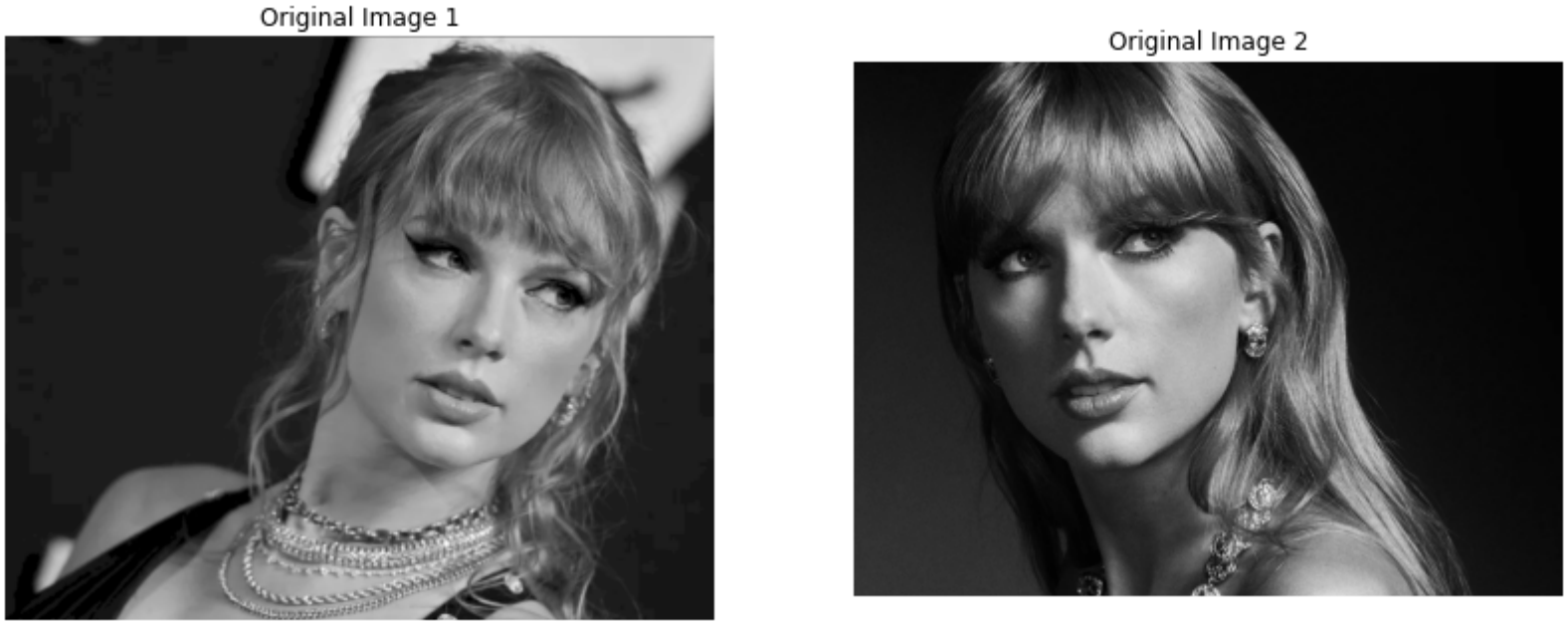
In [19]:

```
image1 = cv2.imread('mom.png', cv2.IMREAD_GRAYSCALE)
image2 = cv2.imread('bnw.png', cv2.IMREAD_GRAYSCALE)

orb = cv2.ORB_create()
keypoints1, descriptors1 = orb.detectAndCompute(image1, None)
keypoints2, descriptors2 = orb.detectAndCompute(image2, None)

# Creating a Brute-Force Matcher
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Matching descriptors between the two images
matches = bf.match(descriptors1, descriptors2)

# Sorting matches by distance
matches = sorted(matches, key=lambda x: x.distance)

# Drawing the first 10 matches
matched_image = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches[:10], outImg=None)
```

In [20]:

```
plt.figure(figsize=(14, 16))

plt.subplot(1, 2, 1)
plt.imshow(image1, cmap='gray')
plt.title('Original Image 1')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(image2, cmap='gray')
plt.title('Original Image 2')
plt.axis('off')
```

Out[20]: (-0.5, 1031.5, 778.5, -0.5)

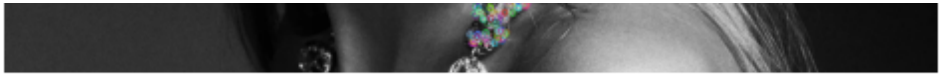


In [21]:

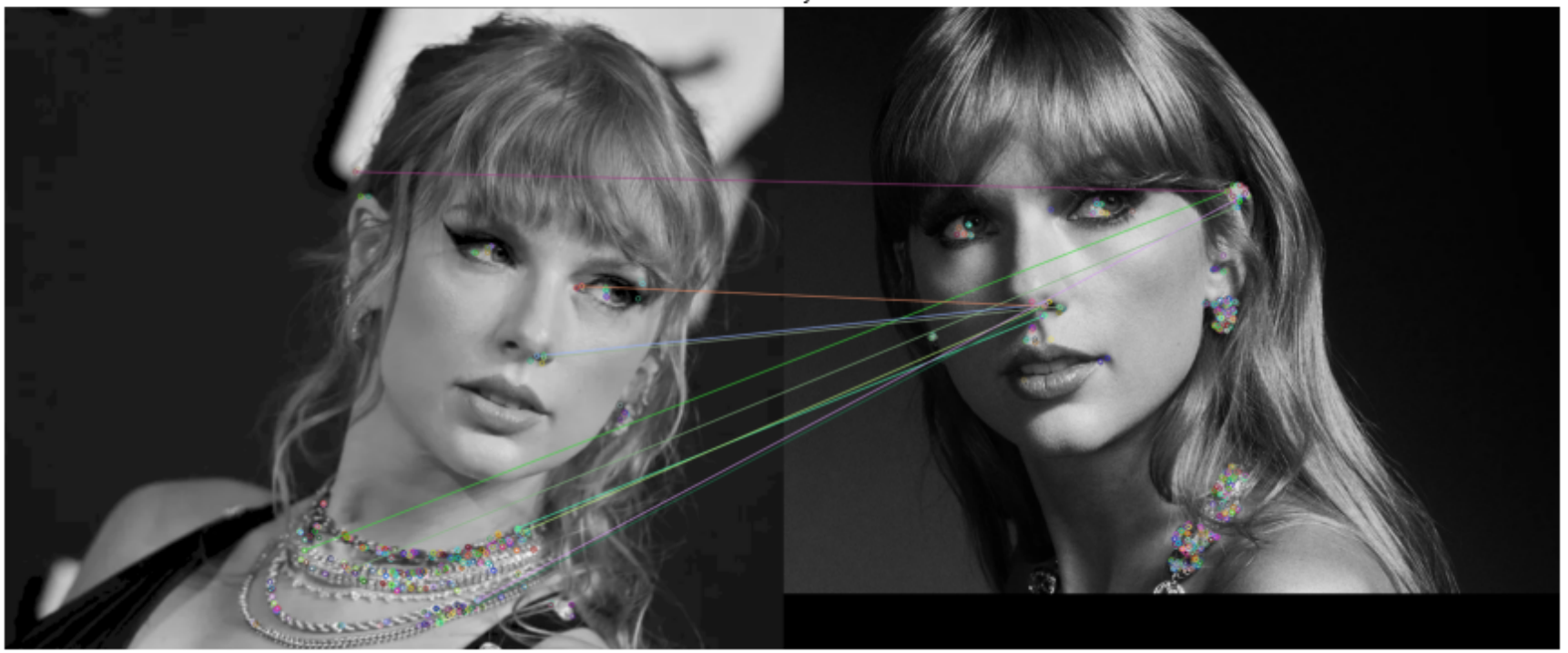
```
plt.figure(figsize=(14, 16))

plt.subplot(3, 1, 1)
plt.imshow(cv2.drawKeypoints(image1, keypoints1, None))
plt.title('Key Points in Image 1')
plt.axis('off')

plt.subplot(3, 1, 2)
plt.imshow(cv2.drawKeypoints(image2, keypoints2, None))
plt.title('Key Points in Image 2')
plt.axis('off')

plt.subplot(3, 1, 3)
plt.imshow(matched_image)
plt.title('Matched Key Points')
plt.axis('off')

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

Key Points in Image 1

Key Points in Image 2



Matched Key Points



---------- K201716 THE END ----------