Pa3_Bonus

April 22, 2024

1 EE5178: Panoramic Stitching

2 Brief Overview

In this problem set, you will implement panoramic stitching. Given two input images, we will "stitch" them together to create a simple panorama. To construct the image panorama, we will use concepts learned in class such as keypoint detection, local invariant descriptors, RANSAC, and perspective warping.

The panoramic stitching algorithm consists of four main steps which we ask you to implement in individual functions:

1. Detect keypoints and extract local invariant descriptors from two input images.
2. Match the descriptors between the two images.
3. Apply RANSAC to estimate a homography matrix between the extracted features.
4. Apply a perspective transformation using the homography matrix to merge image into a panorama.

Functions to implement (refer to function comments for more detail):

1. get_features
2. match_keypoints
3. find_homography and transform_ransac
4. panoramic_stitching

3 Starting

Run the following code to import the modules you'll need. After your finish the assignment, remember to run all cells and save the note book to your local machine as a .ipynb file for Canvas submission.

[1]:
```
%matplotlib inline
import cv2
import random
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

[2]:
```
%%capture
! wget -O img1.jpg "https://drive.google.com/uc?
↪export=download&id=1omMydL6ADxq_vW5gl_1EFhdzT9kaMhUt"
! wget -O img2.jpg "https://drive.google.com/uc?
↪export=download&id=12lxB1ArAlwGn97XgBgt-SFyjE7udMGvf"
```

4 Visualize Input Images

[3]:
```
img1 = plt.imread('img1.jpg')
img2 = plt.imread('img2.jpg')

def plot_imgs(img1, img2):
  fig, ax = plt.subplots(1, 2, figsize=(15, 20))
  for a in ax:
    a.set_axis_off()
  ax[0].imshow(img1)
  ax[1].imshow(img2)

plot_imgs(img1, img2)
```

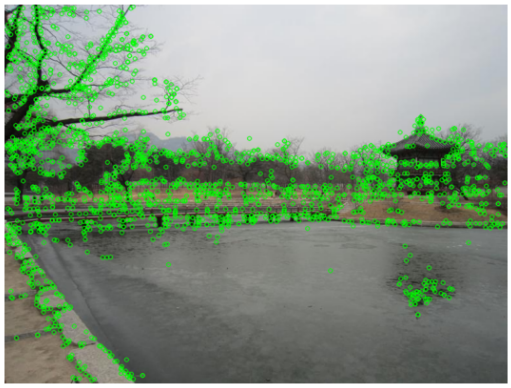
5 Compute SURF/ SIFT/ ORB Features and Match Keypoints

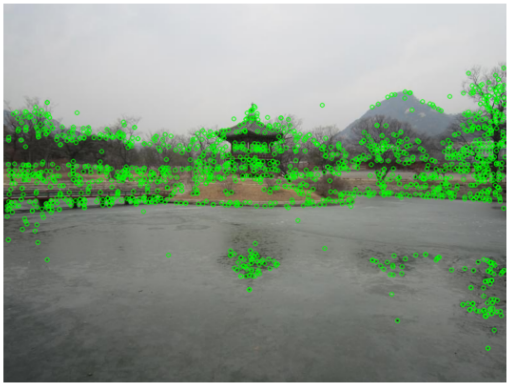
[4]:
```
def get_features(img):
  '''
    Compute SURF/SIFT/ORB features using cv2 library functions. Use default␣
↪parameters when computing the keypoints.
    Input:
      img: cv2 image
    Returns:
      keypoints: a list of cv2 keypoints
      descriptors: a list of feature descriptors
```

```
  '''
  gray_image = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

  sift = cv2.SIFT_create()
  keypoints = sift.detect(gray_image,None)
  keypoints,descriptors = sift.compute(gray_image,keypoints)

  return keypoints, descriptors

def match_keypoints(desc_1, desc_2, ratio=0.75):
  '''
    You may use cv2 library functions.
    Input:
      desc_1, desc_2: list of feature descriptors
    Return:
      matches: list of feature matches
  '''
  # ================================================
  # TODO
  # Create BFMatcher object
  bf = cv2.BFMatcher()

  # using KNN
  matches = bf.knnMatch(desc_1,desc_2,k=2)

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

  # ================================================
  return good_matches
```

[5]:
```
kp_1, desc_1 = get_features(img1)
kp_2, desc_2 = get_features(img2)

kp_img1 = cv2.drawKeypoints(img1, kp_1, None, color=(0,255,0), flags=0)
kp_img2 = cv2.drawKeypoints(img2, kp_2, None, color=(0,255,0), flags=0)

print('keypoints for img1 and img2')
plot_imgs(kp_img1, kp_img2)
```

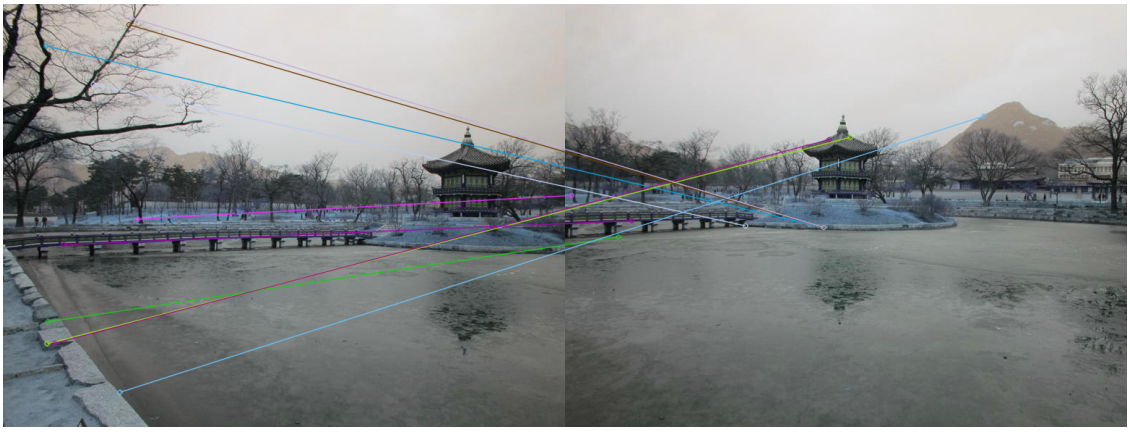
```
keypoints for img1 and img2
```





[201]:
```
matches = match_keypoints(desc_1, desc_2)

print(type(matches), len(matches))

# print(matches)

match_plot = cv2.drawMatches(img1, kp_1, img2, kp_2, matches[:10], None,␣
↪flags=2)

print("feature matches")
cv2_imshow(match_plot)
```

```
<class 'list'> 406
feature matches
```



6 Compute Homography Matrix using RANSAC

[132]:
```
def find_homography(pts_1, pts_2):
  '''
    Implement Direct Linear Transform to find a homography that estimates the␣
 ↪transformation mapping from pts_1 to pts_2.
    e.g. If x is in pts_1 and y is in pts_2, then y = H * x
    Input:
      pts_1, pts_1: (N, 2) matrix
    Return:
      H: the resultant homography matrix (3 x 3)
  '''
  # ================================================
  # TODO
  # ================================================

  # form the A - Matrix
  # pt1 - src
  # pt2 - dst
  A = []
  # H = np.array([])
  # print(len(pts_1))
  for (pt_s, pt_d) in zip(pts_1, pts_2):
    A.append([-pt_s[0],-pt_s[1],-1, 0,0,0, 1*pt_d[0]*pt_s[0],␣
 ↪1*pt_d[0]*pt_s[1], 1*pt_d[0]])
    A.append([0,0,0, -pt_s[0],-pt_s[1],-1 , 1*pt_d[1]*pt_s[0],␣
 ↪1*pt_d[1]*pt_s[1], 1*pt_d[1] ])

  # A- Matrix
  A = np.array(A)
  # print(A)

  # find the solution
  u, s, vh = np.linalg.svd(A)
  homography = vh[-1].reshape((3,3))

  homography /= homography[2,2] #scaled by the last element
  H = homography
  sums = np.sum(homography ** 2)
  # print("sum : ", sums)
  # print("H Matrix\n", H)
  return H
```

[133]:
```
def transform_ransac(pts_1, pts_2):
  '''
    Implement RANSAC to estimate homography matrix.
    Input:
```

```
    pts_1, pts_1: (N, 2) matrices
  Return:
    best_model: homography matrix with most inliers
'''
best_inliers = []
final_H = []
best_model = np.array([])

total_pts = np.concatenate((pts_1, pts_2), axis=1).reshape(-1,4)

for i in range(100): # can set to 5000

  random_kp = random.choices(total_pts, k=4)

  random_kp1 = [random_kp[0][:2], random_kp[1][:2], random_kp[2][:2],␣
↪random_kp[3][:2]]
  random_kp2 = [random_kp[0][2:], random_kp[1][2:], random_kp[2][2:],␣
↪random_kp[3][2:]]
  H = find_homography(random_kp1, random_kp2)
  inliers = []
  for src_pt, targ_pt in zip(pts_1, pts_2):
    p = np.array([src_pt[0], src_pt[1], 1]).reshape(3,1)
    p_1 = np.array([targ_pt[0], targ_pt[1], 1]).reshape(3,1)
    Hp = np.dot(H, p)
    Hp = Hp/Hp[2] #scale

    dist = np.linalg.norm(p_1 - Hp)
    # print(dist)

    if dist < 5:
      inliers.append([src_pt[0], src_pt[1],targ_pt[0], targ_pt[1]])

  # print(len(inliers), " , ", len(best_inliers))
  if len(inliers) > len(best_inliers):
    best_inliers = inliers
    best_model = H
    # print('updated H')

# print(H)
return best_model
```

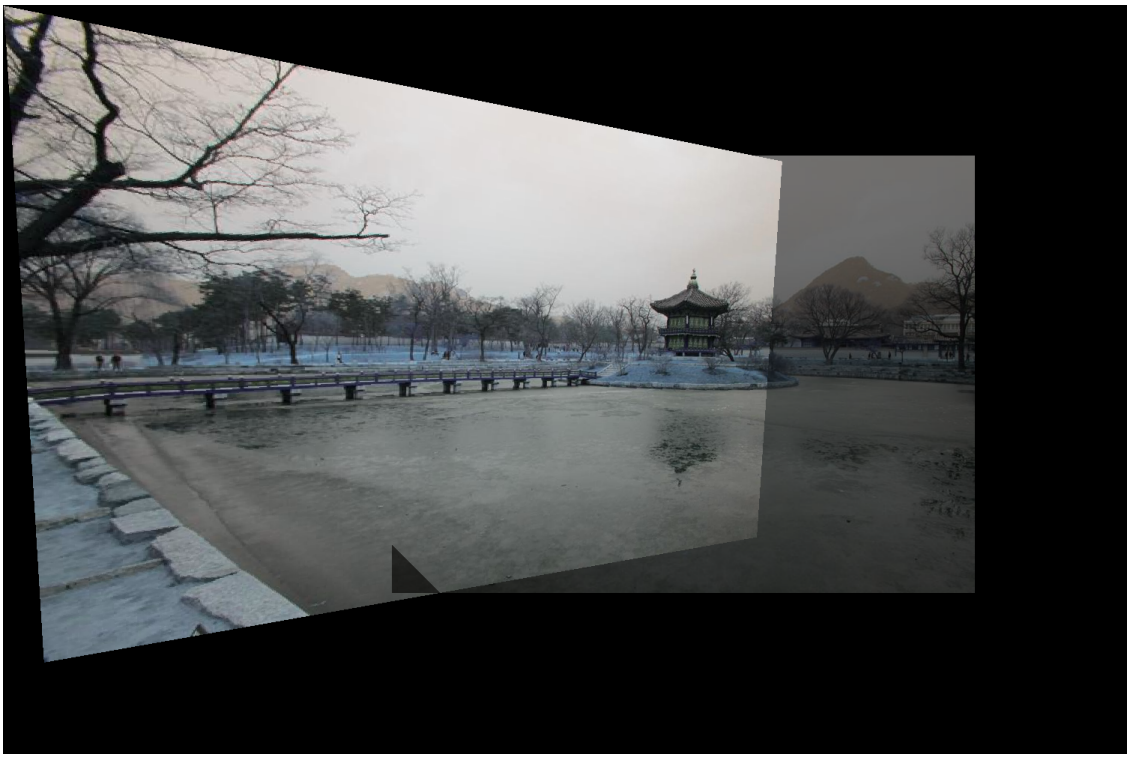
7 Panoramic Stitching

[195]:
```
import math
def panoramic_stitching(img1, img2):
  '''
    Generate a panoramic image using the obtained homography matrix.
    Input:
      img1, img2: cv2 images
    Return:
      final_img: cv2 image of panorama
  '''
  # ================================================

  kp_1, desc_1 = get_features(img1)
  kp_2, desc_2 = get_features(img2)

  matches = match_keypoints(desc_1, desc_2) # query, Train

  src_pts = np.float32([ kp_1[m.queryIdx].pt for m in matches ]).reshape(-1,2)
  dst_pts = np.float32([ kp_2[m.trainIdx].pt for m in matches ]).reshape(-1,2)

  # for i in range(len(src_pts)):
  #   # if (abs((src_pts[i][0] - dst_pts[i][0])) and (abs(src_pts[i][1] -
 ↪dst_pts[i][1]))) <= 1:
  #   print(" (x,y) : ", src_pts[i] ," \t Dest pts : ", dst_pts[i] )

  # print(dst_pts.shape, type(dst_pts))

  H = transform_ransac(src_pts, dst_pts)

  rows, cols, ch = img1.shape
  print(rows,cols)

  H_inv = np.linalg.inv(H)


  # find the initial Bounding Box for img1
  # Take the four corners of the image and then forward Transform
  x_bl, y_bl, k1 = np.matmul(H, np.array([0,0,1]).T)
  x_br, y_br, k2 = np.matmul(H, np.array([cols-1,0,1]).T)
  x_tr, y_tr, k3 = np.matmul(H, np.array([cols-1,rows-1,1]).T)
  x_tl, y_tl, k4 = np.matmul(H, np.array([0,rows-1,1]).T)

  x_bl, y_bl = x_bl/k1, y_bl/k1
  x_br, y_br = x_br/k2, y_br/k2
  x_tr, y_tr = x_tr/k3, y_tr/k3
  x_tl, y_tl = x_tl/k4, y_tl/k4
```

```
    #find the BBox for warped image 1
    x_min_img1 = math.floor(min(x_bl, x_br, x_tr, x_tl))
    y_min_img1 = math.floor(min(y_bl, y_br, y_tr, y_tl))
    x_max_img1 = math.ceil(max(x_bl, x_br, x_tr, x_tl))
    y_max_img1 = math.ceil(max(y_bl, y_br, y_tr, y_tl))

    # print("BBOx points : ", x_min_img1, y_min_img1, " , ", x_max_img1,␣
    ↪y_max_img1)

    x_len_img1 = x_max_img1 - x_min_img1 # Number of Cols
    y_len_img1 = y_max_img1 - y_min_img1 # Number of Rows

    img1_wrap = np.zeros((y_len_img1, x_len_img1, 3))
    # print("Min : ", x_min_img1, " , ",y_min_img1, "\nMax : ", x_max_img1, " ,␣
    ↪", y_max_img1)
    # print("Wrap image lens : ", x_len_img1, y_len_img1)
    # print(img1_wrap.shape, img1.shape)

    out_trial = np.zeros((2000,2000,3))
    # x - cols
    # y - rows

    for i in range(2000): # rows
      for j in range(2000): # cols
        if(0 <= i < 584) and (0 <= j < 778):
          out_trial[i+int(y_len_img1/4.7)+15][j+int(x_len_img1/2)] = img2[i][j]/2
          # out_trial[i+int(y_len_img1/4)][j+int(x_len_img1/2)] = img2[i][j]/2
        i_tr, j_tr, k_tr = np.matmul(H_inv, np.
    ↪array([i+x_min_img1,j+y_min_img1,1]).T)
        i_tr, j_tr = i_tr//k_tr, j_tr//k_tr
        # print(i_tr, ", ", j_tr)
        if((0 <= i_tr < cols) and (0 <= j_tr < rows)):
          out_trial[int(j),int(i),:] = img1[int(j_tr),int(i_tr),:]

    result_img = out_trial[0:1000, 0:1500,:]

    return result_img
```

[196]:
```
result = panoramic_stitching(img1, img2)
cv2_imshow(result)
```

```
584 778
```



[189]:
```
!pip install nbconvert
```

```
Requirement already satisfied: nbconvert in /usr/local/lib/python3.10/dist-
packages (6.5.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (4.9.4)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (4.12.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages
(from nbconvert) (6.1.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.4)
Requirement already satisfied: jinja2>=3.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (3.1.3)
Requirement already satisfied: jupyter-core>=4.7 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (5.7.2)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.1.5)
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.8.4)
```

```
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (0.10.0)
Requirement already satisfied: nbformat>=5.1 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.10.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (24.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (1.5.1)
Requirement already satisfied: pygments>=2.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert) (2.16.1)
Requirement already satisfied: tinycss2 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.0 in /usr/local/lib/python3.10/dist-
packages (from nbconvert) (5.7.1)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core>=4.7->nbconvert)
(4.2.0)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.10/dist-packages (from nbclient>=0.5.0->nbconvert)
(6.1.12)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert) (2.19.1)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat>=5.1->nbconvert) (4.19.2)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-
packages (from beautifulsoup4->nbconvert) (2.5)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-
packages (from bleach->nbconvert) (0.5.1)
Requirement already satisfied: attrs>=22.2.0 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (23.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from
jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.34.0)
Requirement already satisfied: rpds-py>=0.7.1 in /usr/local/lib/python3.10/dist-
packages (from jsonschema>=2.6->nbformat>=5.1->nbconvert) (0.18.0)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (23.2.1)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.10/dist-packages (from jupyter-
client>=6.1.12->nbclient>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: tornado>=4.1 in /usr/local/lib/python3.10/dist-
packages (from jupyter-client>=6.1.12->nbclient>=0.5.0->nbconvert) (6.3.3)
```

[197]:
```
!apt-get install texlive texlive-xetex texlive-latex-extra pandoc
```

```
Reading package lists… Done
Building dependency tree… Done
Reading state information… Done
pandoc is already the newest version (2.9.2.1-3ubuntu2).
texlive is already the newest version (2021.20220204-1).
texlive-latex-extra is already the newest version (2021.20220204-1).
texlive-xetex is already the newest version (2021.20220204-1).
0 upgraded, 0 newly installed, 0 to remove and 45 not upgraded.
```

[198]:
```
from google.colab import drive
drive.mount("/content/drive/")
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call
drive.mount("/content/drive/", force_remount=True).
```

[200]:
```
!jupyter nbconvert --to pdf /content/drive/MyDrive/"Colab Notebooks"/Pa3_Bonus.
 ↪ipynb
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/Pa3_Bonus.ipynb to pdf
[NbConvertApp] Support files will be in Pa3_Bonus_files/
[NbConvertApp] Making directory ./Pa3_Bonus_files
[NbConvertApp] Making directory ./Pa3_Bonus_files
[NbConvertApp] Making directory ./Pa3_Bonus_files
[NbConvertApp] Making directory ./Pa3_Bonus_files
[NbConvertApp] Writing 76146 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 2387823 bytes to /content/drive/MyDrive/Colab
Notebooks/Pa3_Bonus.pdf
```

[14]:
```

```