ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO
DIGITAL IMAGE PROCESSION AND COMPUTER VISION
PROJECT 5
Panoramic photos (Image Stitching)

GVHD: Ths. Võ Thanh Hùng

—o0o—

SVTH: Nguyễn Tiến Phát - 2011797

TP. HỒ CHÍ MINH, 5/2024

Mục lục

1 Giới thiệu **1**
1.1 Khái niệm về Image Stitching . . . 1

2 Hiện thực **2**
2.1 Cơ sở lý thuyết . . . 2
2.1.1 Phát hiện đặc trưng . . . 2
2.1.2 Kết hợp đặc trưng . . . 2
2.1.3 Homography . . . 2
2.1.4 RANSAC . . . 2
2.1.5 Image wrap và blending . . . 3
2.2 Hiện thực với Python . . . 3
2.2.1 Detect và describe image . . . 3
2.2.2 Match keypoints . . . 4
2.2.3 Draw matches . . . 4
2.2.4 Stitch . . . 4
2.2.5 Run . . . 5

3 Kết quả đạt được và tổng kết **6**
3.1 Kết quả . . . 6
3.1.1 Bộ ảnh sử dụng . . . 6
3.1.2 Use SIFT . . . 6
3.1.2.1 Use ORB . . . 7
3.2 Tổng kết . . . 8
3.3 Tài liệu tham khảo . . . 9
3.4 Phụ lục . . . 10

Danh sách hình vẽ

3.1 Image for stitching 6
3.2 Keypoints 6
3.3 Matches 7
3.4 Result 7
3.5 Keypoints 7
3.6 Matches 7
3.7 Result 8

Danh sách bảng

Chương 1

Giới thiệu

1.1 Khái niệm về Image Stitching

Image stitching, hay còn gọi là ghép ảnh, là một kỹ thuật quan trọng trong lĩnh vực thị giác máy tính, cho phép kết hợp nhiều ảnh lại với nhau để tạo nên một ảnh lớn hơn, thường được gọi là ảnh panorama. Quá trình này bao gồm việc phát hiện các đặc điểm nổi bật trong từng ảnh, tìm kiếm sự tương ứng giữa các đặc điểm này qua các ảnh khác nhau, và cuối cùng là ghép chúng lại một cách mượt mà, đảm bảo tính liền mạch và chất lượng hình ảnh. Các bước chính trong quá trình ghép ảnh bao gồm phát hiện đặc điểm (feature detection), ghép nối đặc điểm (feature matching), ước lượng biến đổi (transformation estimation), biến dạng ảnh (image warping), và pha trộn (blending). Các thuật toán phổ biến cho việc phát hiện đặc điểm bao gồm SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), và ORB (Oriented FAST and Rotated BRIEF). Trong đó, SIFT là một trong những thuật toán đầu tiên và được sử dụng rộng rãi nhất, giúp phát hiện các điểm đặc biệt trong ảnh, không bị ảnh hưởng bởi sự thay đổi về tỷ lệ, xoay, ánh sáng, và góc nhìn. Mỗi đặc điểm được phát hiện sẽ có một mô tả (descriptor) riêng, là một biểu diễn số học ngắn gọn của thông tin hình ảnh xung quanh điểm đặc điểm đó, giúp cho việc ghép nối sau này. Ghép nối đặc điểm là bước tiếp theo, nơi mà các mô tả từ một ảnh được so sánh với các mô tả từ ảnh khác để xác định các cặp đặc điểm tương ứng. Các thuật toán thường được sử dụng cho nhiệm vụ này bao gồm Brute Force Matching và FLANN (Fast Library for Approximate Nearest Neighbors). Sau khi các cặp đặc điểm tương ứng đã được xác định, bước tiếp theo là ước lượng biến đổi, nơi mà các phép biến đổi hình học được tính toán để ghép các ảnh lại với nhau một cách chính xác nhất. Biến dạng ảnh là quá trình điều chỉnh các ảnh để chúng phù hợp với nhau về mặt hình học, và cuối cùng là pha trộn, nơi mà các ảnh được ghép lại một cách mượt mà để không có đường nối rõ ràng giữa chúng.

Kỹ thuật ghép ảnh có nhiều ứng dụng trong các lĩnh vực khác nhau, từ nhiếp ảnh, quảng cáo, đến giám sát an ninh và y tế. Ví dụ, trong nhiếp ảnh, ghép ảnh cho phép tạo ra các bức ảnh panorama có độ phân giải cao và quang cảnh rộng lớn mà không cần đến camera chuyên nghiệp. Trong quảng cáo, ghép ảnh giúp tạo ra các hình ảnh ấn tượng và hấp dẫn. Trong lĩnh vực giám sát an ninh, ghép ảnh được sử dụng để tạo ra các bức ảnh tổng quan từ nhiều camera khác nhau. Trong y tế, ghép ảnh giúp các bác sĩ có cái nhìn tổng quan hơn về cơ thể bệnh nhân từ nhiều hình ảnh chụp cắt lớp. Như vậy, image stitching không chỉ là một công cụ mạnh mẽ trong thị giác máy tính mà còn là một phần không thể thiếu trong nhiều ứng dụng thực tế..

Chương 2

Hiện thực

2.1 Cơ sở lý thuyết

Các bước cơ bản trong image stitching bao gồm phát hiện đặc trưng (Keypoint detection), mô tả(Descriptors), kết hợp đặc trưng (feature matching), Homography, RANSAC, biến dạng ảnh (image warping), và cuối cùng là pha trộn ảnh (blending).

2.1.1 Phát hiện đặc trưng

Chúng ta đã quen thuộc với các Feature extractor như SIFT, ORB. Trong đó, thuật toán ra đời khá sớm là SIFT - Scale Invariant Feature Transform. SIFT được dùng để detect các Keypoint trong ảnh. Những keypoint này là những điểm đặc biệt, giàu tính năng và đặc trưng. Với từng Keypoint, SIFT trả về tọa độ (x,y) kèm Descriptor - vector 128 chiều đại diện cho các đặc trưng của Keypoint đó.

Các Descriptor vector này không/ít bị ảnh hưởng bởi độ xoay, ánh sáng, scale... Điều đó có nghĩa cùng 1 điểm xuất hiện trên 2 ảnh (dù góc chụp, độ sáng khác nhau) sẽ có descriptor xấp xỉ nhau.

2.1.2 Kết hợp đặc trưng

Giả sử ta đã extract được 2 tập keypoint trên 2 ảnh là $S = \{k_1, k_2, \ldots, k_n\}$ và $S' = \{k'_1, k'_2, \ldots, k'_m\}$

Ta cần tiến hành so khớp keypoint trong 2 tập trên với nhau, để tìm ra các cặp keypoint tương ứng trên 2 ảnh. Chúng ta thường sử dụng khoảng cách Euclid giữa 2 Desciptor của 2 keypoint để đo độ sai khác giữa 2 keypoints đó. Chúng ta có thể sử dụng thuật toán FLANN maching hoặc Bruce Force Maching (tính toán vét cạn - nghĩa là ta phải tính Euclid distance giữa 1 keypoint trong S với tất các điểm trong tập còn lại, từ đó tìm ra các cặp điểm match nhau nhất giữa 2 tập). FLANN Matching (Fast Library for Approximate Nearest Neighbors) là sự cải tiến về tốc độ và hiệu năng cao, nhanh hơn BF matching nhờ vào việc chỉ tinh toán những điểm đủ tốt chứ không cần tìm điểm tốt nhất.

2.1.3 Homography

Như đã thực hiện ở Project 4, các phép biến đổi như tịnh tiến, xoay, bóp méo,... được thực hiện bằng cách nhân ma trận trên hệ tọa độ đồng nhất. Nên chúng ta sẽ không nhắc lại cách thức thực hiện ở Project lần này.

2.1.4 RANSAC

Sau khi thực hiện bước 3, ta thực hiện RANSAC - Random Sample Consesus. Đây là một thuật toán đơn giản, lấy mẫu bất kì 4 cặp điểm ngẫu nhiên, tính ma trận H. Tính độ sai khác giữa các điểm target và các điểm input sau khi biến đổi bằng H. Sử dụng công thức:

$$\text{Loss} = -\sum_{i=0}^{N} (\text{distance}(H * k_i, k_i`))$$

Trong đó, k_i và k_i‘ là 1 cặp điểm tương ứng. Lặp lại quá trình lấy mẫu - tính loss này với sô lần đủ lớn. Sau đó chọn H có Loss bé nhất. Như vậy ta đã thu được Homography matrix H dùng để biến đổi tọa độ các điểm trong ảnh input sang ảnh target. Opencv có cung cấp cho ta hàm để ước lượng matrix H

2.1.5 Image wrap và blending

Để làm cho ảnh output được mượt mà hơn, chúng ta có thể wrap lại ảnh để vừa với kích thước mong muốn hoặc bleding màu lại cho ảnh vì giữa giao nhau từ hai ảnh nguồn có thể có sự sai khác trong sắc độ của màu. Tuy nhiên, hai nội dung này sẽ không được thực hiện ở bài tập lớn này.

2.2 Hiện thực với Python

Toàn bộ source code có thể tìm thấy tại đây: Google colab

2.2.1 Detect và describe image

```
import cv2
import numpy as np


def detectAndDescribe(image, feature):
  # convert the image to gr
  image = np.array(image)
  kp= des = None
  if feature == "sift":
    sif = cv2.xfeatures2d.SIFT_create()
    (kp, des) = sif.detectAndCompute(image, None)

  elif feature == "orb":
    orb = cv2.ORB_create()
    kp = orb.detect(image,None)
    kp, des = orb.compute(image, kp)
  else:
    print("Feature not supported")
    return None

  kps = np.float64([k.pt for k in kp])
    # Draw keypoints on the image
  img_with_kps = cv2.drawKeypoints(image, kp, None, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

  return (kps, des, img_with_kps)
```

2.2.2 Match keypoints

```
def matchKeypoints(kpsA, kpsB, featuresA, featuresB, ratio, reprojThresh):
  # compute the raw matches and initialize the list of actual
  # matches
  #matcher = cv2.BFMatcher()
  matcher = cv2.DescriptorMatcher_create("BruteForce")

  rawMatches = matcher.knnMatch(featuresA, featuresB, 2)
  matches = []
  for m in rawMatches:
    # ensure the distance is within a certain ratio of each
    # other (i.e. Grant ratio test)
    if len(m) == 2 and m[0].distance < m[1].distance * ratio:
      matches.append((m[0].trainIdx, m[0].queryIdx))
      # computing a homography requires at least 4 matches
  if len(matches) > reprojThresh:
    # construct the two sets of points
    ptsA = np.float32([kpsA[i] for (_, i) in matches])
    ptsB = np.float32([kpsB[i] for (i, _) in matches])
    # compute the homography between the two sets of points
    (H, status) = cv2.findHomography(ptsA, ptsB, cv2.RANSAC, reprojThresh)
    # return the matches along with the homograpy matrix
    # and status of each matched point
    return (matches, H, status)
  return None
```

2.2.3 Draw matches

```
def drawMatches(imgA, kpsA, imgB, kpsB, matches, status):
  # initialize the output visualization image
  (hA, wA) = imgA.shape[:2]
  (hB, wB) = imgB.shape[:2]
  vis = np.zeros((max(hA, hB), wA + wB, 3), dtype="uint8")
  vis[0:hA, 0:wA] = imgA
  vis[0:hB, wA:] = imgB
  # loop over the matches
  for ((trainIdx, queryIdx), s) in zip(matches, status):
    # only process the match if the keypoint was successfully
    # matched
    if s == 1:
      # draw the match
      ptA = (int(kpsA[queryIdx][0]), int(kpsA[queryIdx][1]))
      ptB = (int(kpsB[trainIdx][0]) + wA, int(kpsB[trainIdx][1]))
      cv2.line(vis, ptA, ptB, (0, 255, 0), 1)  # return the visualization
  return vis
```

2.2.4 Stitch

```
def stitch(img_list, radio = 0.75, reprojThresh = 4.0, showMatches = False, feature='sift'):
  # Detect and describe keypoints in the two images
  kpsA, featureA, img_with_kpA = detectAndDescribe(img_list[0], feature)
  kpsB, featureB, img_with_kpB = detectAndDescribe(img_list[1], feature)
  # Match features between the two images
  M = matchKeypoints(kpsA, kpsB, featureA, featureB, radio, reprojThresh)
  # If the match is not None, unpack the matched keypoint
  # together with the homography
```

```
  if M is not None:
    (matches, H, status) = M
    # Use homography to warp the second image
    result = cv2.warpPerspective(img_list[0], H, (img_list[0].shape[1] + img_list[1].shape[1], img_list[0].shape[0]))
    result[0:img_list[1].shape[0], 0:img_list[1].shape[1]] = img_list[1]
    # If the showMatches flag is set, draw
    # the matched keypoints
    if showMatches:
      vis = drawMatches(img_list[0], kpsA, img_list[1], kpsB, matches, status)
      return (result, vis, img_with_kpA, img_with_kpB)
    return result
  return None
```

2.2.5 Run

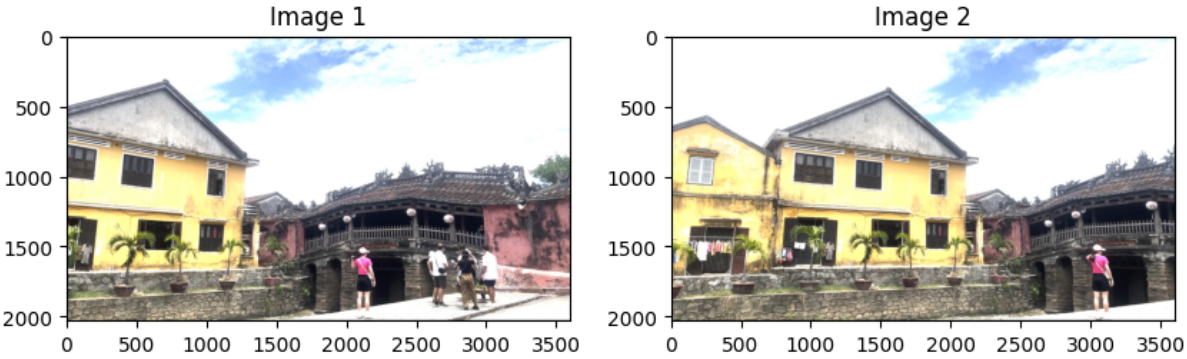
```
import imutils
import cv2

# Convert each PIL image to a NumPy array before resizing
img_array_list = [np.array(img) for img in img_list]

# Resize the images using imutils
resized_img_list = [imutils.resize(img, width=400) for img in img_array_list]
(result, vis, a, b) = stitch(resized_img_list,radio=0.75,reprojThresh = 4, showMatches=True, feature='sift')
print_two_images(a, b,'image A with keypoints', 'image B with keypoints')
show_img(vis)
show_img(result)
```

Chương 3

Kết quả đạt được và tổng kết

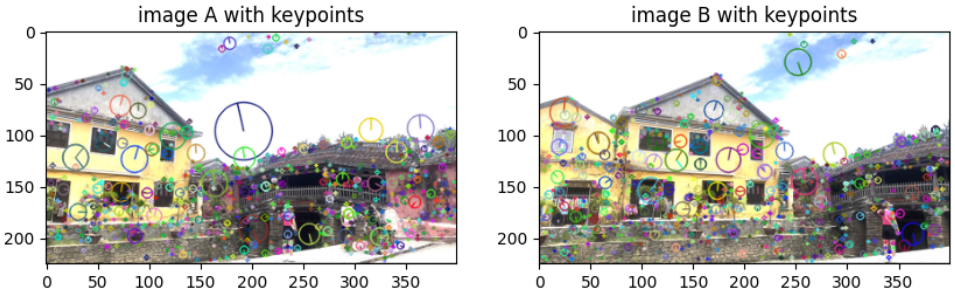
3.1 Kết quả

3.1.1 Bộ ảnh sử dụng

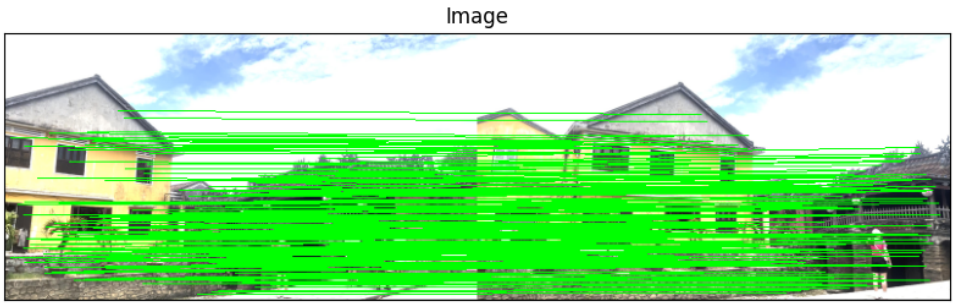


Hình 3.1: *Image for stitching*

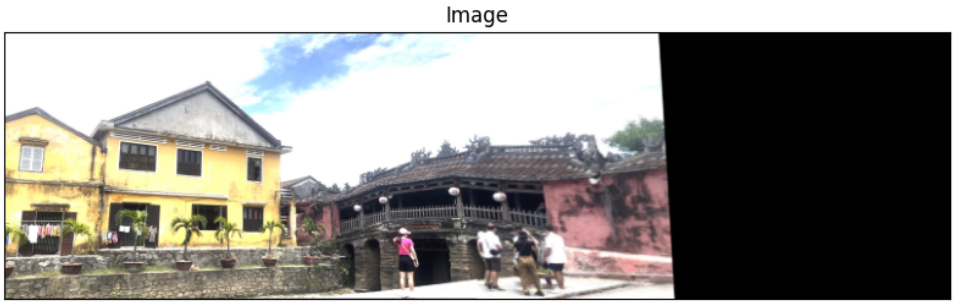
3.1.2 Use SIFT



Hình 3.2: *Keypoints*

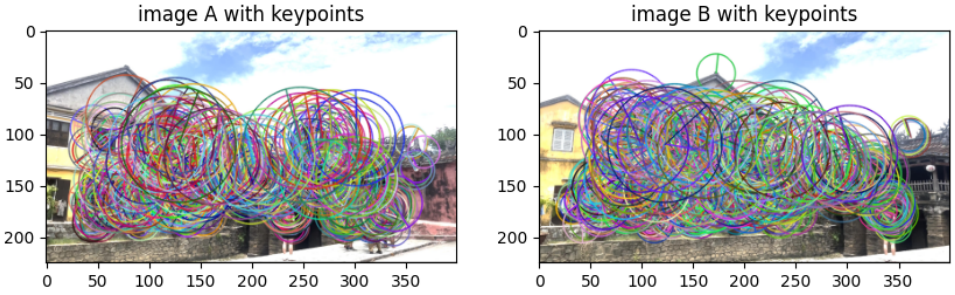


Hình 3.3: *Matches*

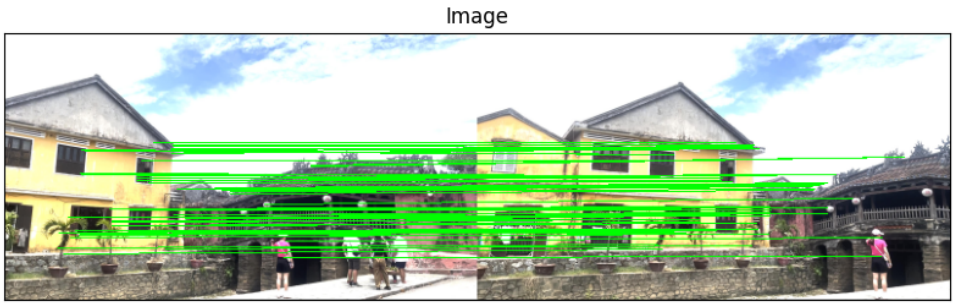


Hình 3.4: *Result*

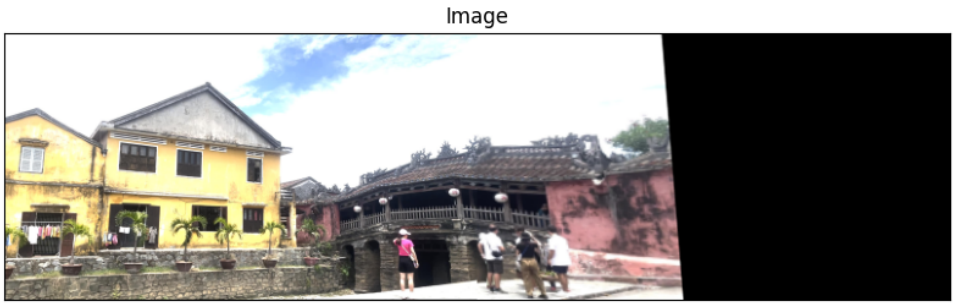
3.1.2.1 Use ORB



Hình 3.5: *Keypoints*



Hình 3.6: *Matches*



Hình 3.7: *Result*

3.2 Tổng kết

Trong quá trình hoàn thành dự án ghép ảnh sử dụng hai phương pháp trích xuất đặc trưng SIFT và ORB, chúng ta đã thu được nhiều bài học quý giá. Đầu tiên, SIFT, với khả năng chống biến đổi hình học và chiếu sáng, đã chứng minh là một công cụ mạnh mẽ trong việc duy trì tính nhất quán của đặc trưng qua các ảnh khác nhau. Mặt khác, ORB, với tốc độ xử lý nhanh hơn và chi phí tính toán thấp hơn, đã cung cấp một lựa chọn hiệu quả về mặt thời gian và tài nguyên. Sự kết hợp của hai phương pháp này đã tạo ra một hệ thống ghép ảnh linh hoạt, có khả năng thích ứng với nhiều loại dữ liệu và điều kiện khác nhau.

Tuy nhiên, không có phương pháp nào là hoàn hảo. SIFT có thể không hiệu quả trong các tình huống có sự thay đổi độ sáng mạnh mẽ, trong khi ORB có thể không duy trì được tính chất đặc trưng trong các trường hợp có biến đổi hình học lớn. Do đó, việc lựa chọn phương pháp phù hợp dựa trên bản chất của dữ liệu và yêu cầu cụ thể của dự án là rất quan trọng.

Cuối cùng, dự án này không chỉ cung cấp một giải pháp kỹ thuật cho việc ghép ảnh mà còn mở ra cơ hội để hiểu sâu hơn về các thuật toán trích xuất đặc trưng và ứng dụng của chúng. Bài học lớn nhất có lẽ là sự cần thiết của việc thử nghiệm và đánh giá liên tục, điều chỉnh phương pháp để phù hợp với từng tình huống cụ thể, và luôn sẵn lòng tiếp nhận những phát hiện mới có thể cải thiện kết quả cuối cùng. Đây là một bước tiến quan trọng trong lĩnh vực xử lý ảnh, và chắc chắn sẽ có ảnh hưởng lâu dài đến các nghiên cứu và ứng dụng trong tương lai.

3.3 Tài liệu tham khảo

1. Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision* (2nd Edition). Cambridge University Press, 2004.
2. Szeliski, Richard. *Computer Vision: Algorithms and Applications*. Springer, 2010.
3. Ma, Yi, Stefano Soatto, Jana Kosecka, and S. Shankar Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. Springer, 2004.
4. Gonzalez, Rafael C., and Richard E. Woods. *Digital Image Processing* (4th Edition). Pearson, 2022.
5. Sonka, Milan, Vaclav Hlavac, and Roger Boyle. *Image Processing, Analysis, and Machine Vision* (4th Edition). Cengage Learning, 2019.
6. Fattal, Raanan; Lischinski, Dani; Werman, Michael (2002). "Gradient domain high dynamic range compression" (PDF). Proceedings of the 29th annual conference on Computer graphics and interactive techniques - SIGGRAPH '02. p. 249. doi:10.1145/566570.566573. ISBN 1581135211. S2CID 1650337.

3.4 Phụ lục