Name: Nguyễn Trương Xuân Nghiêm

ID: 2151010246

Class: DH21CS01C

Assignment 1:

Source Code:

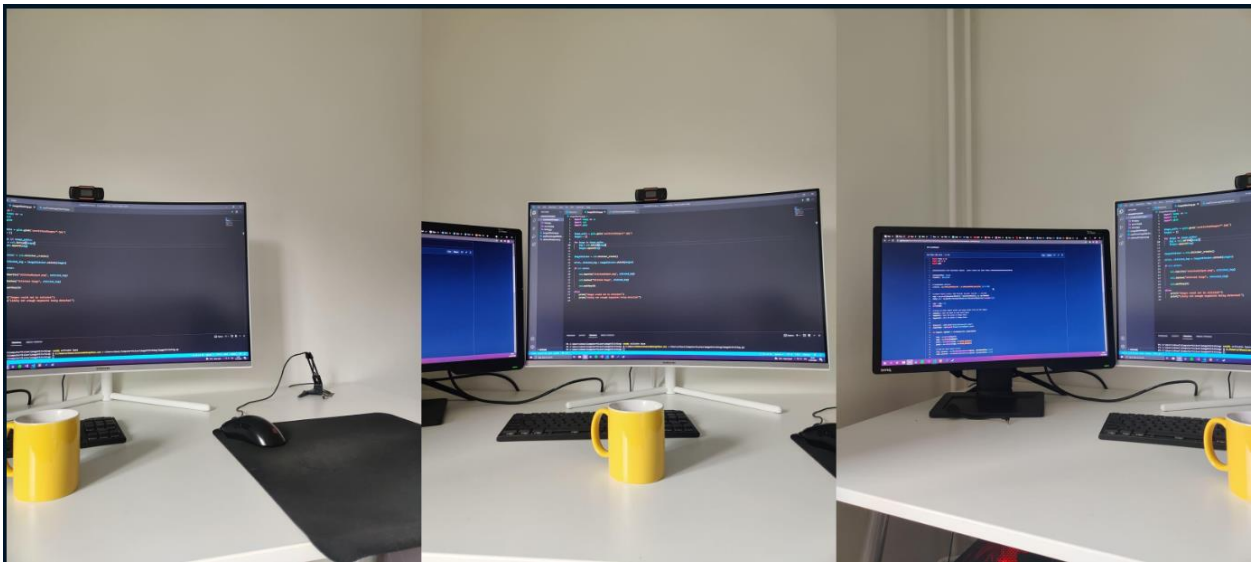
Assignment1_NotUsingSticher.py > stitch_images

```python
import cv2
import numpy as np

def stitch_images(img1, img2, img3):
    # Convert images to grayscale
    gray1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    gray2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    gray3 = cv2.cvtColor(img3, cv2.COLOR_BGR2GRAY)

    # Detect keypoints and descriptors
    sift = cv2.SIFT_create()
    kp1, des1 = sift.detectAndCompute(gray1, None)
    kp2, des2 = sift.detectAndCompute(gray2, None)
    kp3, des3 = sift.detectAndCompute(gray3, None)

    # Match keypoints
    bf = cv2.BFMatcher()
    matches1_2 = bf.knnMatch(des1, des2, k=2)
    matches2_3 = bf.knnMatch(des2, des3, k=2)

    # Apply ratio test
    good_matches1_2 = []
    for m, n in matches1_2:
        if m.distance < 0.75 * n.distance:
            good_matches1_2.append(m)

    good_matches2_3 = []
    for m, n in matches2_3:
        if m.distance < 0.75 * n.distance:
            good_matches2_3.append(m)
```

```
Click to add a breakpoint
    # Estimate homographies
    src_pts1_2 = np.float32(
        [kp1[m.queryIdx].pt for m in good_matches1_2]).reshape(-1, 1, 2)
    dst_pts1_2 = np.float32(
        [kp2[m.trainIdx].pt for m in good_matches1_2]).reshape(-1, 1, 2)
    H1_2, _ = cv2.findHomography(src_pts1_2, dst_pts1_2, cv2.RANSAC, 5.0)

    src_pts2_3 = np.float32(
        [kp2[m.queryIdx].pt for m in good_matches2_3]).reshape(-1, 1, 2)
    dst_pts2_3 = np.float32(
        [kp3[m.trainIdx].pt for m in good_matches2_3]).reshape(-1, 1, 2)
    H2_3, _ = cv2.findHomography(src_pts2_3, dst_pts2_3, cv2.RANSAC, 5.0)

    # Warp images
    h1, w1 = img1.shape[:2]
    h2, w2 = img2.shape[:2]
    h3, w3 = img3.shape[:2]
    warped_img2 = cv2.warpPerspective(img2, H1_2, (w1+w2, h1))
    warped_img3 = cv2.warpPerspective(img3, H2_3, (w2+w3, h2))

    # Resize warped images to match expected dimensions
    warped_img2 = cv2.resize(warped_img2, (w1, h1))
    warped_img3 = cv2.resize(warped_img3, (w3, h3))

    # Combine images
    panorama = np.zeros((max(h1, h3), w1+w2+w3, 3), dtype=np.uint8)
    panorama[:h1, :w1] = img1
    panorama[:h2, w1:w1+w2] = warped_img2
    panorama[:h3, w1+w2:] = warped_img3
```

```
        panorama[:h3, w1+w2:] = warped_img3

        return panorama


    # Load images
    img1 = cv2.imread('images/3_images/first.jpg')
    img2 = cv2.imread('images/3_images/second.jpg')
    img3 = cv2.imread('images/3_images/third.jpg')

    # Check if images are loaded successfully
    if img1 is None or img2 is None or img3 is None:
        print("Error: One or more images could not be loaded.")
        exit()

    # Stitch images
    panorama = stitch_images(img1, img2, img3)

    # Scale result
    scale_percent = 30  # percent of original size
    width = int(panorama.shape[1] * scale_percent / 100)
    height = int(panorama.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized_panorama = cv2.resize(panorama, dim, interpolation=cv2.INTER_AREA)

    # Display panorama
    cv2.imshow('Panorama', resized_panorama)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

```

Result:



Assignment 2:

Source Code:

```python
import numpy as np
import cv2
import glob
import imutils

image_paths = glob.glob('images\\3_images\\*.jpg')
images = []


for image in image_paths:
    img = cv2.imread(image)
    images.append(img)
    cv2.imshow("Image", img)
    cv2.waitKey(0)


imageStitcher = cv2.Stitcher_create()

error, stitched_img = imageStitcher.stitch(images)

if not error:

    scale_percent = 30  # percent of original size
    width = int(stitched_img.shape[1] * scale_percent / 100)
    height = int(stitched_img.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized_panorama = cv2.resize(
        stitched_img, dim, interpolation=cv2.INTER_AREA)

    cv2.imwrite("stitchedOutput.png", resized_panorama)
    cv2.imshow("Stitched Img", resized_panorama)
    cv2.waitKey(0)
```

```
33
34      stitched_img = cv2.copyMakeBorder(
35          stitched_img, 10, 10, 10, 10, cv2.BORDER_CONSTANT, (0, 0, 0))
36
37      gray = cv2.cvtColor(stitched_img, cv2.COLOR_BGR2GRAY)
38      thresh_img = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]
39
40      cv2.imshow("Threshold Image", thresh_img)
41      cv2.waitKey(0)
42
43      contours = cv2.findContours(
44          thresh_img.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
45
46      contours = imutils.grab_contours(contours)
47      areaOI = max(contours, key=cv2.contourArea)
48
49      mask = np.zeros(thresh_img.shape, dtype="uint8")
50      x, y, w, h = cv2.boundingRect(areaOI)
51      cv2.rectangle(mask, (x, y), (x + w, y + h), 255, -1)
52
53      minRectangle = mask.copy()
54      sub = mask.copy()
```

```
56      while cv2.countNonZero(sub) > 0:
57          minRectangle = cv2.erode(minRectangle, None)
58          sub = cv2.subtract(minRectangle, thresh_img)
59
60      contours = cv2.findContours(
61          minRectangle.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
62
63      contours = imutils.grab_contours(contours)
64      areaOI = max(contours, key=cv2.contourArea)
65
66      cv2.imshow("minRectangle Image", minRectangle)
67      cv2.waitKey(0)
68
69      x, y, w, h = cv2.boundingRect(areaOI)
70
71      stitched_img = stitched_img[y:y + h, x:x + w]
72
73      scale_percent = 30  # percent of original size
74      width = int(stitched_img.shape[1] * scale_percent / 100)
75      height = int(stitched_img.shape[0] * scale_percent / 100)
76      dim = (width, height)
77      resized_panorama = cv2.resize(
78          stitched_img, dim, interpolation=cv2.INTER_AREA)
79
80      cv2.imwrite("stitchedOutputProcessed.png", resized_panorama)
81
82      cv2.imshow("Stitched Image Processed", resized_panorama)
83
84      cv2.waitKey(0)
```

```
87  else:
88      print("Images could not be stitched!")
89      print("Likely not enough keypoints being detected!")
```

Result: