In [ ]:

```
import cv2
import numpy as np
import glob

# Load Calibration Parameters:
#     Function load_calibration(calibration_file): calibration_matrix
#         Load calibration data from the file
#         Extract camera matrix and distortion coefficients
#         Return camera matrix and distortion coefficients

def load_calibration(calibration_file):
    calibration_data = np.load(calibration_file)
    mtx = calibration_data['arr_0']
    dist = calibration_data['arr_1']
    return mtx, dist

# calibration_file = 'calibration_matrix.npz'
# mtx, dist = load_calibration(calibration_file)

# print("Camera Matrix:\n", mtx)
# print("Distortion Coefficients:\n", dist)


# Undistort Image:
#     Function undistort_image(image, camera_matrix, dist_coeffs):
#         Get image dimensions (height, width)
#         Compute new camera matrix for undistortion
#         Undistort the image (use cv2 undistort)
#         Crop the undistorted image using ROI
#         Return undistorted image

def undistort_image(image, camera_matrix, dist_coeffs):
    h, w = image.shape[:2]
    new_camera_mtx, roi = cv2.getOptimalNewCameraMatrix(camera_matrix, di

    undistorted_image = cv2.undistort(image, camera_matrix, dist_coeffs,

    x, y, w, h = roi
    undistorted_image = undistorted_image[y:y+h, x:x+w]

    return undistorted_image


# Harris Corner Detection:
#     Function harris_corner_detection(image):
#         Convert the image to grayscale
#         Apply Harris corner detection
#         Dilate corners
#         Mark corners on the image
#         Return image with marked corners and detected corners

def harris_corner_detection(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = np.float32(gray)

    dst = cv2.cornerHarris(gray, 2, 3, 0.04)
    dst = cv2.dilate(dst, None)
```

```
    image[dst > 0.01 * dst.max()] = [0, 0, 255] #marks up the image

    return image, dst
#dst is a 2D array


# Match Features Between Images:
#     Function match_features(image1, image2):
#         Detect keypoints and descriptors in image1 using SIFT
#         Detect keypoints and descriptors in image2 using SIFT
#         Match descriptors using brute-force matcher
#         Extract matched points from both images
#         Return matched points from image1 and image2

def match_features(image1, image2):
    # Example: Match features using SIFT
    sift = cv2.SIFT_create()
    kp1, des1 = sift.detectAndCompute(image1, None)
    kp2, des2 = sift.detectAndCompute(image2, None)

    # Match descriptors using brute-force matcher
    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des1, des2, k=2)

    # Apply ratio test
    good_matches = []
    for m, n in matches:
        if m.distance < 0.75 * n.distance:
            good_matches.append(m)

    # Ensure enough matches for homography calculation
    if len(good_matches) < 4:
        return None, None

    points1 = np.float32([kp1[m.queryIdx].pt for m in good_matches]).resh
    points2 = np.float32([kp2[m.trainIdx].pt for m in good_matches]).resh
    #this part extracts points that fit the criteria
    #retrieves the coordinates of the keypoint in the first
    # image (image1) corresponding to the descriptor index m.queryIdx
    print(points1, points2)

    return points1, points2




# Create Mosaic:
#     Function create_mosaic(images, camera_matrix, dist_coeffs):
#         Undistort all images using undistort_image function
#         Initialize mosaic with the first undistorted image
#         For each subsequent undistorted image:
#             Detect Harris corners in both mosaic and current image usin
#             Match features between mosaic and current image using match
#             Estimate homography using matched points
#             Warp mosaic image using the estimated homography
#             Blend current image into mosaic
#         Return final mosaic image

def create_mosaic(images, camera_matrix, dist_coeffs):
    undistorted_images = [undistort_image(img, camera_matrix, dist_coeffs
```

```
    mosaic = undistorted_images[0].copy()

    for i in range(1, len(undistorted_images)):
        mosaic_corners, _ = harris_corner_detection(mosaic)
        current_corners, _ = harris_corner_detection(undistorted_images[i

        points1, points2 = match_features(mosaic_corners, current_corners

        if points1 is None or points2 is None:
            continue

        points1 = points1.reshape(-1, 1, 2)
        points2 = points2.reshape(-1, 1, 2)

        H, _ = cv2.findHomography(points1, points2, cv2.RANSAC, 5.0)

        warped_image = cv2.warpPerspective(undistorted_images[i], H, (mos

        mask = np.zeros_like(mosaic, dtype=np.uint8)
        mask[warped_image[:, :, 0] != 0] = 255

        try:
            blend = cv2.seamlessClone(warped_image, mosaic, mask, (0, 0),
        except cv2.error as e:
            print(f"Error in seamlessClone: {e}")
            print(f"Dimensions: mosaic={mosaic.shape}, warped_image={warp
            continue

        mosaic = blend

    return mosaic



# Main:
#     Load camera matrix and distortion coefficients from calibration fil
#     Load images from specified directory
#     Create mosaic using create_mosaic function
#     Save the mosaic image to a file
```

In [ ]:

```
def main():
    calibration_file = "calibration_matrix.npz"
    camera_matrix, dist_coeffs = load_calibration(calibration_file)
    images_path = '/home/joy/laboratory_2024/week_1_Hw/camera_calibration
    #mosaic latin student center
    images_files = glob.glob(images_path)
    images = [cv2.imread(img) for img in images_files]

    # Create mosaic using create_mosaic function
    mosaic_image = create_mosaic(images, camera_matrix, dist_coeffs)

    # # Display the mosaic image
    cv2.imshow('Mosaic', mosaic_image)
    cv2.waitKey(50000)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

```
[[[  6.1556625 149.36554  ]]

 [[  6.6627803 153.17912  ]]

 [[ 18.45982    83.939255 ]]

 [[ 18.521297  191.9274   ]]

 [[ 29.518925  185.12979  ]]

 [[ 47.008278  151.9112   ]]

 [[ 47.46058    72.56242  ]]

 [[ 69.49294   135.644    ]]] [[[ 14.332233  189.97389  ]]

 [[  7.0647006 156.86507  ]]

 [[ 11.945629   90.12427  ]]

 [[ 14.332233  189.97389  ]]

 [[ 22.427622  179.482    ]]

 [[ 27.342367  198.59018  ]]

 [[ 51.203167   69.75192  ]]

 [[ 16.615416  185.67725  ]]]
Error in seamlessClone: OpenCV(4.10.0) /io/opencv/modules/core/src/matri
x.cpp:808: error: (-215:Assertion failed) 0 <= roi.x && 0 <= roi.width &&
roi.x + roi.width <= m.cols && 0 <= roi.y && 0 <= roi.height && roi.y + ro
i.height <= m.rows in function 'Mat'

Dimensions: mosaic=(239, 73, 3), warped_image=(239, 73, 3), mask=(239, 73,
3)
[[[  6.555626 152.79846 ]]

 [[ 31.2788   185.46613 ]]

 [[ 34.69577   84.803825]]

 [[ 45.333897  75.377846]]] [[[  6.19398  132.22882 ]]

 [[ 41.815273 183.36057 ]]

 [[ 34.73017   87.05955 ]]

 [[ 34.73017   87.05955 ]]]
Error in seamlessClone: OpenCV(4.10.0) /io/opencv/modules/core/src/matri
x.cpp:808: error: (-215:Assertion failed) 0 <= roi.x && 0 <= roi.width &&
roi.x + roi.width <= m.cols && 0 <= roi.y && 0 <= roi.height && roi.y + ro
i.height <= m.rows in function 'Mat'

Dimensions: mosaic=(239, 73, 3), warped_image=(239, 73, 3), mask=(239, 73,
3)
```

In [ ]:
```python
def main():
    calibration_file = "calibration_matrix.npz"
    camera_matrix, dist_coeffs = load_calibration(calibration_file)
```

```
    images_path = 'home/joy/laboratory_2024/week_1_Hw/camera_calibration_
    images_files = glob.glob(images_path)
    images = [cv2.imread(img) for img in images_files]

    # Create mosaic using create_mosaic function
    mosaic_image = create_mosaic(images, camera_matrix, dist_coeffs)

    # # Display the mosaic image
    cv2.imshow('Mosaic', mosaic_image)
    cv2.waitKey(50000)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

In [ ]:

```
def main():
    calibration_file = "calibration_matrix.npz"
    camera_matrix, dist_coeffs = load_calibration(calibration_file)
    images_path = 'home/joy/laboratory_2024/week_1_Hw/camera_calibration_
    images_files = glob.glob(images_path)
    images = [cv2.imread(img) for img in images_files]

    # Create mosaic using create_mosaic function
    mosaic_image = create_mosaic(images, camera_matrix, dist_coeffs)

    # # Display the mosaic image
    cv2.imshow('Mosaic', mosaic_image)
    cv2.waitKey(50000)
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```