Homework3_denizaytemiz

November 7, 2021

1 ECE 4554/ ECE 5554 / Computer Vision

This file contains Problem 5, which is the coding portion of Homework 3. Your job is to implement/modify the sections within this notebook that are marked with "TO DO".

1.1 TO DO: Enter your Virginia Tech Username (PID) here: _____906470552_____

1.2 Honor Code reminder

This is not a "team project". Please review the Honor Code statement in the syllabus.

1.3 Submission guidelines for the coding problems (Google Colab)

1. Please verify that you have entered your Virginia Tech Username in all of the appropriate places.
2. After clicking Runtime->Run all, verify that all of your solutions are visible in this notebook.
3. Click File->Save near the top of the page to save the latest version of your notebook at Google Drive.
4. Verify that the last 2 cells have executed, creating a PDF version of this notebook at Google Drive. (Note: if you face difficulty with this step, please refer to https://pypi.org/project/notebook-as-pdf/)
5. Look at the PDF file and check that all of your solutions are displayed correctly there.
6. Download your notebook file and the PDF version to your laptop.
7. On your laptop, create a ZIP version of this notebook file. (Please don't include the separate data files.) Use file name Homework3_Code_USERNAME.zip, with your own Username.
8. For your PDF version, use file name Homework3_Notebook_USERNAME.pdf, with your own Username.
9. **Submit these 2 files and your PDF file for Problems 1-4 SEPARATELY to Canvas.** Do not zip them all together.

1.4 Overview

This problem consists of the following 4 parts. By completing these parts, you will develop code that will automatically create a composite image from several overlapping images. Each part has equal weight in grading.

a) 2D homography

b) Image warping

c) RANSAC using SIFT keypoints

d) Image stitching

2 Environment setup

[1]:
```
# Mount your Google Drive to this notebook
# The purpose is to allow your code to access to your files
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

[2]:
```
# Change the directory to your own working directory
# Any files under your working directory are available to your code
# TO DO: enter the name of your directory
import os
os.chdir('/content/drive/MyDrive/ECE5554_Computer_Vision/HW3')
```

[3]:
```
# The following is needed to allow the use of SIFT
# (need OpenCV version 4.5 or later)
# For a new installation, you may see a message:
#  "You must restart the runtime in order to use newly installed versions."
#  One way to do this is to click "Runtime" -> "Restart Runtime"
# If you see an error message, you may need to run this step again, after␣
 ↪running the "Import" block that follows
!pip install opencv-python==4.5.3.*
```

```
Collecting opencv-python==4.5.3.*
  Downloading opencv_python-4.5.3.56-cp37-cp37m-manylinux2014_x86_64.whl (49.9
MB)
     || 49.9 MB 16 kB/s
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7
/dist-packages (from opencv-python==4.5.3.*) (1.19.5)
Installing collected packages: opencv-python
  Attempting uninstall: opencv-python
    Found existing installation: opencv-python 4.1.2.30
    Uninstalling opencv-python-4.1.2.30:
      Successfully uninstalled opencv-python-4.1.2.30
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9
which is incompatible.
Successfully installed opencv-python-4.5.3.56
```

[4]:
```
# Import library modules
import sys
import cv2
import numpy as np
import random
import matplotlib.pyplot as plt
# PIL is the Python Imaging Library
from PIL import Image
# The following is a substitute for cv2.imshow, which Colab does not allow
from google.colab.patches import cv2_imshow

print('Python version:', sys.version)
print('OpenCV version:', cv2.__version__)
print('NumPy version: ', np.__version__)
```

```
Python version: 3.7.12 (default, Sep 10 2021, 00:21:48)
[GCC 7.5.0]
OpenCV version: 4.5.3
NumPy version:  1.19.5
```

3 Getting started

Several image files were provided to you (Newman*.png, mandrill.tif, Rubiks_cube.jpg). Upload all of them to your working directory.

The following functions are helpful for loading images into floating-point format, and for displaying images that are in that format. Let's use those functions to visualize 3 separate images that we want to stitch together. Notice that you can adjust the size of the figure that is displayed.

[5]:
```
def load_image(filename):
  img = np.asarray(Image.open(filename))
  img = img.astype("float32")/255.0
  return img

def show_image(img):
  fig = plt.figure()
  fig.set_size_inches(18, 10) # You can adjust the size of the displayed figure
  plt.imshow(img)

left_img = load_image("Newman1.png")
center_img = load_image("Newman2.png")
right_img = load_image("Newman3.png")
show_image(np.concatenate([left_img, center_img, right_img], axis=1))
```

Next, verify that you can use OpenCV tools to detect SIFT-based keypoints. These library functions allow for many options. For example, if you change the `nfeatures` parameter for `SIFT_create`, you'll see different numbers of detected keypoints. You may want to experiment with these parameters later.

[6]:
```
def testSIFT(img1):
  sift = cv2.SIFT_create(nfeatures=500)
  kp = sift.detect(img1, None)
  img1=cv2.drawKeypoints(img1, kp, None, color=(0, 255, 255))

  fig = plt.figure()
  fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
  plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))


img = cv2.imread("Newman1.png", cv2.IMREAD_COLOR)
testSIFT(img)
```

4 Part (a): 2D homography

Implement the 2 functions that are shown in the next code block. (OpenCV has a findHomography() function, and other related functions, but for this assignment you must write your own version.)

1. **compute_homography(src, dst)** receives two matrices, each of size Nx2. Each matrix contains N two-dimensional points. For each value of i, src[i] and dst[i] are corresponding points from two different images. The function should return the homography matrix H of size 3x3 that maps every point from the source (src) to the destination (dst). Guidance: You may assume that N is at least 4. You can set up the problem in a matrix-based, least-squares format. (A somewhat similar problem is in the lecture slides on page 40 of packet 5.) Helpful functions are `np.linalg.eig()` and `np.linalg.eigh()` for computing eigenvalues and eigenvectors. The latter function will prevent warnings due to small imaginary values if you are working with matrices that are real and symmetric.
2. **apply_homography(src, H)** receives points in matrix src (an Nx2 matrix), and a homography transformation H (a 3x3 matrix). This function should use the homography matrix to transform each point in src to a new destination point. Store the resulting points in matrix dst, which is the same size as src. The function should return dst. Guidance: remember to use homogeneous coordinates when implementing this transformation.

[7]:
```
def compute_homography(src, dst):
  n, m = src.shape
  #n = len(src)
  #initializing A matrix in the eqn A*h=0
  A = np.zeros([2*n, 9])
  for i in range(0, n):
    A[2*i,0]=src[i,0]
    A[2*i,1]=src[i,1]
    A[2*i,2]= 1
    A[2*i,3]= 0
    A[2*i,4]= 0
    A[2*i,5]= 0
    A[2*i,6]=-dst[i,0]*src[i,0]
    A[2*i,7]=-dst[i,0]*src[i,1]
    A[2*i,8]=-dst[i,0]
    A[2*i+1,0]= 0
    A[2*i+1,1]= 0
    A[2*i+1,2]= 0
    A[2*i+1,3]= src[i,0]
    A[2*i+1,4]= src[i,1]
    A[2*i+1,5]= 1
    A[2*i+1,6]=-dst[i,1]*src[i,0]
    A[2*i+1,7]=-dst[i,1]*src[i,1]
    A[2*i+1,8]=-dst[i,1]

  AT= A.transpose()
  sym_matrix = np.matmul(AT,A)
  #eigen_val is the array that stores the eigen values with respective
  #columns of the eigen_mat which are the eigenvectors.
  eigen_val,eigen_mat = np.linalg.eigh(sym_matrix)
  #stores the index of the min value in the eigen_val list
  index = np.argmin(eigen_val)
  #the h matrix is the column of the eigen_mat(eigenvector) that corresponds to␣
 ↪the smallest value of
  #eigenvalue.
  h = eigen_mat[:,index]
  #normalizing the vector
  h = h/h[8]
  #resize 9 lengthed array to 3x3 matrix
  homography = h.reshape(3,3)
  return homography

def apply_homography(src, H):
  ##initializing the dst matrix
  #in homogeneous coordinates:
  #dst[point1]= homography*source[point1]
  dst = np.zeros([src.shape[0], 2])
```

```
  s = np.ones([3,1])
  # loop over the rows of src = loop over points xs,ys
  for i in range(0, src.shape[0]):
    #s is = [xs(point i) ys(point i) 1], in homogeneous coordinate.
    s[0] = src[i,0]
    s[1] = src[i,1]
    #d is =[w*xd(point i) w*yd(point i) w], in homogeneous coordinate.
    d = np.matmul(H, s)
    #we normalize the d array to make it in the form of d = [xd(point i)␣
↪yd(point i) 1]
    d[0] = d[0]/d[2]
    d[1] = d[1]/d[2]
    #print(d)
    #then we convert it back to inhomogeneous coordinates by simply elimanting␣
↪1 at the end of the array
    dst[i,0] = d[0]
    dst[i,1] = d[1]
    #dst[i] = [xd(point i) yd(point i)]
  return dst
```

You do not need to change the following function. It will test your homography code, and should help you in debugging. Corresponding points are placed in src_pts and dst_pts. If your homography code is correct, it should map the points given in test_pts to locations that are close to the points given in match_pts_correct. If you have correctly implemented compute_homography() and apply_homography(), then the printed difference values should be close to 0. (A small difference, such as 0.001, should be acceptable.)

[8]:
```
def test_homography():
  src_pts = np.matrix('0, 0; 1, 0; 1, 1; 0, 1')
  dst_pts = np.matrix('5, 4; 7, 4; 7, 5; 6, 6')
  H = compute_homography(src_pts, dst_pts)
  test_pts = np.matrix('0,  0; 1, 0; 1, 1; 0, 1')
  match_pts = apply_homography(test_pts, H)
  match_pts_correct = np.matrix('5, 4; 7, 4; 7, 5; 6, 6')
  print('Your 1st solution differs from our solution by: %f'
    % np.square(match_pts - match_pts_correct).sum())

  src_pts = np.matrix('0, 0; 1, 0; 1, 1; 0, 1; 2, 3')
  dst_pts = np.matrix('5, 4; 7, 4; 7, 5; 6, 6; 7.25, 5.5')
  H = compute_homography(src_pts, dst_pts)
  test_pts = np.matrix('0,  0; 1, 0; 1, 1; 0, 1')
  match_pts = apply_homography(test_pts, H)
  match_pts_correct = np.matrix('5, 4; 7, 4; 7, 5; 6, 6')
  print('Your 2nd solution differs from our solution by: %f'
    % np.square(match_pts - match_pts_correct).sum())

  src_pts = np.matrix('347, 313; 502, 341; 386, 571; 621, 508')
  dst_pts = np.matrix('274, 286; 436, 305; 305, 527; 615, 506')
```

```
  H = compute_homography(src_pts, dst_pts)
  test_pts = np.matrix('259, 505; 350, 371; 400, 675; 636, 104')
  match_pts = apply_homography(test_pts, H)
  match_pts_correct = np.matrix('195.13761083, 448.12645033;'
    '275.27269386, 336.54819916;'
    '317.37663747, 636.78403426;'
    '618.50438823, 28.78963905')
  print('Your 3rd solution differs from our solution by: %f'
    % np.square(match_pts - match_pts_correct).sum())

test_homography()
```

```
Your 1st solution differs from our solution by: 0.000000
Your 2nd solution differs from our solution by: 0.000000
Your 3rd solution differs from our solution by: 0.000013
```

5 Part (b): Image warping using a 2D homography

Implement the following function so that it performs image warping from a source image (src_img) to a newly created destination image (dst_img). Do not use any additional OpenCV functions. The homography H that is provided indicates a desired mapping from src_img to dst_img. To prevent gaps in the output image, however, it is suggested that your implementation should iterate over all pixels in dst_img. In that case, your code should use the *inverse* of H to find values in src_img as it iterates over dst_img.

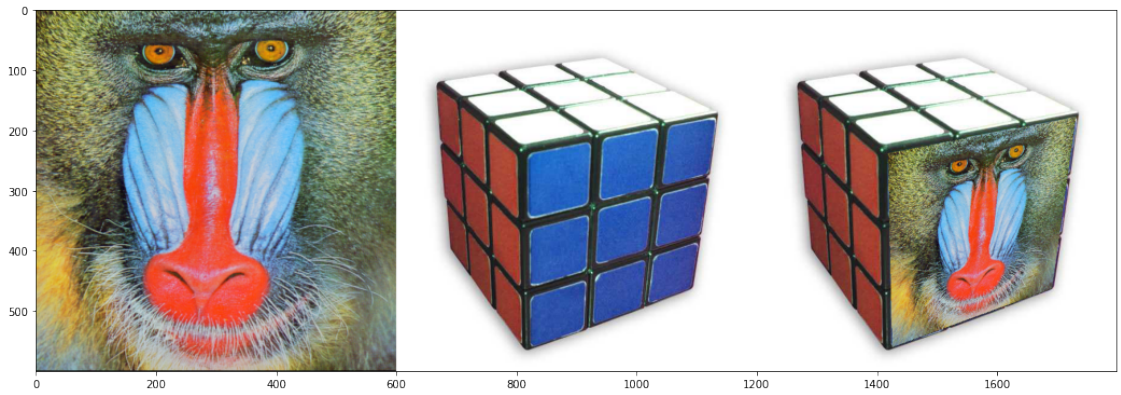
[9]:
```
def warp_img(src_img, H, dst_img_size):
  '''Warping of a source image using a homography.
   Input:
      src_img: source image with shape (m, n, 3)
      H: homography, with shape (3, 3), from source image to destination image
      dst_img_size: height and width of destination image; shape (2,)
   Output:
      dst_img: destination image; height and width specified by dst_img_size␣
↪parameter

   TO DO: Implement the warp_img function.
  '''
  ##dst_img_size[0]=dest image row number, dst_img_size[1] is column number
  ##each element is 3 dimensional(RGB)
  dst_img = np.zeros([dst_img_size[0], dst_img_size[1], 3])
  H_inv = np.linalg.inv(H)
  ##reverse mapping
  for x in range(0, dst_img_size[0]):
    for y in range(0, dst_img_size[1]):
      s = np.array([x,y,1])
      source = np.matmul(H_inv, s)
      xd, yd = int(round(source[0]/source[2])), int(round(source[1]/source[2]))
```

```
      xs = int(xd)
      ys = int(yd)
      if 0 <= xs < src_img.shape[0] and 0 <= ys < src_img.shape[1]:
        dst_img[y,x,:] = src_img[ys, xs, :]
  return dst_img
```

Use the functions below to help debug your implementation. You do not need to modify these functions. If your code is correct, the output here should show a mandrill image that has been warped to overlay the blue side of a Rubik's cube.

[10]:
```
def binary_mask(img):
  '''Create a binary mask of the image content.
   Input:
    img: source image, shape (m, n, 3)
   Output:
    mask: image of shape (m, n) and type 'int'. For pixel [i, j] of mask,
      if pixel [i, j] in img is nonzero in any channel, assign 1 to mask[i, j].
      Else, assign 0 to mask[i, j].
  '''
  mask = (img[:, :, 0] > 0) | (img[:, :, 1] > 0) | (img[:, :, 2] > 0)
  mask = mask.astype("int")
  return mask

def test_warp():
  src_img = load_image('mandrill.tif')
  canvas_img = load_image('Rubiks_cube.jpg')

  # The following are corners of the mandrill image
  src_pts = np.matrix('0, 0; 0, 511; 511, 511; 511, 0')
  # The following are corners of the blue face of the Rubik's cube
  canvas_pts = np.matrix('218, 238; 225, 560; 490, 463; 530, 178')

  # The following was used during debugging
  # print(canvas_pts)
  # test_x=218
  # test_y=238
  # cv2.circle(canvas_img, (218, 238), 4, (255, 0, 0), thickness=10)

  H = compute_homography(src_pts, canvas_pts)
  dst_img = warp_img(src_img, H, [canvas_img.shape[0], canvas_img.shape[1]])
  dst_mask = 1 - binary_mask(dst_img)
  dst_mask = np.stack((dst_mask,) * 3, -1)
  out_img = np.multiply(canvas_img, dst_mask) + dst_img

  dsize = (600, 600) # width and height of canvas_im
  src_smaller = cv2.resize(src_img, dsize, interpolation=cv2.INTER_AREA)

  warped_img = np.concatenate((src_smaller, canvas_img, out_img), axis=1)
```

```
  show_image(np.clip(warped_img, 0, 1))

test_warp()
```



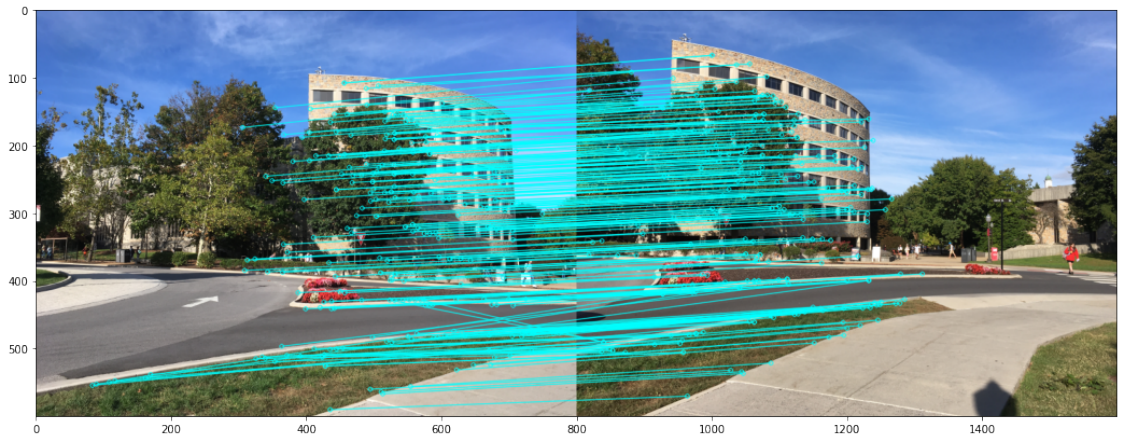
6 Part (c) RANSAC using SIFT keypoints

You have already confirmed that you can detect SIFT-based keypoints. Next, verify that you can use a matching technique from OpenCV that tries to detect corresponding keypoints between 2 images. The code should display the correspondences and draw lines between them.

By the way, cv2.BFmatcher() is OpenCV's "brute force" matcher. More description is given in https://github.com/abidrahmank/OpenCV2-Python-Tutorials/blob/master/source/py_tutorials/py_feature2d/py_matcher/py_matcher.rst

[11]:
```
def genSIFTMatchPairs(img1, img2):
  sift = cv2.SIFT_create()
  #kp = sift.detect(img1, None)
  #kp = cv2.SIFT(edgeThreshold=10)
  kp1, des1 = sift.detectAndCompute(img1, None)
  kp2, des2 = sift.detectAndCompute(img2, None)
  bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)
  matches = bf.match(des1, des2)
  matches = sorted(matches, key = lambda x:x.distance)

  pts1 = np.zeros((250, 2))
  pts2 = np.zeros((250, 2))
  for i in range(250):
    pts1[i,:] = kp1[matches[i].queryIdx].pt
    pts2[i,:] = kp2[matches[i].trainIdx].pt

  return pts1, pts2, matches[:250], kp1, kp2

def test_matches():
  img1 = cv2.imread('Newman1.png')
```

```
  img2 = cv2.imread('Newman3.png')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)

  # In the following, parameter flags=2 will remove unmatched points from the␣
↪display
  matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
↪flags=2,
    matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

  fig = plt.figure()
  fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
  plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))


test_matches()
```



Notice that some of the matches from the previous step are not correct. It is possible that those false matches could act as outliers in your homography estimation. In that case we would expect a poor result in any composite image that we create.

As discussed during lectures, the RANSAC algorithm is a popular way to deal with outliers during model fitting. Here you must implement your own function that uses the RANSAC approach to compute a homography H.

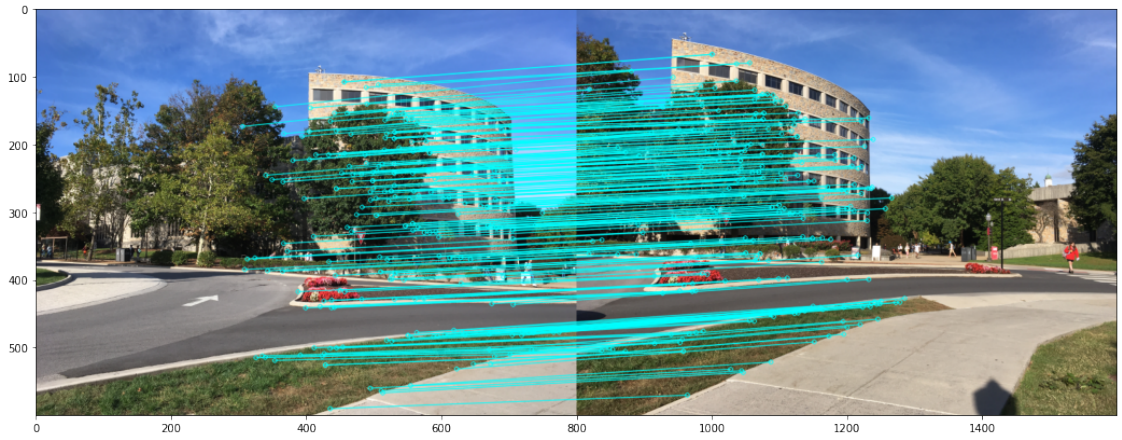
[12]:
```
def RANSAC(Xs, Xd, max_iter, eps):
  ##initialization of variables
  ## temp variables are for loop variables they will be replaced in each␣
↪iteration
  H = np.zeros([3,3])
  inliers_id = []
  ##returns 4 indexes from Xs matrix(corresponds to point indexes), p is list
  ##outer loop that determines how many set of random 4 points we will try out.
  for k in range(max_iter):
    p = random.sample(range(0, len(Xs)), 4)
```

```
    ## arrays for 4 random points are stored in src and dst(corresponding␣
↪points)
    src = np.array([Xs[p[0]],Xs[p[1]],Xs[p[2]],Xs[p[3]]])
    dst = np.array([Xd[p[0]],Xd[p[1]],Xd[p[2]],Xd[p[3]]])
    ##temp_H is the estimated homography from random points
    temp_H = compute_homography(src,dst)
    ##calculated Xd matrix is cXd.
    cXd = apply_homography(Xs,temp_H)
    temp_inliers_id = []
    for i in range(0,len(Xd)):
      ##error between each point correspondence
      error = np.sum(np.square(Xd[i] - cXd[i]))
      ##if error is less then threshold then save the indexes of the points␣
↪that are good approximations
      if error <= eps:
        temp_inliers_id.append(i)
        ## if for a particular homography there are more inliers then the␣
↪previous one,
        ## we replace the saved homography matrix with better one and keep the␣
↪index array
        ## associated with it
    if len(temp_inliers_id) > len(inliers_id):
      H = temp_H
      inliers_id = temp_inliers_id

  return inliers_id, H
```

Now we can look at the matches between keypoints after using your RANSAC implementation. You do not need to modify the code in the next block. If you implemented RANSAC correctly, hopefully all of the incorrect matches have been discarded in the following output.

[13]:
```
def test_ransac():
  img1 = cv2.imread('Newman1.png')
  img2 = cv2.imread('Newman3.png')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)

  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  new_matches = []
  for i in range(len(inliers_idx)):
    new_matches.append(matches1to2[inliers_idx[i]])

  matching_result = cv2.drawMatches(img1, kp1, img2, kp2, new_matches, None,␣
↪flags=2,
                                    matchColor = (255, 255, 0),␣
↪singlePointColor=(0, 0, 255))

  fig = plt.figure()
  fig.set_size_inches(18, 10) # You can adjust the size of the displayed figure
```

```
  plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))

test_ransac()
```



7 Part (d): Image stitching.

We are now ready to create a panorama from the three images that were shown at the beginning. The stitch_img() function below receives a Python list of images, which the code should stitch together to form one large image. You must use the functions that you have written to create the panorama.

We do not expect the result to be perfect, partly because some lens distortion is always present and it will prevent the homography from aligning things perfectly. However, for full credit, try to write code that will stitch the images together with very little noticable seam.

Use the following code to test your implementation. This code just reads in the images, calls your stitch_img() function, and plots the results.

Finally, use 3 or more photos of your own to create a panorama. It is suggested that you take new photos using your camera (e.g., with your phone or laptop). Remember to stand in place and try to rotate the camera about its point of projection. (If the camera position changes, then the transformation between 2 images is no longer a homography.)

Please include these new photos as part of your HW3 submission. We would like to select the best panoramas (as decided by the graders) to show during a lecture. (If you prefer that we do not show your photos to the class, that is fine. In that case, please tell us using a comment at the top of the following code block.) To reduce computation time, it is acceptable to resize your images down to a smaller size before processing them. You could consider using `cv2.resize()` for that purpose. For example, the provided Newman Library images were reduced to 800x600 from much larger dimensions.

Question 5

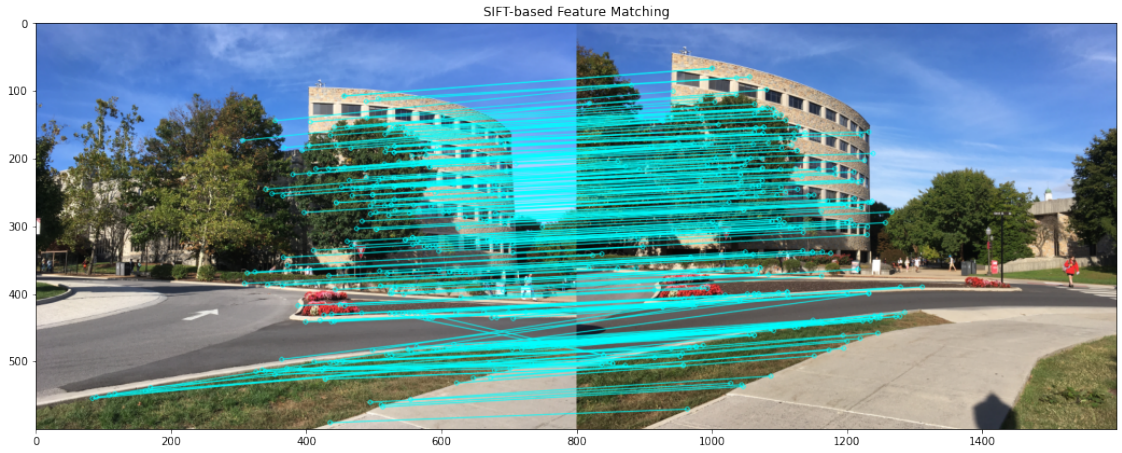
[14]:
```
import time
```

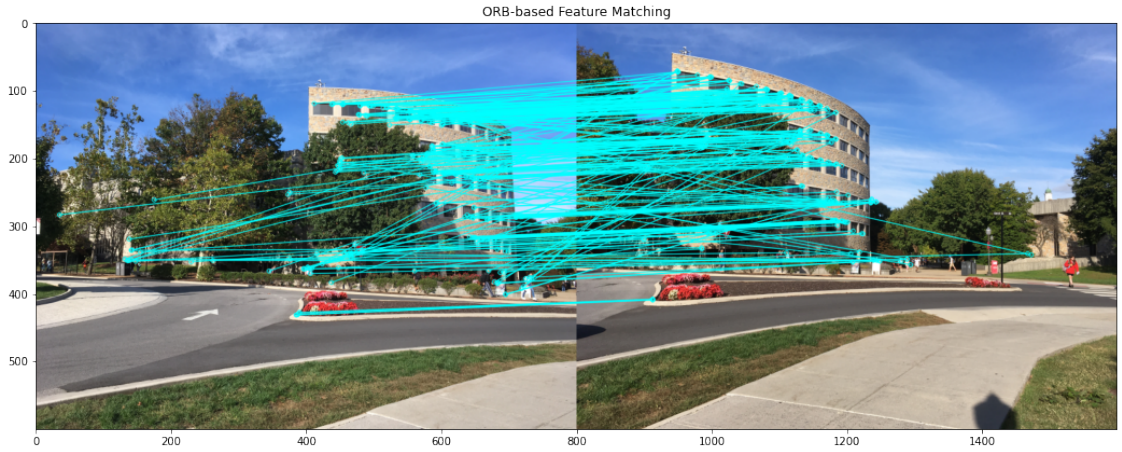
[15]:
```
def genSIFTMatchPairs(img1, img2):
  tic = time.perf_counter()
```

```
  sift = cv2.SIFT_create()
  #kp = sift.detect(img1, None)
  #kp = cv2.SIFT(edgeThreshold=10)
  kp1, des1 = sift.detectAndCompute(img1, None)
  kp2, des2 = sift.detectAndCompute(img2, None)
  bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)
  matches = bf.match(des1, des2)
  matches = sorted(matches, key = lambda x:x.distance)

  pts1 = np.zeros((250, 2))
  pts2 = np.zeros((250, 2))
  for i in range(250):
    pts1[i,:] = kp1[matches[i].queryIdx].pt
    pts2[i,:] = kp2[matches[i].trainIdx].pt
  toc = time.perf_counter()
  print('Computation Time for SIFT:', toc-tic)
  return pts1, pts2, matches[:250], kp1, kp2

def test_matches1():
  img1 = cv2.imread('Newman1.png')
  img2 = cv2.imread('Newman3.png')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
  # In the following, parameter flags=2 will remove unmatched points from the␣
 ↪display
  matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
 ↪flags=2,
  matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

  fig = plt.figure()
  fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
  plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
  plt.title("SIFT-based Feature Matching")
## Generating matching pairs with ORB
def genORBMatchPairs(img1, img2):
  tic = time.perf_counter()
  orb = cv2.ORB_create()
  kp1, des1 = orb.detectAndCompute(img1, None)
  kp2, des2 = orb.detectAndCompute(img2, None)
  bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=False)
  matches = bf.match(des1, des2)
  matches = sorted(matches, key = lambda x:x.distance)
  pts1 = np.zeros((250, 2))
  pts2 = np.zeros((250, 2))
  for i in range(250):
    pts1[i,:] = kp1[matches[i].queryIdx].pt
    pts2[i,:] = kp2[matches[i].trainIdx].pt
  toc = time.perf_counter()
```

```
  print('Computation Time for ORB:', toc-tic)
  return pts1, pts2, matches[:250], kp1, kp2

def test_matches2():
  img1 = cv2.imread('Newman1.png')
  img2 = cv2.imread('Newman3.png')
  pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)

  # In the following, parameter flags=2 will remove unmatched points from the␣
 ↪display
  matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
 ↪flags=2,
  matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

  fig = plt.figure()
  fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
  plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
  plt.title("ORB-based Feature Matching")

test_matches1()
test_matches2()
```

```
Computation Time for SIFT: 0.8690416430000028
Computation Time for ORB: 0.1930382850000001
```





[16]:
```
img1 = cv2.imread('Newman1.png')
  img2 = cv2.imread('Newman3.png')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for SIFT:', len(inliers_idx))

  pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for ORB:', len(inliers_idx))
```

```
Computation Time for SIFT: 0.9554361710000308
Number of Correct Matches after RANSAC for SIFT: 227
Computation Time for ORB: 0.10252658199999587
Number of Correct Matches after RANSAC for ORB: 144
```

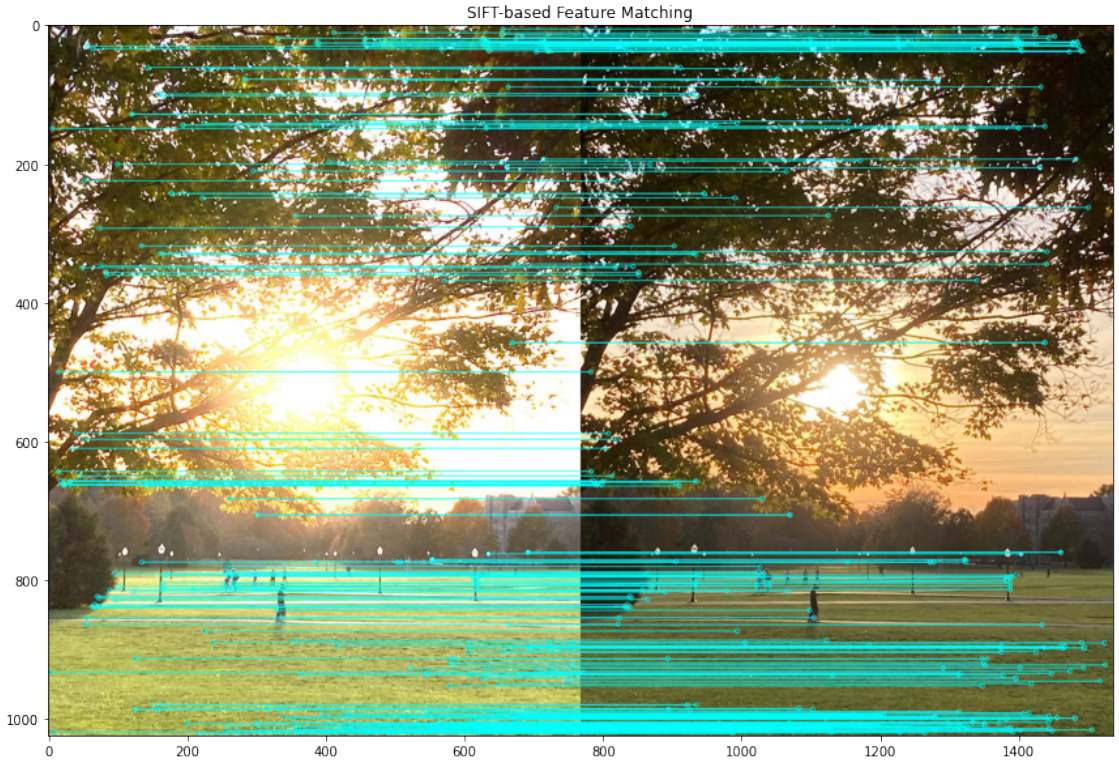
We try SIFT and ORB for varying imgage intensities in the following.

[17]:
```
def test_matches3():
  img1 = cv2.imread('img1.jpeg')
  img2 = cv2.imread('img2.jpeg')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
  # In the following, parameter flags=2 will remove unmatched points from the␣
 ↪display
  matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
 ↪flags=2,
  matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

  fig = plt.figure()
  fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
  plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
  plt.title("SIFT-based Feature Matching")
```

[18]:
```
def test_matches4():
  img1 = cv2.imread('img1.jpeg')
  img2 = cv2.imread('img2.jpeg')
  pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)

  # In the following, parameter flags=2 will remove unmatched points from the␣
 ↪display
  matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
 ↪flags=2,
  matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

  fig = plt.figure()
  fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
  plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
  plt.title("ORB-based Feature Matching")
```

[19]:
```
test_matches3()
```

```
Computation Time for SIFT: 4.3664269369999715
```



[20]:
```
test_matches4()
```

```
Computation Time for ORB: 0.18389229000001706
```

[21]:
```
print('We try SIFT and ORB for varying imgage intensities in the following.')

  img1 = cv2.imread('img1.jpeg')
  img2 = cv2.imread('img2.jpeg')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for SIFT:', len(inliers_idx))

  pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for ORB:', len(inliers_idx))
```

```
We try SIFT and ORB for varying imgage intensities in the following.
Computation Time for SIFT: 4.497718271000053
Number of Correct Matches after RANSAC for SIFT: 249
Computation Time for ORB: 0.1380116820000694
Number of Correct Matches after RANSAC for ORB: 190
```

We try SIFT and ORB for rotation of 180 degrees.

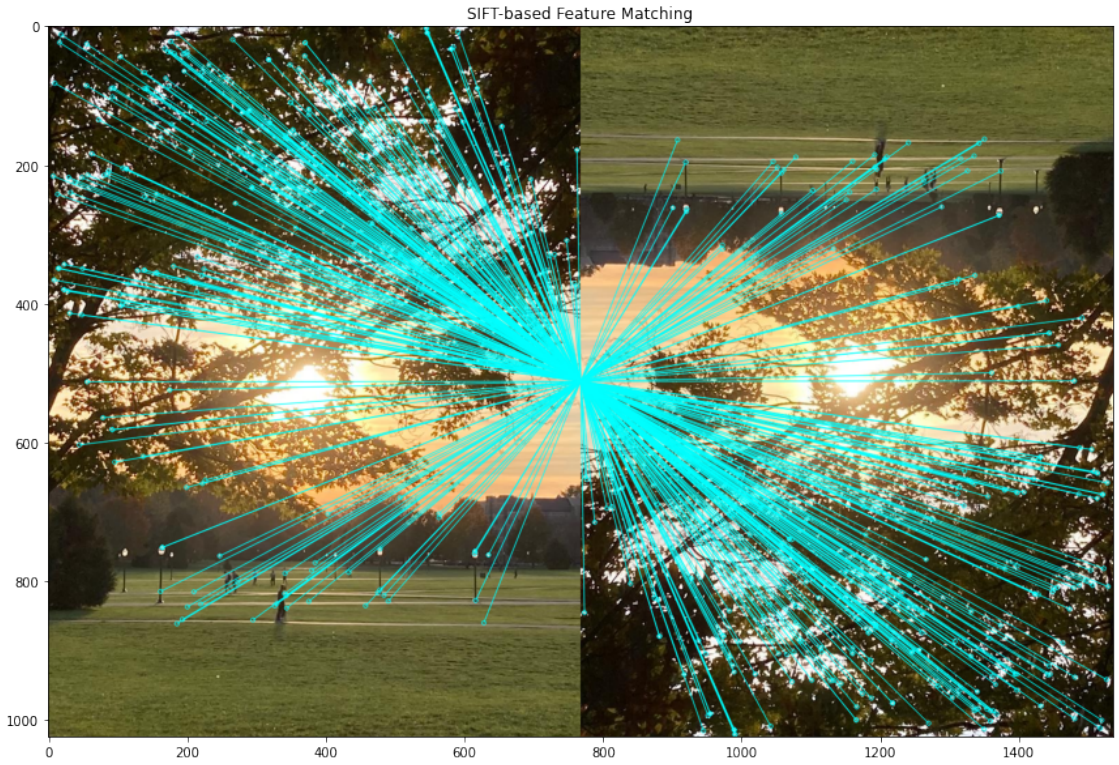
[22]:
```
def test_matches5():
  img1 = cv2.imread('img2.jpeg')
  img2 = cv2.imread('rot.jpeg')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
```

```
 # In the following, parameter flags=2 will remove unmatched points from the␣
↪display
 matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
↪flags=2,
 matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

 fig = plt.figure()
 fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
 plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
 plt.title("SIFT-based Feature Matching")
```

[23]:
```
def test_matches6():
 img1 = cv2.imread('img2.jpeg')
 img2 = cv2.imread('rot.jpeg')
 pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)

 # In the following, parameter flags=2 will remove unmatched points from the␣
↪display
 matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
↪flags=2,
 matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

 fig = plt.figure()
 fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
 plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
 plt.title("ORB-based Feature Matching")
```

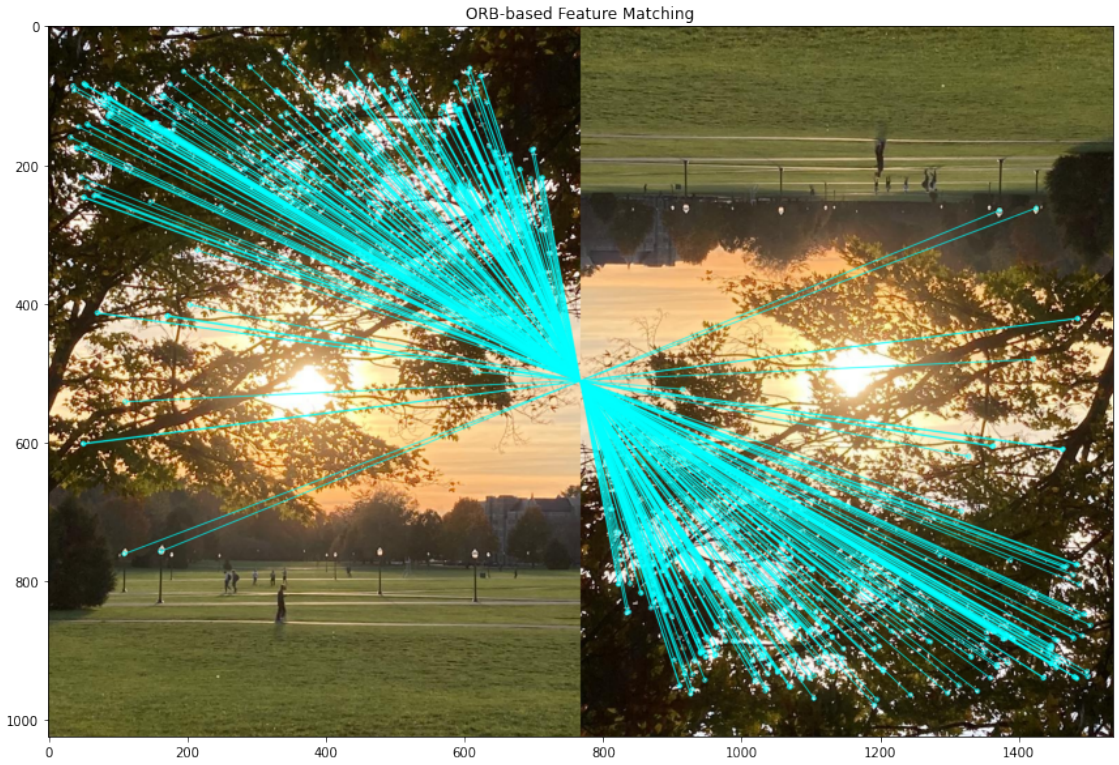
[24]:
```
test_matches5()
```

```
Computation Time for SIFT: 4.27413044299999
```



[25]:
```
test_matches6()
```

```
Computation Time for ORB: 0.163997391999942
```



[26]:
```
print('We try SIFT and ORB for varying rotation of 180 degrees.')

  img1 = cv2.imread('img2.jpeg')
  img2 = cv2.imread('rot.jpeg')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for SIFT:', len(inliers_idx))

  pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for ORB:', len(inliers_idx))
```

```
We try SIFT and ORB for varying rotation of 180 degrees.
Computation Time for SIFT: 4.4128645439999445
Number of Correct Matches after RANSAC for SIFT: 250
Computation Time for ORB: 0.13478695399999197
Number of Correct Matches after RANSAC for ORB: 250
```

We try SIFT and ORB for scaling in the following.

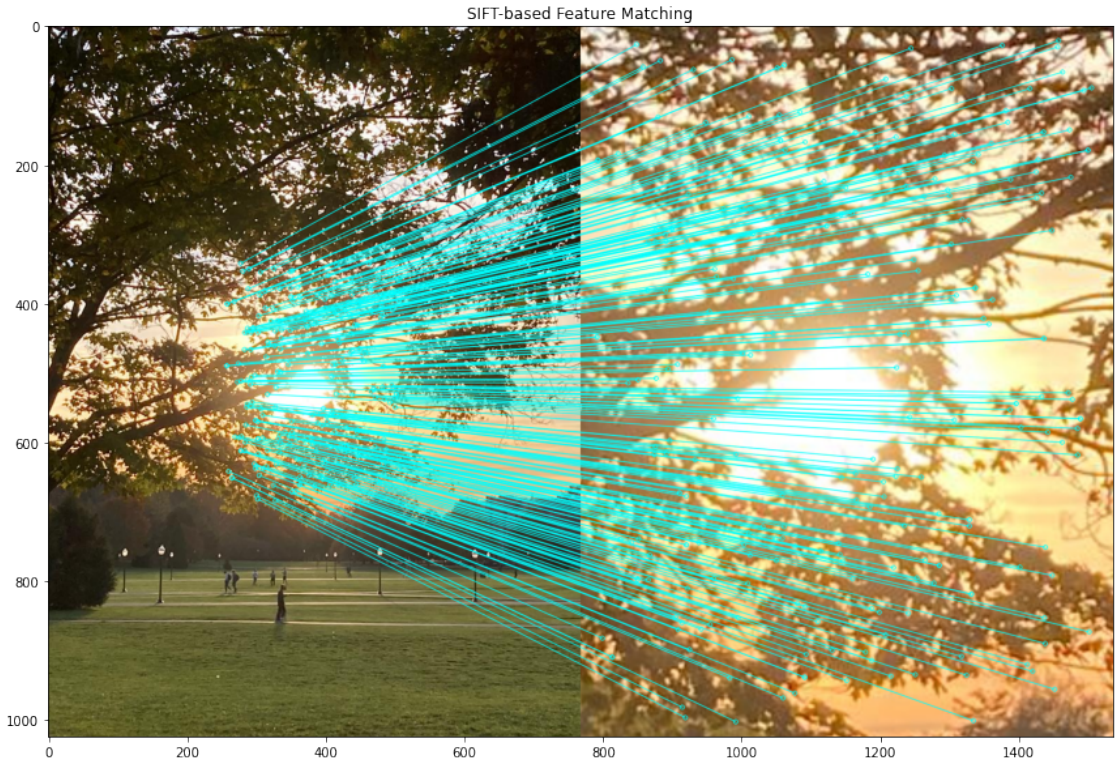
[27]:
```
def test_matches7():
  img1 = cv2.imread('img2.jpeg')
  img2 = cv2.imread('scale.jpeg')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
```

```
 # In the following, parameter flags=2 will remove unmatched points from the␣
↪display
 matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
↪flags=2,
 matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

 fig = plt.figure()
 fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
 plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
 plt.title("SIFT-based Feature Matching")
```

[28]:
```
def test_matches8():
 img1 = cv2.imread('img2.jpeg')
 img2 = cv2.imread('scale.jpeg')
 pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)

 # In the following, parameter flags=2 will remove unmatched points from the␣
↪display
 matching_result = cv2.drawMatches(img1, kp1, img2, kp2, matches1to2, None,␣
↪flags=2,
 matchColor = (255, 255, 0), singlePointColor=(0, 0, 255))

 fig = plt.figure()
 fig.set_size_inches(18,10) # You can adjust the size of the displayed figure
 plt.imshow(cv2.cvtColor(matching_result, cv2.COLOR_BGR2RGB))
 plt.title("ORB-based Feature Matching")
```

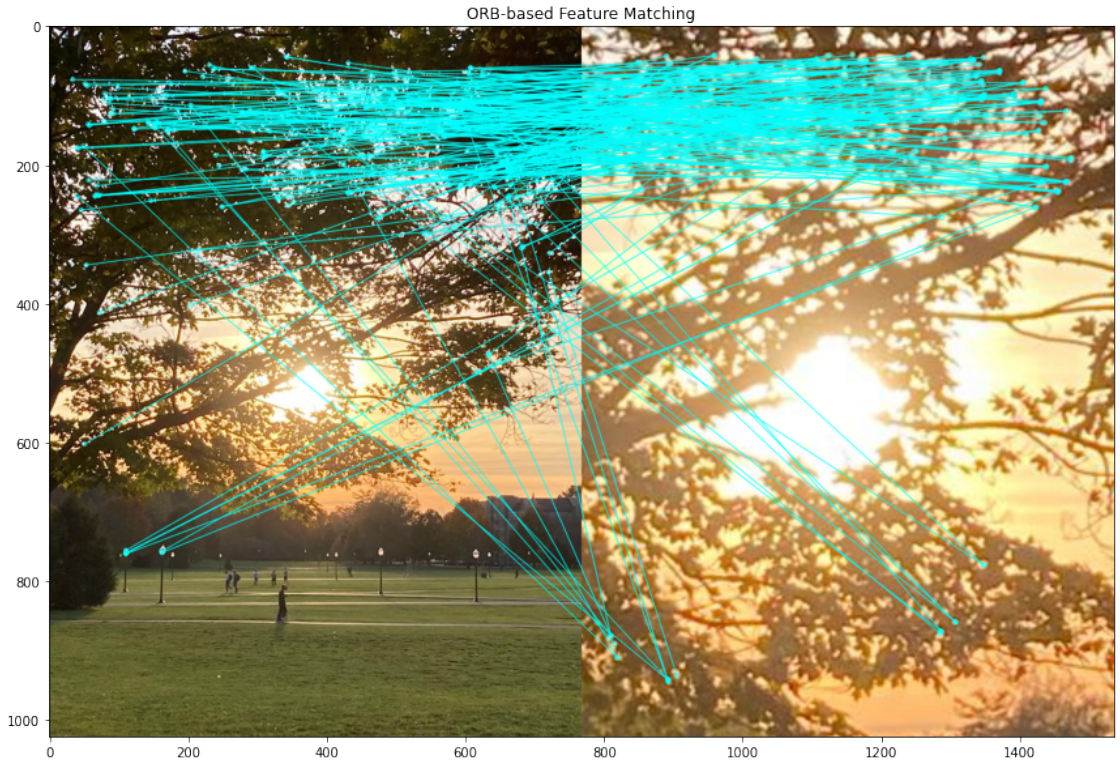
[29]:
```
test_matches7()
```

```
Computation Time for SIFT: 2.5655824680000023
```



[30]:
```
test_matches8()
```

```
Computation Time for ORB: 0.15035022099993967
```



[31]:
```
print('We try SIFT and ORB for scaling in the following.')

  img1 = cv2.imread('img2.jpeg')
  img2 = cv2.imread('scale.jpeg')
  pts1, pts2, matches1to2, kp1, kp2 = genSIFTMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for SIFT:', len(inliers_idx))

  pts1, pts2, matches1to2, kp1, kp2 = genORBMatchPairs(img1, img2)
  inliers_idx, H = RANSAC(pts1, pts2, 500, 50)
  print('Number of Correct Matches after RANSAC for ORB:', len(inliers_idx))
```

```
We try SIFT and ORB for scaling in the following.
Computation Time for SIFT: 2.7171486659999573
Number of Correct Matches after RANSAC for SIFT: 250
Computation Time for ORB: 0.13323104100004457
Number of Correct Matches after RANSAC for ORB: 11
```

8 The comparision between ORB and SIFT

8.0.1 Special Cases:

8.0.2 1. Varying image intensity

8.0.3 2. Rotation by 180 degrees

8.0.4 3. Scaling by %160

8.0.5 In each case one can observe ORB is significantly faster than SIFT. However in most cases, in every case we show, SIFT works better than ORB in terms of accuracy. For instance when we use the photos of the Newman library, we get the report as;

8.0.6 Computation Time for SIFT: 0.8949596839993319

8.0.7 Number of Correct Matches after RANSAC for SIFT: 227

8.0.8 Computation Time for ORB: 0.08123339599842438

8.0.9 Number of Correct Matches after RANSAC for ORB: 144

8.0.10 Which shows ORB is 10 times faster but SIFT gets almost 2 times more correct points. Therefore depending on whether the application requires speed or accuracy one would be more useful than the other.

8.0.11 Comparing the special cases;

8.0.12 In varying image intensities case, SIFT is more accurate but ORB is much faster. In case of rotation by 180 degrees however, both works pretty well. ORB is as accurate as SIFT yet again it is much faster. So one can say it is preferable in this case. Considering scaling, again SIFT works much better than ORB in terms of the number of correctly matched points. It is expected SIFT to be accurate since it is scale invariant. In this case ORB has a very poor accuracy.

---

9 Creating a PDF version of your current notebook

[ ]:
```
#The following two installation steps are needed to generate a PDF version of␣
 ↪the notebook
#(These lines are needed within Google Colab, but are not needed within a local␣
 ↪version of Jupyter notebook)
!apt-get -qq install texlive texlive-xetex texlive-latex-extra pandoc
!pip install --quiet pypandoc
```

```
Extracting templates from packages: 100%
Preconfiguring packages ...
Selecting previously unselected package fonts-droid-fallback.
(Reading database ... 155219 files and directories currently installed.)
Preparing to unpack .../00-fonts-droid-fallback_1%3a6.0.1r16-1.1_all.deb ...
```

```
Unpacking fonts-droid-fallback (1:6.0.1r16-1.1) ...
Selecting previously unselected package fonts-lato.
Preparing to unpack .../01-fonts-lato_2.0-2_all.deb ...
Unpacking fonts-lato (2.0-2) ...
Selecting previously unselected package poppler-data.
Preparing to unpack .../02-poppler-data_0.4.8-2_all.deb ...
Unpacking poppler-data (0.4.8-2) ...
Selecting previously unselected package tex-common.
Preparing to unpack .../03-tex-common_6.09_all.deb ...
Unpacking tex-common (6.09) ...
Selecting previously unselected package fonts-lmodern.
Preparing to unpack .../04-fonts-lmodern_2.004.5-3_all.deb ...
Unpacking fonts-lmodern (2.004.5-3) ...
Selecting previously unselected package fonts-noto-mono.
Preparing to unpack .../05-fonts-noto-mono_20171026-2_all.deb ...
Unpacking fonts-noto-mono (20171026-2) ...
Selecting previously unselected package fonts-texgyre.
Preparing to unpack .../06-fonts-texgyre_20160520-1_all.deb ...
Unpacking fonts-texgyre (20160520-1) ...
Selecting previously unselected package javascript-common.
Preparing to unpack .../07-javascript-common_11_all.deb ...
Unpacking javascript-common (11) ...
Selecting previously unselected package libcupsfilters1:amd64.
Preparing to unpack .../08-libcupsfilters1_1.20.2-0ubuntu3.1_amd64.deb ...
Unpacking libcupsfilters1:amd64 (1.20.2-0ubuntu3.1) ...
Selecting previously unselected package libcupsimage2:amd64.
Preparing to unpack .../09-libcupsimage2_2.2.7-1ubuntu2.8_amd64.deb ...
Unpacking libcupsimage2:amd64 (2.2.7-1ubuntu2.8) ...
Selecting previously unselected package libijs-0.35:amd64.
Preparing to unpack .../10-libijs-0.35_0.35-13_amd64.deb ...
Unpacking libijs-0.35:amd64 (0.35-13) ...
Selecting previously unselected package libjbig2dec0:amd64.
Preparing to unpack .../11-libjbig2dec0_0.13-6_amd64.deb ...
Unpacking libjbig2dec0:amd64 (0.13-6) ...
Selecting previously unselected package libgs9-common.
Preparing to unpack .../12-libgs9-common_9.26~dfsg+0-0ubuntu0.18.04.14_all.deb
...
Unpacking libgs9-common (9.26~dfsg+0-0ubuntu0.18.04.14) ...
Selecting previously unselected package libgs9:amd64.
Preparing to unpack .../13-libgs9_9.26~dfsg+0-0ubuntu0.18.04.14_amd64.deb ...
Unpacking libgs9:amd64 (9.26~dfsg+0-0ubuntu0.18.04.14) ...
Selecting previously unselected package libjs-jquery.
Preparing to unpack .../14-libjs-jquery_3.2.1-1_all.deb ...
Unpacking libjs-jquery (3.2.1-1) ...
Selecting previously unselected package libkpathsea6:amd64.
Preparing to unpack .../15-libkpathsea6_2017.20170613.44572-8ubuntu0.1_amd64.deb
...
Unpacking libkpathsea6:amd64 (2017.20170613.44572-8ubuntu0.1) ...
```

```
Selecting previously unselected package libpotrace0.
Preparing to unpack .../16-libpotrace0_1.14-2_amd64.deb ...
Unpacking libpotrace0 (1.14-2) ...
Selecting previously unselected package libptexenc1:amd64.
Preparing to unpack .../17-libptexenc1_2017.20170613.44572-8ubuntu0.1_amd64.deb
...
Unpacking libptexenc1:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Selecting previously unselected package rubygems-integration.
Preparing to unpack .../18-rubygems-integration_1.11_all.deb ...
Unpacking rubygems-integration (1.11) ...
Selecting previously unselected package ruby2.5.
Preparing to unpack .../19-ruby2.5_2.5.1-1ubuntu1.10_amd64.deb ...
Unpacking ruby2.5 (2.5.1-1ubuntu1.10) ...
Selecting previously unselected package ruby.
Preparing to unpack .../20-ruby_1%3a2.5.1_amd64.deb ...
Unpacking ruby (1:2.5.1) ...
Selecting previously unselected package rake.
Preparing to unpack .../21-rake_12.3.1-1ubuntu0.1_all.deb ...
Unpacking rake (12.3.1-1ubuntu0.1) ...
Selecting previously unselected package ruby-did-you-mean.
Preparing to unpack .../22-ruby-did-you-mean_1.2.0-2_all.deb ...
Unpacking ruby-did-you-mean (1.2.0-2) ...
Selecting previously unselected package ruby-minitest.
Preparing to unpack .../23-ruby-minitest_5.10.3-1_all.deb ...
Unpacking ruby-minitest (5.10.3-1) ...
Selecting previously unselected package ruby-net-telnet.
Preparing to unpack .../24-ruby-net-telnet_0.1.1-2_all.deb ...
Unpacking ruby-net-telnet (0.1.1-2) ...
Selecting previously unselected package ruby-power-assert.
Preparing to unpack .../25-ruby-power-assert_0.3.0-1_all.deb ...
Unpacking ruby-power-assert (0.3.0-1) ...
Selecting previously unselected package ruby-test-unit.
Preparing to unpack .../26-ruby-test-unit_3.2.5-1_all.deb ...
Unpacking ruby-test-unit (3.2.5-1) ...
Selecting previously unselected package libruby2.5:amd64.
Preparing to unpack .../27-libruby2.5_2.5.1-1ubuntu1.10_amd64.deb ...
Unpacking libruby2.5:amd64 (2.5.1-1ubuntu1.10) ...
Selecting previously unselected package libsynctex1:amd64.
Preparing to unpack .../28-libsynctex1_2017.20170613.44572-8ubuntu0.1_amd64.deb
...
Unpacking libsynctex1:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Selecting previously unselected package libtexlua52:amd64.
Preparing to unpack .../29-libtexlua52_2017.20170613.44572-8ubuntu0.1_amd64.deb
...
Unpacking libtexlua52:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Selecting previously unselected package libtexluajit2:amd64.
Preparing to unpack
.../30-libtexluajit2_2017.20170613.44572-8ubuntu0.1_amd64.deb ...
```

```
Unpacking libtexluajit2:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Selecting previously unselected package libzzip-0-13:amd64.
Preparing to unpack .../31-libzzip-0-13_0.13.62-3.1ubuntu0.18.04.1_amd64.deb ...
Unpacking libzzip-0-13:amd64 (0.13.62-3.1ubuntu0.18.04.1) ...
Selecting previously unselected package lmodern.
Preparing to unpack .../32-lmodern_2.004.5-3_all.deb ...
Unpacking lmodern (2.004.5-3) ...
Selecting previously unselected package preview-latex-style.
Preparing to unpack .../33-preview-latex-style_11.91-1ubuntu1_all.deb ...
Unpacking preview-latex-style (11.91-1ubuntu1) ...
Selecting previously unselected package t1utils.
Preparing to unpack .../34-t1utils_1.41-2_amd64.deb ...
Unpacking t1utils (1.41-2) ...
Selecting previously unselected package tex-gyre.
Preparing to unpack .../35-tex-gyre_20160520-1_all.deb ...
Unpacking tex-gyre (20160520-1) ...
Selecting previously unselected package texlive-binaries.
Preparing to unpack .../36-texlive-
binaries_2017.20170613.44572-8ubuntu0.1_amd64.deb ...
Unpacking texlive-binaries (2017.20170613.44572-8ubuntu0.1) ...
Selecting previously unselected package texlive-base.
Preparing to unpack .../37-texlive-base_2017.20180305-1_all.deb ...
Unpacking texlive-base (2017.20180305-1) ...
Selecting previously unselected package texlive-fonts-recommended.
Preparing to unpack .../38-texlive-fonts-recommended_2017.20180305-1_all.deb ...
Unpacking texlive-fonts-recommended (2017.20180305-1) ...
Selecting previously unselected package texlive-latex-base.
Preparing to unpack .../39-texlive-latex-base_2017.20180305-1_all.deb ...
Unpacking texlive-latex-base (2017.20180305-1) ...
Selecting previously unselected package texlive-latex-recommended.
Preparing to unpack .../40-texlive-latex-recommended_2017.20180305-1_all.deb ...
Unpacking texlive-latex-recommended (2017.20180305-1) ...
Selecting previously unselected package texlive.
Preparing to unpack .../41-texlive_2017.20180305-1_all.deb ...
Unpacking texlive (2017.20180305-1) ...
Selecting previously unselected package texlive-pictures.
Preparing to unpack .../42-texlive-pictures_2017.20180305-1_all.deb ...
Unpacking texlive-pictures (2017.20180305-1) ...
Selecting previously unselected package texlive-latex-extra.
Preparing to unpack .../43-texlive-latex-extra_2017.20180305-2_all.deb ...
Unpacking texlive-latex-extra (2017.20180305-2) ...
Selecting previously unselected package texlive-plain-generic.
Preparing to unpack .../44-texlive-plain-generic_2017.20180305-2_all.deb ...
Unpacking texlive-plain-generic (2017.20180305-2) ...
Selecting previously unselected package tipa.
Preparing to unpack .../45-tipa_2%3a1.3-20_all.deb ...
Unpacking tipa (2:1.3-20) ...
Selecting previously unselected package texlive-xetex.
```

```
Preparing to unpack .../46-texlive-xetex_2017.20180305-1_all.deb ...
Unpacking texlive-xetex (2017.20180305-1) ...
Setting up libgs9-common (9.26~dfsg+0-0ubuntu0.18.04.14) ...
Setting up libkpathsea6:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up libjs-jquery (3.2.1-1) ...
Setting up libtexlua52:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up fonts-droid-fallback (1:6.0.1r16-1.1) ...
Setting up libsynctex1:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up libptexenc1:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up tex-common (6.09) ...
update-language: texlive-base not installed and configured, doing nothing!
Setting up poppler-data (0.4.8-2) ...
Setting up tex-gyre (20160520-1) ...
Setting up preview-latex-style (11.91-1ubuntu1) ...
Setting up fonts-texgyre (20160520-1) ...
Setting up fonts-noto-mono (20171026-2) ...
Setting up fonts-lato (2.0-2) ...
Setting up libcupsfilters1:amd64 (1.20.2-0ubuntu3.1) ...
Setting up libcupsimage2:amd64 (2.2.7-1ubuntu2.8) ...
Setting up libjbig2dec0:amd64 (0.13-6) ...
Setting up ruby-did-you-mean (1.2.0-2) ...
Setting up t1utils (1.41-2) ...
Setting up ruby-net-telnet (0.1.1-2) ...
Setting up libijs-0.35:amd64 (0.35-13) ...
Setting up rubygems-integration (1.11) ...
Setting up libpotrace0 (1.14-2) ...
Setting up javascript-common (11) ...
Setting up ruby-minitest (5.10.3-1) ...
Setting up libzzip-0-13:amd64 (0.13.62-3.1ubuntu0.18.04.1) ...
Setting up libgs9:amd64 (9.26~dfsg+0-0ubuntu0.18.04.14) ...
Setting up libtexluajit2:amd64 (2017.20170613.44572-8ubuntu0.1) ...
Setting up fonts-lmodern (2.004.5-3) ...
Setting up ruby-power-assert (0.3.0-1) ...
Setting up texlive-binaries (2017.20170613.44572-8ubuntu0.1) ...
update-alternatives: using /usr/bin/xdvi-xaw to provide /usr/bin/xdvi.bin
(xdvi.bin) in auto mode
update-alternatives: using /usr/bin/bibtex.original to provide /usr/bin/bibtex
(bibtex) in auto mode
Setting up texlive-base (2017.20180305-1) ...
mktexlsr: Updating /var/lib/texmf/ls-R-TEXLIVEDIST...
mktexlsr: Updating /var/lib/texmf/ls-R-TEXMFMAIN...
mktexlsr: Updating /var/lib/texmf/ls-R...
mktexlsr: Done.
tl-paper: setting paper size for dvips to a4: /var/lib/texmf/dvips/config
/config-paper.ps
tl-paper: setting paper size for dvipdfmx to a4: /var/lib/texmf/dvipdfmx
/dvipdfmx-paper.cfg
tl-paper: setting paper size for xdvi to a4: /var/lib/texmf/xdvi/XDvi-paper
```

```
tl-paper: setting paper size for pdftex to a4:
/var/lib/texmf/tex/generic/config/pdftexconfig.tex
Setting up texlive-fonts-recommended (2017.20180305-1) ...
Setting up texlive-plain-generic (2017.20180305-2) ...
Setting up texlive-latex-base (2017.20180305-1) ...
Setting up lmodern (2.004.5-3) ...
Setting up texlive-latex-recommended (2017.20180305-1) ...
Setting up texlive-pictures (2017.20180305-1) ...
Setting up tipa (2:1.3-20) ...
Regenerating '/var/lib/texmf/fmtutil.cnf-DEBIAN'... done.
Regenerating '/var/lib/texmf/fmtutil.cnf-TEXLIVEDIST'... done.
update-fmtutil has updated the following file(s):
        /var/lib/texmf/fmtutil.cnf-DEBIAN
        /var/lib/texmf/fmtutil.cnf-TEXLIVEDIST
If you want to activate the changes in the above file(s),
you should run fmtutil-sys or fmtutil.
Setting up texlive (2017.20180305-1) ...
Setting up texlive-latex-extra (2017.20180305-2) ...
Setting up texlive-xetex (2017.20180305-1) ...
Setting up ruby2.5 (2.5.1-1ubuntu1.10) ...
Setting up ruby (1:2.5.1) ...
Setting up ruby-test-unit (3.2.5-1) ...
Setting up rake (12.3.1-1ubuntu0.1) ...
Setting up libruby2.5:amd64 (2.5.1-1ubuntu1.10) ...
Processing triggers for mime-support (3.60ubuntu1) ...
Processing triggers for libc-bin (2.27-3ubuntu1.3) ...
/sbin/ldconfig.real: /usr/local/lib/python3.7/dist-
packages/ideep4py/lib/libmkldnn.so.0 is not a symbolic link
```

[ ]:
```
# TO DO: Provide the full path to your Jupyter notebook file
!jupyter nbconvert --to PDF "/content/drive/MyDrive/ECE5554_Computer_Vision/HW3/
 ↪Homework3_denizaytemiz.ipynb"
```