24-678: Computer Vision for Engineers

Carnegie Mellon University

PS4

Due: **10/4/2024 (Fri) 5 PM @ Gradescope**

Issued: **9/25/2024 (Wed)**

Weight: **5% of total grade**

Note:

PS4-1 Image Mosaicing with Bi-linear Transformation

Multiple images can be merged into a single image by image Mosaicing. The process consists of three steps:

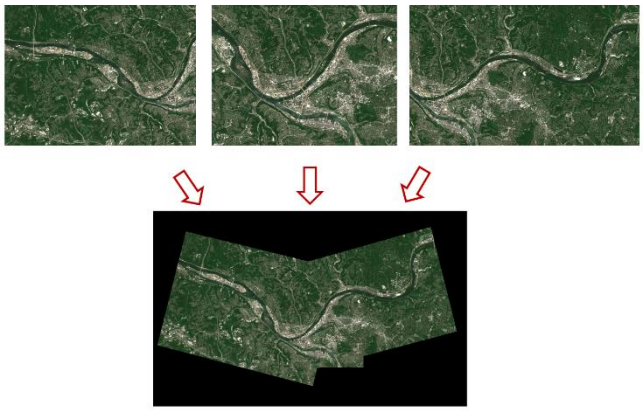
(1) detecting keypoints, and matching them,

(2) identifying bi-linear geometric transformations that match feature points, and

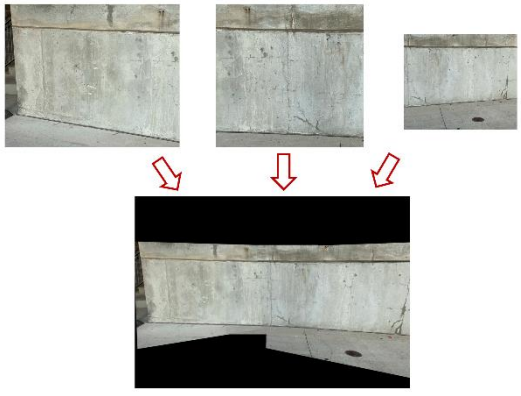
(3) blending transformed images together into a single image.

Image Mosaicing is useful for many practical tasks including: stitching up multiple aerial images of a city into a single map (see Figure 1a), and stitching up multiple images taken from a drone into a single image (see Figure 1b).

In this problem set, you are given a functioning demo code for stitching two images together. Your task is to understand how the code works and extend it to stitch three images.



(a) aerial images for generating a map



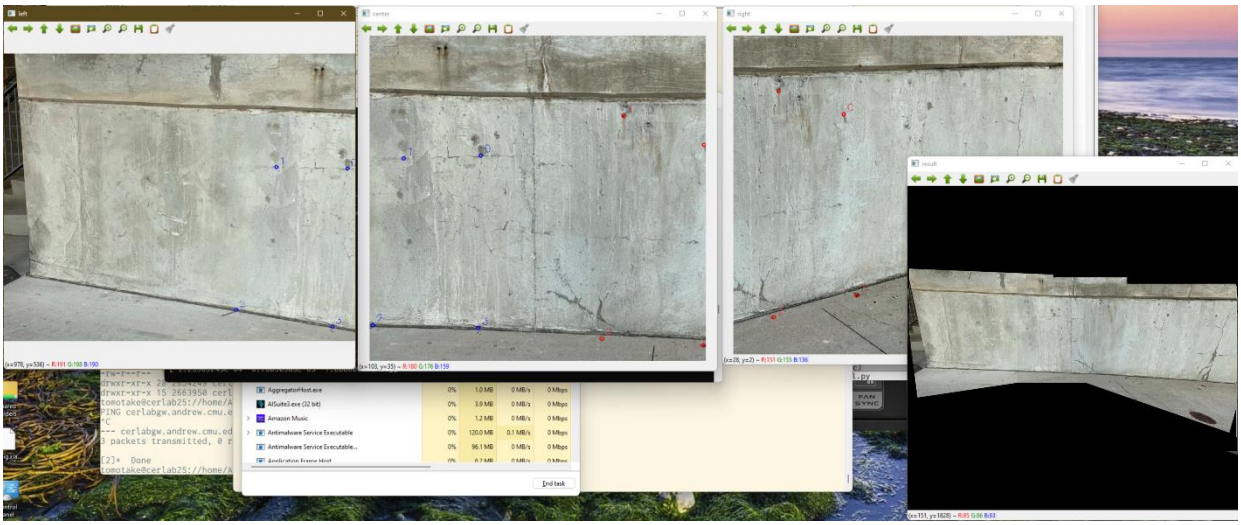
(b) drone images for detecting cracks in concrete

Figure 1: Applications of Image Mosaicing

(a) input image - left

(b) input image - center

(c) input image - right



(d) specified keypoints and their correspondences

Figure 2: Input images, feature points, and output stitched image

Your final code should:

(1) read three images, image-left, image-center, and image-right (see Figures 2a, 2b, and 2c),

(2) prompt the user to specify four keypoints in the right image – the ones that you use for stitching the right and center images,

(3) prompt the user to specify four corresponding keypoints in the center image,

(4) prompt the user to specify four keypoints in the left image – the ones that you use for stitching the left and center images,

(5) prompt the user to specify four corresponding keypoints in the center image,

(6) stitch the three images into a single image using the Affine transformation, as shown in Figure 2d, and

(7) save the stitched image, image-stitched, as an image file.

Apply your code to four sets of images shown in Figure 3: pittsburgh, wall, house, and door.

Submission

To prepare for the submission of your work on Gradescope, create:

(1) a folder called “ps4-1,” that contains the following files:

- source code file(s)
- improved images created by your program:
  - pittsburgh-stitched.png
  - wall-stitched.png
  - house-stitched.png
  - door-stitched.png
- "readme.txt" file that includes:
  - Operating system
  - IDE you used to write and run your code
  - The number of hours you spent to finish this problem

(2) a PDF file that contains the printouts and screenshots of all the files in the ps4-1 folder. (Include, if any, the mathematical derivation and/or description of your method in the PDF file. Handwritten notes should be scanned and included in the PDF file.)

Submit your work on Gradescope

Submit two files on Gradescope – replace “andrewid” with your own Andrew ID:

(1) **andrewid-ps4-files.zip** – this ZIP file should contain the ps4-1 folder and all the files requested.

(2) **andrewid-ps4-report.pdf** – this PDF file serves as the report of your work, and it should contain the printouts and screenshots of all the files in the “ps4-1” folder. (Include, if any, the mathematical derivation and/or description of your method in the PDF file. Handwritten notes should be scanned and included in the PDF file.)

Please organize pages with section titles and captions to make the report easy to read.

(a) pittsburgh-left.jpg, pittsburgh-center.jpg, pittsburgh-right.jpg

(b) wall-left.png, wall-center.png, wall-right.png

(c) house-left.jpg, house-center.jpg, house-right.jpg

(d) door-left.jpg, door-center.jpg, door-right.jpg

Figure 3. Images to be stitched together

Appendix: Code for Image Mosaicing Two Images


```
# import the necessary packages
import cv2
import numpy as np
import sys
import json
import argparse
def savePick():
    global pick
    data = {}
    data["pick"] = pick
    with open('result.json', 'w') as outfile:
        json.dump(data, outfile)
def loadPick():
    global pick
    with open('result.json') as file:
        data = json.load(file)
    pick = data["pick"]
    print(pick)
def combine():
    global result, imageC, imageL, imageR, pick
    (h,w) = imageC.shape[:2]
    cng = cv2.cvtColor(result, cv2.COLOR_BGR2GRAY)
    th, mask_c = cv2.threshold(cng, 1, 255, cv2.THRESH_BINARY)
    mask_c = mask_c / 255
    # right
    src_pnts = np.empty([4,2], np.float32)
    dst_pnts = np.empty([4,2], np.float32)
    for i in range(4):
        src_pnts[i][0] = float(pick[0][i][0])
        src_pnts[i][1] = float(pick[0][i][1])
        dst_pnts[i][0] = float(pick[1][i][0]+w)
        dst_pnts[i][1] = float(pick[1][i][1]+h)
    M = cv2.getPerspectiveTransform(src_pnts, dst_pnts)
    rn = cv2.warpPerspective(imageR, M, (w*3,h*3))
    rng = cv2.cvtColor(rn, cv2.COLOR_BGR2GRAY)
    th, mask_r = cv2.threshold(rng, 1, 255, cv2.THRESH_BINARY)
    #cv2.imwrite("mask_r.png", mask_r)
    mask_r = mask_r / 255

    # left
    M = np.identity(3)
    ln = cv2.warpPerspective(imageL, M, (w*3,h*3))
    lng = cv2.cvtColor(ln, cv2.COLOR_BGR2GRAY)
    th, mask_l = cv2.threshold(lng, 1, 255, cv2.THRESH_BINARY)
    mask_l = mask_l / 255
    #cv2.imwrite("mask_l.png", mask_l)
    # alpha blending
    # mask element: number of pictures at that coordinate
    mask = np.array(mask_c + mask_l + mask_r, float)
    # alpha weight
    ag = np.full(mask.shape, 0.0, dtype=float)
    # weight: 1.0 / (num of picture)
    ag = 1.0 / np.maximum(1,mask) # avoid 0 division
    # generate result image from 3 images + alpha weight
    result[:,:,0] = result[:,:,0]*ag[:,:] + ln[:,:,0]*ag[:,:] + rn[:,:,0]*ag[:,:]
    result[:,:,1] = result[:,:,1]*ag[:,:] + ln[:,:,1]*ag[:,:] + rn[:,:,1]*ag[:,:]
    result[:,:,2] = result[:,:,2]*ag[:,:] + ln[:,:,2]*ag[:,:] + rn[:,:,2]*ag[:,:]
    cv2.imwrite("result.jpg", result)
    cv2.imshow("result", result)
'''
pick 4 points from right image (red point)
```

```
'''
def right_click(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONUP:
        mousePick(x, y, 0)
'''
pick 4 points from center (correspond to right, red point)
'''
def center_click_r(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONUP:
        mousePick(x, y, 1)
'''
pick 4 points from left (blue point)
'''
def left_click(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONUP:
        mousePick(x, y, 2)
'''
pick 4 points from center (correspond to left, blue point)
'''
def center_click_l(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONUP:
        mousePick(x, y, 3)
def mousePick(x, y, idx):
    global rn, cn, ln, imageR, imageC, imageL, pick
    if idx == 0:
        src = imageR
        dst = rn
        wn = "right"
    elif idx == 1 or idx == 3:
        src = imageC
        dst = cn
        wn = "center"
    elif idx == 2:
        src = imageL
        dst = ln
        wn = "left"
    #print(idx, x, y)
    pick[idx].append((x,y))
    dst = src.copy()
    if idx < 2:
        col = (0, 0, 255)
    else:
        col = (255, 0, 0)
    # place circle on the picked point and text its serial (0-3)
    for i in range(len(pick[idx])):
        dst = cv2.circle(dst, pick[idx][i], 5, col, 2)
        dst = cv2.putText(dst, str(i), (pick[idx][i][0]+10, pick[idx][i][1]-10),
                          cv2.FONT_HERSHEY_SIMPLEX,1, col, 1)
    cv2.imshow(wn, dst)
    # to make sure image is update
    cv2.waitKey(1)
    if len(pick[idx]) >= 4:
        print('Is it OK? (y/n)')
        i = input()
        if i == 'y' or i == 'Y':
            if idx >= 3:
                savePick()
                combine()
            elif idx == 0:
                print('center 4 points')
                cv2.setMouseCallback("center", center_click_r)
            elif idx == 1:
                print('left 4 points')
                cv2.setMouseCallback("left", left_click)
            elif idx == 2:
                print('center 4 points')
```

```
                cv2.setMouseCallback("center", center_click_l)
        else:
            pick[idx] = []
            dst = src.copy()
            cv2.imshow(wn, dst)
parser = argparse.ArgumentParser(description='Combine 3 images')
parser.add_argument('-d', '--data', type=int, help='Dataset index', default=3)
args = parser.parse_args()
dataset = args.data
if dataset == 0:
    imageL = cv2.imread("wall-left.png")
    imageC = cv2.imread("wall-center.png")
    imageR = cv2.imread("wall-right.png")
elif dataset == 1:
    imageL = cv2.imread("door-left.jpg")
    imageC = cv2.imread("door-center.jpg")
    imageR = cv2.imread("door-right.jpg")
elif dataset == 2:
    imageL = cv2.imread("house-left.jpg")
    imageC = cv2.imread("house-center.jpg")
    imageR = cv2.imread("house-right.jpg")
else:
    imageL = cv2.imread("pittsburgh-left.jpg")
    imageC = cv2.imread("pittsburgh-center.jpg")
    imageR = cv2.imread("pittsburgh-right.jpg")
result =
cv2.copyMakeBorder(imageC,imageC.shape[0],imageC.shape[0],imageC.shape[1],imageC.shape[1],
                           borderType=cv2.BORDER_CONSTANT,value=[0,0,0])
print(imageL.shape,imageC.shape,imageR.shape, result.shape)
cv2.namedWindow("left",cv2.WINDOW_NORMAL)
cv2.namedWindow("center",cv2.WINDOW_NORMAL)
cv2.namedWindow("right",cv2.WINDOW_NORMAL)
cv2.namedWindow("result",cv2.WINDOW_NORMAL)
ln = imageL.copy()
cn = imageC.copy()
rn = imageR.copy()
cv2.imshow("left", ln)
cv2.imshow("center", cn)
cv2.imshow("right", rn)
cv2.imshow("result", result)
pick = []
pick.append([])
pick.append([])
pick.append([])
pick.append([])
print('use saved points? (y/n)')
i = input()
if i == 'y' or i == 'Y':
    loadPick()
    combine()
else:
    print("right 4 points")
    cv2.setMouseCallback("right", right_click)
cv2.waitKey()
# close all open windows
cv2.destroyAllWindows()
```