EECS 442 PS9: Panoramic Stitching

Calvin Tran, cktran

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 (we will be using ORB) 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_orb_features` (2 points)
2. `match_keypoints` (2 points)
3. `find_homography` (2 points)
4. `transform_ransac` (2 points)
5. `panoramic_stitching` (2 points)

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.

In [15]:

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

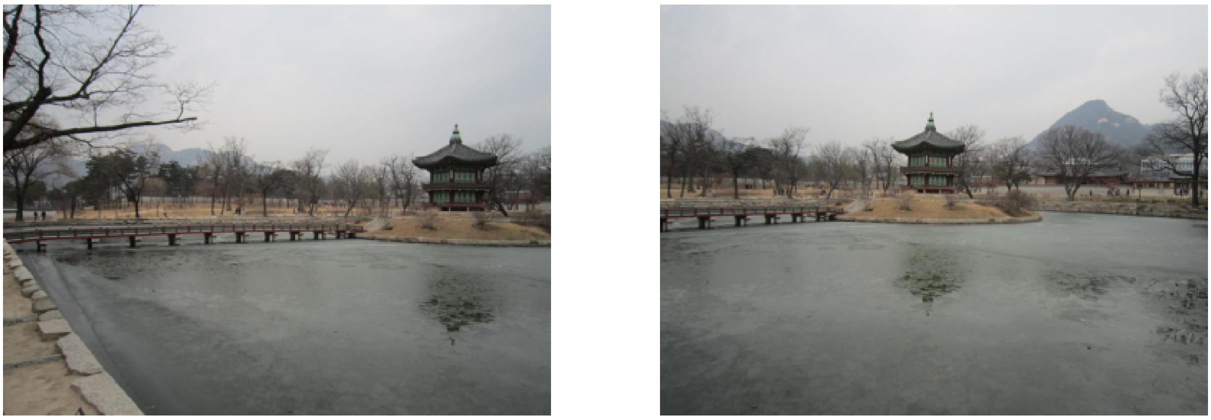
In [16]:

```
%%capture
! wget -O img1.jpg "https://drive.google.com/uc?export=download&id=1omMydL6ADxq_vW5g
! wget -O img2.jpg "https://drive.google.com/uc?export=download&id=12lxB1ArAlwGn97Xg
```

Visualize Input Images

In [17]:

```
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)
```



(a) Feature Extraction

(i) Compute ORB Features

In [18]:

```
def get_orb_features(img):
  '''
    Compute ORB features using cv2 library functions.
    Use default parameters when computing the keypoints.
    Hint: you will need cv2.ORB_create() and some related functions
    Input:
      img: cv2 image
    Returns:
      keypoints: a list of cv2 keypoints
      descriptors: a list of ORB descriptors
  '''
  #############################################################################
  #                                    TODO                                   #
  #############################################################################
  orb = cv2.ORB_create()
  keypoints, descriptors = orb.detectAndCompute(img, None)
  #############################################################################
  #                              END OF YOUR CODE                             #
  #############################################################################
  return keypoints, descriptors
```

(ii) Match Keypoints

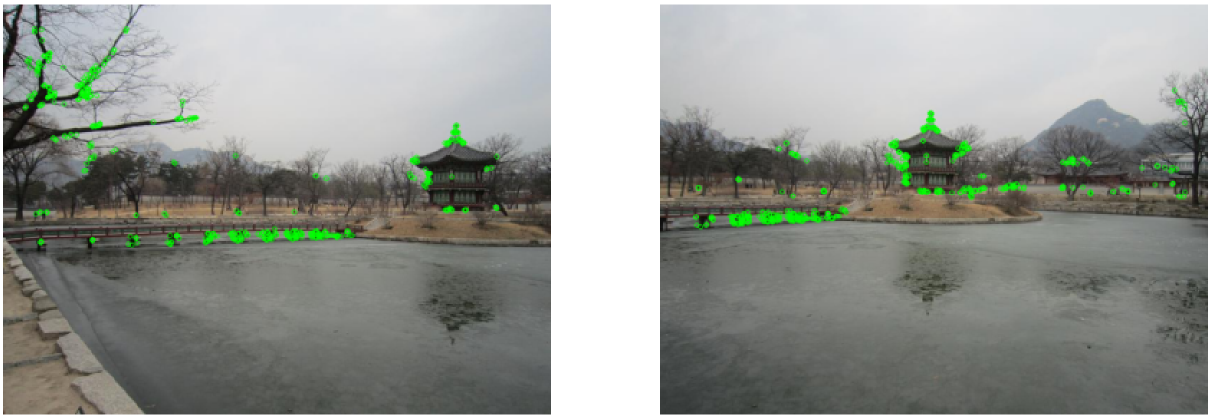
In [19]:

```
def match_keypoints(desc_1, desc_2, ratio=0.75):
  '''
    Compute matches between feature descriptors of two images using
    Lowe's ratio test. You may use cv2 library functions.
    Hint: you may need to use cv2.DescriptorMatcher_create or cv2.BFMatcher
    and some related functions
    Input:
      desc_1, desc_2: list of feature descriptors
    Return:
      matches: list of feature matches
  '''
  #############################################################################
  #                                   TODO                                    #
  #############################################################################
  bf = cv2.BFMatcher()
  raw_matches = bf.knnMatch(desc_1, desc_2, 2)

  matches = []
  for m, n in raw_matches:
        if m.distance < n.distance * ratio:
            matches.append(m)
  #############################################################################
  #                             END OF YOUR CODE                              #
  #############################################################################

  return matches
```

In [20]:

```
kp_1, desc_1 = get_orb_features(img1)
kp_2, desc_2 = get_orb_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)
```

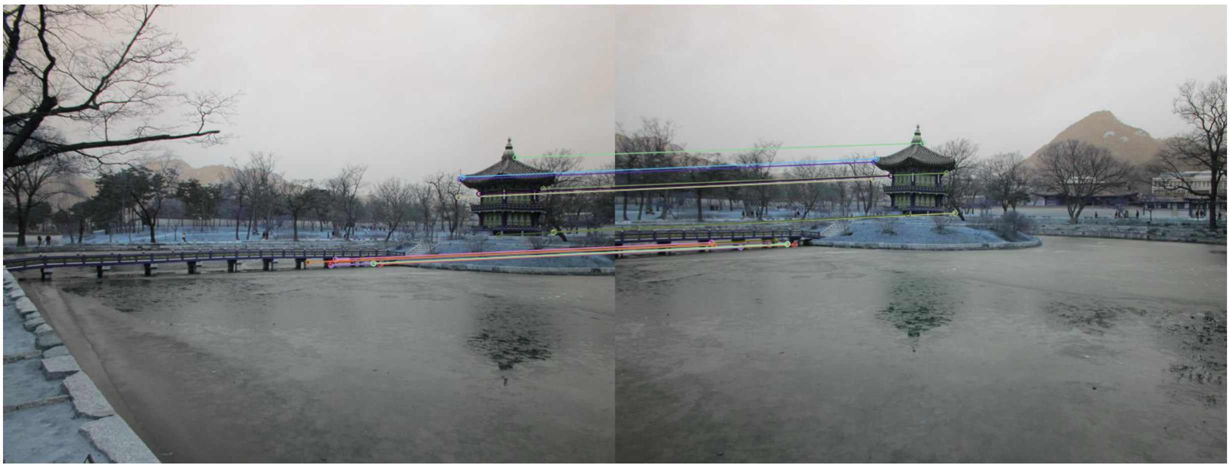
```
keypoints for img1 and img2
```



In [21]:

```
matches = match_keypoints(desc_1, desc_2)
match_plot = cv2.drawMatches(img1, kp_1, img2, kp_2, matches[:20], None, flags=2)
print("orb feature matches")
cv2_imshow(match_plot)
```

```
orb feature matches
```



(b) Find Homography Matrix

In [22]:

```
def find_homography(pts_1, pts_2):
  '''
    Use either nonlinear least squares or 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

    Hint if using nonlinear least square:
      The objective function to optimize here is:
      ||pts_1 - cart(H*homog(pts_2))||^2 where homog(x) converts x into
      homogeneous coordinates and cart(x) converts x to cartesian coordinates.
      You can use scipy.optimize.least_squares for this.

    Hint if using direct linear transform:
      The solution is given by the right-singular vector with the smallest singular
      You can use np.linalg.svd for this.

    Input:
      pts_1, pts_1: (N, 2) matrix
    Return:
      H: the resultant homography matrix (3 x 3)
  '''
  ############################################################################
  #                                  TODO                                    #
  ############################################################################
  A = []
  for i in range(len(pts_1)):
      x, y = pts_1[i][0], pts_1[i][1]
      xp, yp = pts_2[i][0], pts_2[i][1]
      A.append([-x, -y, -1, 0, 0, 0, x*xp, y*xp, xp])
      A.append([0, 0, 0, -x, -y, -1, x*yp, y*yp, yp])
  A = np.asarray(A)
  _, _, V = np.linalg.svd(A)
  H = V[-1].reshape(3, 3)
  H = H / H[-1, -1]
  ############################################################################
  #                              END OF YOUR CODE                            #
  ############################################################################
  return H
```

(c) Implement RANSAC

In [23]:

```
def transform_ransac(x1, x2, verbose=False):
  '''
    Implements RANSAC to estimate homography matrix.
    Hint: Follow the RANSAC steps outlined in the lecture slides.
    Input:
      pts_1, pts_1: (N, 2) matrices
    Return:
      best_model: homography matrix with most inliers
  '''
  ##############################################################################
  #                                   TODO                                     #
  ##############################################################################
  maxCount = 0
  best_model = None
  bestInliers_1 = None
  bestInliers_2 = None
  for i in range(1000):
    randomIdx = random.sample(range(len(x1)), 4)
    pts_1 = np.array([x1[pt] for pt in randomIdx])
    pts_2 = np.array([x2[pt] for pt in randomIdx])

    H = find_homography(pts_1, pts_2)

    count = 0
    inlier_1 = []
    inlier_2 = []

    p_i = np.insert(x1, 2, 1, axis=1).T
    p_ip = x2.T
    Hp_i = np.dot(H, p_i)
    for h in range(2):
      for k in range(len(Hp_i[h])):
        Hp_i[h][k] = Hp_i[h][k]/Hp_i[-1][k]
    Hp_i = Hp_i[:2]
    A = p_ip - Hp_i
    err = np.sqrt(np.sum(np.square(A), axis=0))

    for j in range(len(err)):
      if err[j] < 2:
        count += 1
        inlier_1.append(x1[j])
        inlier_2.append(x2[j])

    inlier_1 = np.asarray(inlier_1)
    inlier_2 = np.asarray(inlier_2)

    if count > maxCount:
      bestInliers_1 = inlier_1
      bestInliers_2 = inlier_2
      maxCount = count

  best_model = find_homography(bestInliers_1, bestInliers_2)
  ##############################################################################
  #                             END OF YOUR CODE                               #
  ##############################################################################
  # return best_inliers, best_model
  return best_model
```

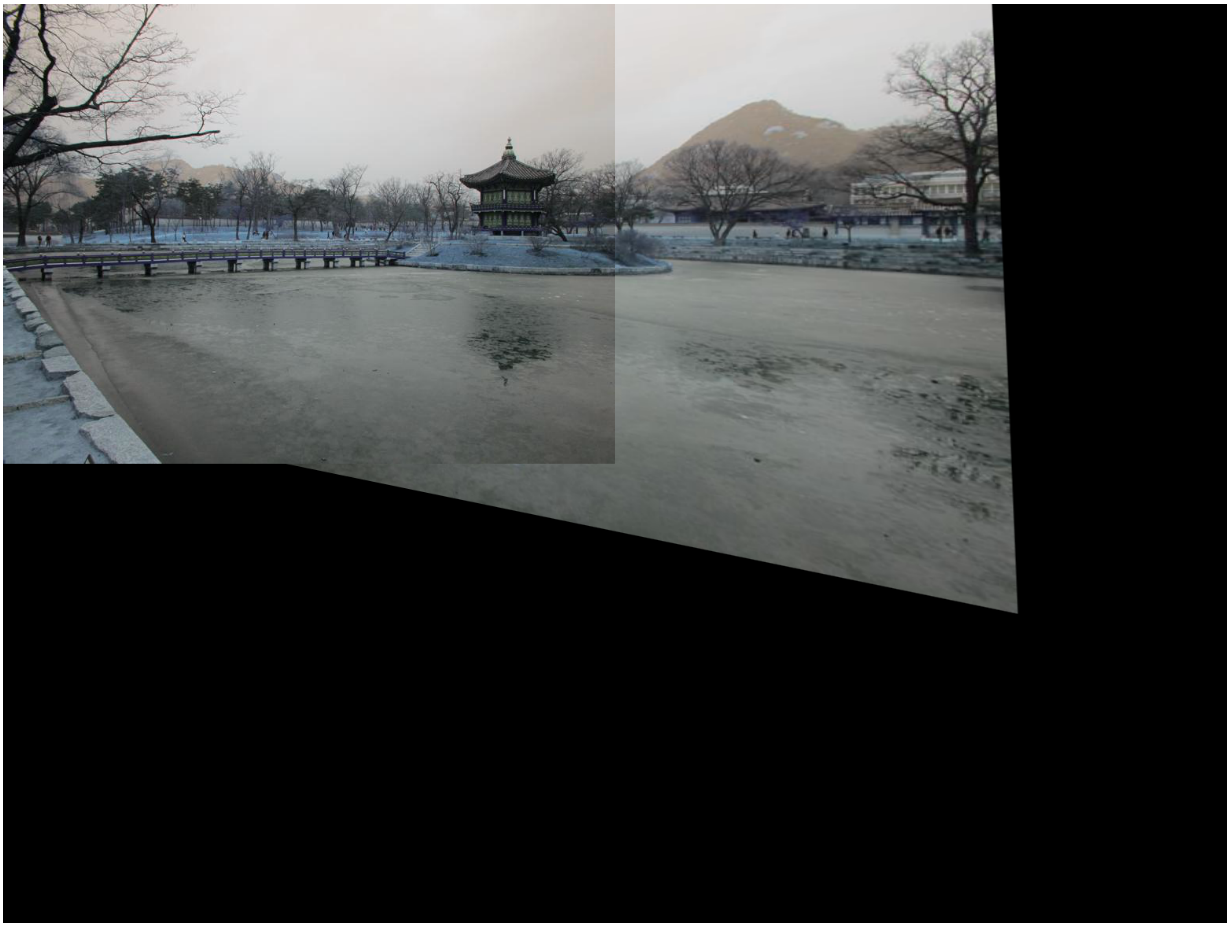
(d) Panoramic Stitching

In [24]:

```
def panoramic_stitching(img1, img2):
  '''
    Given a pair of overlapping images, generate a panoramic image.
    Hint: use the functions that you've written in the previous parts.
    Input:
      img1, img2: cv2 images
    Return:
      final_img: cv2 image of panorama
  '''
  #############################################################################
  #                                   TODO                                    #
  # 1. detect keypoints and extract orb feature descriptors                   #
  # 2. match features between two images                                      #
  # 3. compute homography matrix H transforming points from pts_2 to pts_1.   #
  # Note the order here (not pts_1 to pts_2)!                                 #
  #############################################################################
  kp_1, desc_1 = get_orb_features(img1)
  kp_2, desc_2 = get_orb_features(img2)

  matches = match_keypoints(desc_1, desc_2)

  pts_1 = np.array([kp_1[idx.queryIdx].pt for idx in matches]).reshape(-1, 2)
  pts_2 = np.array([kp_2[idx.trainIdx].pt for idx in matches]).reshape(-1, 2)

  H = transform_ransac(pts_2, pts_1)
  #############################################################################
  #                             END OF YOUR CODE                              #
  #############################################################################
  # apply perspective wrap to stitch images together
  final_img = cv2.warpPerspective(img2, H, (img2.shape[1] + img1.shape[1], img2.shap
  final_img[0:img1.shape[0], 0:img1.shape[1]] = img1

  return final_img
```

In [25]:

```
result = panoramic_stitching(img1, img2)
cv2_imshow(result)
```



Convert to PDF

If the below cell doesn't work, try this alternative.

In [ ]:

```
# generate pdf
# %%capture
!git clone https://gist.github.com/bc5f1add34fef7c7f9fb83d3783311e2.git
!cp bc5f1add34fef7c7f9fb83d3783311e2/colab_pdf.py colab_pdf.py
from colab_pdf import colab_pdf
# change the name to your ipynb file name shown on the top left of Colab window
# Important: make sure that your file name does not contain spaces!
colab_pdf('cktran_09859713.ipynb')
```