Kunal Goyal
CS558 – Computer Vision
Homework3

Code:

Ps4.py:

```
# ps4

import cv2
import numpy as np
import sys
from collections import OrderedDict
import random
from grad_utils import *
from harris_corners import *
from get_keypoints import *
from get_matches import *
from draw_keypoints import *
from ransac_trans import *
from ransac_sim import *
from ransac_sim_affine import *

imgs = ['transA.jpg', 'transB.jpg', 'simA.jpg', 'simB.jpg']

# Harris Corners
# ==============
def ps4_1_a():
    images = imgs[0:3:2]
    for idx, img in enumerate(images):
        img = cv2.imread('input/'+img, cv2.IMREAD_GRAYSCALE)
        # calculate the X and Y gradients of the two images using the above filter
        img_grad_x = calc_grad_x(img, 3, norm=True)
        img_grad_y = calc_grad_y(img, 3, norm=True)
        # save the gradient pair
        cv2.imwrite('output/ps4-1-a-'+str(idx+1)+'.png', np.hstack((img_grad_x,
                                                    img_grad_y)))
    print('Finished calculating and saved the gradients of the images!')
        #  cv2.imshow('', np.hstack((img_grad_x, img_grad_y)))
        # cv2.waitKey(0); cv2.destroyAllWindows()

def ps4_1_b():
    for idx, img_name in enumerate(imgs):
        # read image form file
        img = cv2.imread('input/'+img_name, cv2.IMREAD_GRAYSCALE)
        # calculate harris values for image
        Rs = harris_values(img, window_size=3, harris_scoring=0.04, norm=True)
        # save harris values image
        cv2.imwrite('output/ps4-1-b-'+str(idx+1)+'.png', Rs)
    print('Finished and saved all harris value images to files.')



def ps4_1_c():
    for idx, img_name in enumerate(imgs):
```

Kunal Goyal
CS558 – Computer Vision
Homework3

```
        # read image form file
        img = cv2.imread('input/'+img_name, cv2.IMREAD_COLOR)
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
        gray = np.float32(gray)
        #  corners = cv2.cornerHarris(gray,2,3,0.06) > 0.001*corners.max()
        corners = harris_corners(img, window_size=3, harris_scoring=0.04,
                                 threshold=1e-3, nms_size=5)
        img[corners > 0] = [0, 0, 255]
        cv2.imwrite('output/ps4-1-c-'+str(idx+1)+'.png', img)
        #  cv2.imshow('',img); cv2.waitKey(0); cv2.destroyAllWindows()
    print('Finished harris corner detection and saved new images!')

# SIFT Features
# =============
def ps4_2_a():
    for idx, img_name in enumerate(imgs):
        img = cv2.imread('input/'+img_name, cv2.IMREAD_COLOR)
        img, keypoints = get_keypoints(img)
        cv2.imwrite('output/ps4-2-a-'+str(idx+1)+'.png', img)
    print('Finished keypoint detection and saved drawn images!')


def ps4_2_b():
    for idx in range(len(imgs)//2):
        img1 = cv2.imread('input/'+imgs[2*idx], cv2.IMREAD_COLOR)
        img2 = cv2.imread('input/'+imgs[2*idx+1], cv2.IMREAD_COLOR)
        matching = draw_keypoints(img1, img2)
        cv2.imwrite('output/ps4-2-b-'+str(idx+1)+'.png', matching)
    print('Finished keypoint detection and saved matched keypoints!')

# RANSAC
# ======
def ps4_3_a():
    transA = cv2.imread('input/transA.jpg', cv2.IMREAD_COLOR)
    transB = cv2.imread('input/transB.jpg', cv2.IMREAD_COLOR)
    matching = ransac_trans(transA, transB)
    cv2.imwrite('output/ps4-3-a-1.png', matching)

def ps4_3_b():
    simA = cv2.imread('input/simA.jpg', cv2.IMREAD_COLOR)
    simB = cv2.imread('input/simB.jpg', cv2.IMREAD_COLOR)
    matching,_ = ransac_sim(simA, simB)
    cv2.imwrite('output/ps4-3-b-1.png', matching)

def ps4_3_c():
    simA = cv2.imread('input/simA.jpg', cv2.IMREAD_COLOR)
    simB = cv2.imread('input/simB.jpg', cv2.IMREAD_COLOR)
    matching,_ = ransac_sim_affine(simA, simB)
    cv2.imwrite('output/ps4-3-c-1.png', matching)

def ps4_3_d():
```

Kunal Goyal
CS558 – Computer Vision
Homework3

```
    simA = cv2.imread('input/simA.jpg', cv2.IMREAD_COLOR)
    simB = cv2.imread('input/simB.jpg', cv2.IMREAD_COLOR)
    _,transform = ransac_sim(simA, simB)
    if len(transform) == 0:
        print('Failed to calculate affine transform!')
        return
    warpedB = cv2.warpAffine(simB, transform, simB.shape[1::-1],
                             flags=cv2.WARP_INVERSE_MAP)
    blend = warpedB * 0.5
    blend[:simA.shape[0], :simA.shape[1]] += simA * 0.5
    blend = cv2.normalize(blend, blend, alpha=0, beta=255,
                          norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    #cv2.imshow('picture3d', blend)
    #cv2.waitKey(0); cv2.destroyAllWindows()
    cv2.imwrite('output/ps4-3-d-1.png', blend)

def ps4_3_e():
    simA = cv2.imread('input/simA.jpg', cv2.IMREAD_COLOR)
    simB = cv2.imread('input/simB.jpg', cv2.IMREAD_COLOR)
    _,transform = ransac_sim_affine(simA, simB)
    if len(transform) == 0:
        print('Failed to calculate affine transform!')
        return
    warpedB = cv2.warpAffine(simB, transform, simB.shape[1::-1],
                             flags=cv2.WARP_INVERSE_MAP)
    blend = warpedB * 0.5
    blend[:simA.shape[0], :simA.shape[1]] += simA * 0.5
    blend = cv2.normalize(blend, blend, alpha=0, beta=255,
                          norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    #cv2.imshow('picture3e', blend)
    #cv2.waitKey(0); cv2.destroyAllWindows()
    cv2.imwrite('output/ps4-3-e-1.png', blend)

ps4_list = OrderedDict([('1a', ps4_1_a), ('1b', ps4_1_b), ('1c', ps4_1_c),
                        ('2a', ps4_2_a), ('2b', ps4_2_b), ('3a', ps4_3_a),
                        ('3b', ps4_3_b), ('3c', ps4_3_c), ('3d', ps4_3_d),
                        ('3e', ps4_3_e)])

if __name__ == '__main__':
    if len(sys.argv) == 2:
        if sys.argv[1] in ps4_list:
            print('\nExecuting task %s\n=================='%sys.argv[1])
            ps4_list[sys.argv[1]]()
        else:
            print('\nGive argument from list {1a,1b,2a,2b,3a,3b,3c,3d}\
                for the corresponding task.')
    else:
        print('\n * Executing all tasks: * \n')
        for idx in range(len(ps4_list)):
            print('\nExecuting task: %s\n=================='%
                list(ps4_list.keys())[idx])
```

Kunal Goyal
CS558 – Computer Vision
Homework3

```
    list(ps4_list.values())[idx]()
```

grad_utils.py:

```
import cv2
import numpy as np

def calc_grad_x(img, k_sobel=3, norm=False):
    grad_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, k_sobel)
    if norm:
        grad_x = cv2.normalize(grad_x, grad_x, alpha=0, beta=255,
                               norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    return grad_x

def calc_grad_y(img, k_sobel=3, norm=False):
    grad_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, k_sobel)
    if norm:
        grad_y = cv2.normalize(grad_y, grad_y, alpha=0, beta=255,
                               norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    return grad_y

def calc_grad_orientation(Ix, Iy):
    return np.arctan2(Iy, Ix)

def calc_grad_mag(Ix, Iy):
    return np.sqrt(Ix ** 2 + Iy ** 2)
```

harris_corners.py:

```
import cv2
import numpy as np
from grad_utils import *

def harris_values(img, window_size=5, harris_scoring=0.04, norm=False):
    # calculate image gradients on x and y dimensions
    Ix = calc_grad_x(img, 3)
    Iy = calc_grad_y(img, 3)
    Ixx = Ix ** 2
    Ixy = Ix * Iy
    Iyx = Iy * Ix
    Iyy = Iy ** 2
    # create the weight window matrix
    c = np.zeros((window_size,)*2, dtype=np.float32);
    c[window_size // 2, window_size // 2] = 1.0
    w = cv2.GaussianBlur(c, (window_size,)*2, 0)
```

```
    #  w = np.ones((window_size,)*2)
    # calculate the harris values for all pixels of the image
    Rs = np.zeros(img.shape, dtype=np.float32)
    for r in range(w.shape[0]//2, img.shape[0] - w.shape[0]//2):
        minr = max(0, r - w.shape[0]//2)
        maxr = min(img.shape[0], minr + w.shape[0])
        for c in range(w.shape[1]//2, img.shape[1] - w.shape[1]//2):
            minc = max(0, c - w.shape[1]//2)
            maxc = min(img.shape[1], minc + w.shape[1])
            wIxx = Ixx[minr:maxr, minc:maxc]
            wIxy = Ixy[minr:maxr, minc:maxc]
            wIyx = Iyx[minr:maxr, minc:maxc]
            wIyy = Iyy[minr:maxr, minc:maxc]
            Mxx = (w * wIxx).sum()
            Mxy = (w * wIxy).sum()
            Myx = (w * wIyx).sum()
            Myy = (w * wIyy).sum()
            M = np.array([Mxx, Mxy, Myx, Myy]).reshape((2,2))
            Rs[r,c] = np.linalg.det(M)- harris_scoring * (M.trace() ** 2)
    if norm:
        Rs = cv2.normalize(Rs, Rs, alpha=0, beta=255,
                           norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    return Rs

def harris_corners(img, window_size=5, harris_scoring=0.04, threshold=1e-2,  nms_size=10):
    if len(img.shape) > 2:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # calculate harris values for all valid pixels
    Rs = harris_values(img, window_size, harris_scoring)
    # apply thresholding
    Rs = Rs * (Rs > (threshold * Rs.max())) * (Rs > 0)
    # apply non maximal suppression
    rows, columns = np.nonzero(Rs)
    new_Rs = np.zeros(Rs.shape)
    for r,c in zip(rows,columns):
        minr = max(0, r - nms_size // 2)
        maxr = min(img.shape[0], minr + nms_size)
        minc = max(0, c - nms_size // 2)
        maxc = min(img.shape[1], minc + nms_size)
        if Rs[r,c] == Rs[minr:maxr,minc:maxc].max():
            new_Rs[r,c] = Rs[r,c]
            #  Rs[minr:r, minc:c] = 0
            #  Rs[r+1:maxr, c+1:maxc] = 0
    return new_Rs
```

get_keypoints.py :

```
import cv2
import numpy as np
```

Kunal Goyal
CS558 – Computer Vision
Homework3

```
from harris_corners import *

def get_keypoints(img, draw_keypoints=True):
    if len(img) > 1:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY).astype(np.float32)

    # find harris corners -> keypoints
    # !!! Uses opencv builtin detector, since it's faster than mine
    corners = cv2.cornerHarris(gray,2,3,0.04)
    corners = cv2.normalize(corners, corners, alpha=0, beta=255,
                    norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    threshold=85
    rows, cols = np.nonzero(corners > threshold)
    # Calculate the image gradients
    Ix = calc_grad_x(gray)
    Iy = calc_grad_y(gray)
    O = calc_grad_orientation(Ix, Iy)
    Mag = calc_grad_mag(Ix, Iy)
    # Assign the keypoints
    keypoints = np.zeros((len(rows),))
    keypoints = []
    for i in range(len(rows)):
        r = rows[i];
        c = cols[i]
        kp = cv2.KeyPoint(float(c), float(r), size=20, angle=np.rad2deg(O[r,c]), octave=0)
        keypoints.append(kp)
    # Draw the keypoints on the image
    if draw_keypoints:
        cv2.drawKeypoints(img, keypoints, img,
                    flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
    return img, keypoints
```

get_matches.py:

```
import cv2
import numpy as np
from get_keypoints import *

def get_matches(img1, img2):
    img1_feat, kpts1 = get_keypoints(img1, draw_keypoints=False)
    img2_feat, kpts2 = get_keypoints(img2, draw_keypoints=False)

    # create sift instance
    sift = cv2.xfeatures2d.SIFT_create()
    # get descriptors
    descriptors1 = sift.compute(img1, kpts1)[1]
    descriptors2 = sift.compute(img2, kpts2)[1]

    # get matches
    bfm = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
```

```
    matches = bfm.match(descriptors1, descriptors2)
    matches = sorted(matches, key = lambda x:x.distance)

    return img1_feat, img2_feat, kpts1, kpts2, matches
```

draw_keypoints.py:

```
import cv2
import numpy as np
from get_matches import *

def draw_keypoints(img1, img2):
    _,_,kpts1, kpts2, matches = get_matches(img1, img2)
    matched_image = np.array([])
    matched_image = cv2.drawMatches(img1, kpts1, img2, kpts2, matches[:10],
                                    flags=2, outImg=matched_image)
    return matched_image
```

ransac_trans.py:

```
import cv2
import numpy as np
import random
from get_matches import *

def ransac_trans(transA, transB):
    img1, img2, kpts1, kpts2, matches = get_matches(transA, transB)
    tolerance = 10
    best_match = 0
    consensus_set = []
    # Find consensus of translation between a random key point and the rest for
    # a number of times to find the best match regarding translation
    for i in range(100):
        idx = random.randint(0, len(matches)-1)
        kp1 = kpts1[matches[idx].queryIdx]
        kp2 = kpts2[matches[idx].trainIdx]
        dx = int(kp1.pt[0] - kp2.pt[0])
        dy = int(kp1.pt[1] - kp2.pt[1])
        temp_consensus_set = []
        for j, match in enumerate(matches):
            kp1 = kpts1[match.queryIdx]
            kp2 = kpts2[match.trainIdx]
            dxi = int(kp1.pt[0] - kp2.pt[0])
            dyi = int(kp1.pt[1] - kp2.pt[1])
            if abs(dx - dxi) < tolerance and abs(dy - dyi) < tolerance:
                temp_consensus_set.append(j)
        if len(temp_consensus_set) > len(consensus_set):
            consensus_set = temp_consensus_set
```

```
        best_match = idx
    # calculate best match translation
    kp1 = kpts1[matches[best_match].queryIdx]
    kp2 = kpts2[matches[best_match].trainIdx]
    dx = int(kp1.pt[0] - kp2.pt[0])
    dy = int(kp1.pt[1] - kp2.pt[1])
    consensus_matches = np.array(matches)[consensus_set]
    matched_image = np.array([])
    # draw matches with biggest consensus
    matched_image = cv2.drawMatches(img1, kpts1, img2, kpts2, consensus_matches,
                            flags=2, outImg=matched_image)
    print('Best match: idx=%d with consensus=%d or %d%%\nTranslation: dx=%dpx '
          'and dy=%dpx'%(best_match, len(consensus_set),
                    100*len(consensus_set) / len(matches), dx, dy))
    return matched_image
```

ransac_sim.py:

```
import cv2
import numpy as np
import random
from get_matches import *

def ransac_sim(simA, simB):
    _, _, kpts1, kpts2, matches = get_matches(simA, simB)
    img1 = simA
    img2 = simB
    tolerance = 1 # used when comparing similarity matrices of match pairs
    #  best_match = 0
    consensus_set = []
    best_sim = []
    # find consensus of translation between a random keypoint and the rest for
    # a number of times to find the best match regarding translation
    for i in range(100):
        idxs = random.sample(range(len(matches)), 2)
        # calc similarity between kp1 and kp2
        kp11 = kpts1[matches[idxs[0]].queryIdx]
        kp12 = kpts2[matches[idxs[0]].trainIdx]
        kp21 = kpts1[matches[idxs[1]].queryIdx]
        kp22 = kpts2[matches[idxs[1]].trainIdx]
        A = np.array([[kp11.pt[0], -kp11.pt[1], 1, 0],
                      [kp11.pt[1], kp11.pt[0], 0, 1],
                      [kp21.pt[0], -kp21.pt[1], 1, 0],
                      [kp21.pt[1], kp21.pt[0], 0, 1]])
        b = np.array([kp12.pt[0], kp12.pt[1], kp22.pt[0], kp22.pt[1]])
        Sim,_,_,_ = np.linalg.lstsq(A, b)
        Sim = np.array([[Sim[0], -Sim[1], Sim[2]],
                        [Sim[1], Sim[0], Sim[3]]])
        temp_consensus_set = []
        for j in range(len(matches) - 1):
            match = matches[j]
            kp11 = kpts1[matches[j].queryIdx]
            kp12 = kpts2[matches[j].trainIdx]
```

```
        kp21 = kpts1[matches[j+1].queryIdx]
        kp22 = kpts2[matches[j+1].trainIdx]
        A = np.array([[kp11.pt[0], -kp11.pt[1], 1, 0],
                  [kp11.pt[1], kp11.pt[0], 0, 1],
                  [kp21.pt[0], -kp21.pt[1], 1, 0],
                  [kp21.pt[1], kp21.pt[0], 0, 1]])
        b = np.array([kp12.pt[0], kp12.pt[1], kp22.pt[0], kp22.pt[1]])
        Sim2,_,_,_ = np.linalg.lstsq(A, b)
        Sim2 = np.array([[Sim2[0], -Sim2[1], Sim2[2]],
                    [Sim2[1], Sim2[0], Sim2[3]]])
        if (np.array(np.abs(Sim-Sim2)) < tolerance).all():
          temp_consensus_set.append(j)
          temp_consensus_set.append(j+1)
      if len(temp_consensus_set) > len(consensus_set):
        consensus_set = temp_consensus_set
        #  best_match = idxs
        best_sim = Sim
    consensus_matches = np.array(matches)[consensus_set]
    matched_image = np.array([])
    # draw matches with biggest consensus
    matched_image = cv2.drawMatches(img1, kpts1, img2, kpts2, consensus_matches[:100],
                            flags=2, outImg=matched_image)
    print('Best match:\nSim=\n%s\n with consensus=%d or %d%%'%(
      best_sim, len(consensus_set)/2, 100*len(consensus_set)/2/len(matches)))
    return matched_image, best_sim
```

ransac_sim_affine.py:

```
import cv2
import numpy as np
import random
from get_matches import *

def ransac_sim_affine(simA, simB):
    img1, img2, kpts1, kpts2, matches = get_matches(simA, simB)
    tolerance = 1 # used when comparing similarity matrices of match pairs
    consensus_set = []
    best_sim = []
    # find consensus of translation between a random keypoint and the rest for
    # a number of times to find the best match regarding translation
    for i in range(100):
      idxs = random.sample(range(len(matches)), 3)
      # calc similarity between kp1, kp2 and kp3
      kp11 = kpts1[matches[idxs[0]].queryIdx]
      kp12 = kpts2[matches[idxs[0]].trainIdx]
      kp21 = kpts1[matches[idxs[1]].queryIdx]
      kp22 = kpts2[matches[idxs[1]].trainIdx]
      kp31 = kpts1[matches[idxs[2]].queryIdx]
      kp32 = kpts2[matches[idxs[2]].trainIdx]
      #  A = np.array([[kp11.pt[0], kp11.pt[1], 1, 0, 0, 0],
                #  [0, 0, 0, kp11.pt[0], kp11.pt[1], 1],
                #  [kp21.pt[0], kp21.pt[1], 1, 0, 0, 0],
```

Kunal Goyal
CS558 – Computer Vision
Homework3

```
            #  [0, 0, 0, kp21.pt[0], kp21.pt[1], 1],
            #  [kp31.pt[0], kp31.pt[1], 1, 0, 0, 0],
            #  [0, 0, 0, kp31.pt[0], kp31.pt[1], 1]])
    #  b = np.array([kp12.pt[0], kp12.pt[1], kp22.pt[0], kp22.pt[1],
            #  kp32.pt[0], kp32.pt[1]])
    #  Sim,_,_,_ = np.linalg.lstsq(A, b)
    #  Sim = Sim.reshape((2, 3))
    pts1 = np.float32([[kp11.pt[0], kp11.pt[1]],
                [kp21.pt[0], kp21.pt[1]],
                [kp31.pt[0], kp31.pt[1]]])
    pts2 = np.float32([[kp12.pt[0], kp12.pt[1]],
                [kp22.pt[0], kp22.pt[1]],
                [kp32.pt[0], kp32.pt[1]]])
    Sim = cv2.getAffineTransform(pts1, pts2)
    temp_consensus_set = []
    for j in range(len(matches)-2):
      kp11 = kpts1[matches[j].queryIdx]
      kp12 = kpts2[matches[j].trainIdx]
      kp21 = kpts1[matches[j+1].queryIdx]
      kp22 = kpts2[matches[j+1].trainIdx]
      kp31 = kpts1[matches[j+2].queryIdx]
      kp32 = kpts2[matches[j+2].trainIdx]
      pts1 = np.float32([[kp11.pt[0], kp11.pt[1]],
                  [kp21.pt[0], kp21.pt[1]],
                  [kp31.pt[0], kp31.pt[1]]])
      pts2 = np.float32([[kp12.pt[0], kp12.pt[1]],
                  [kp22.pt[0], kp22.pt[1]],
                  [kp32.pt[0], kp32.pt[1]]])
      Sim2 = cv2.getAffineTransform(pts1, pts2)
      if (np.array(np.abs(Sim-Sim2)) < tolerance).all():
        temp_consensus_set.append(j)
        temp_consensus_set.append(j+1)
        temp_consensus_set.append(j+2)
    if len(temp_consensus_set) > len(consensus_set):
      consensus_set = temp_consensus_set
      best_sim = Sim
consensus_matches = np.array(matches)[consensus_set]
matched_image = np.array([])
# draw matches with biggest consensus
matched_image = cv2.drawMatches(img1, kpts1, img2, kpts2, consensus_matches,
                    flags=2, outImg=matched_image)
print('Best match:\nSim=\n%s\n with consensus=%d or %d%%'%(
  best_sim, len(consensus_set)/3, 100*len(consensus_set)/len(matches)))
return matched_image, best_sim
```

Kunal Goyal
CS558 – Computer Vision
Homework3

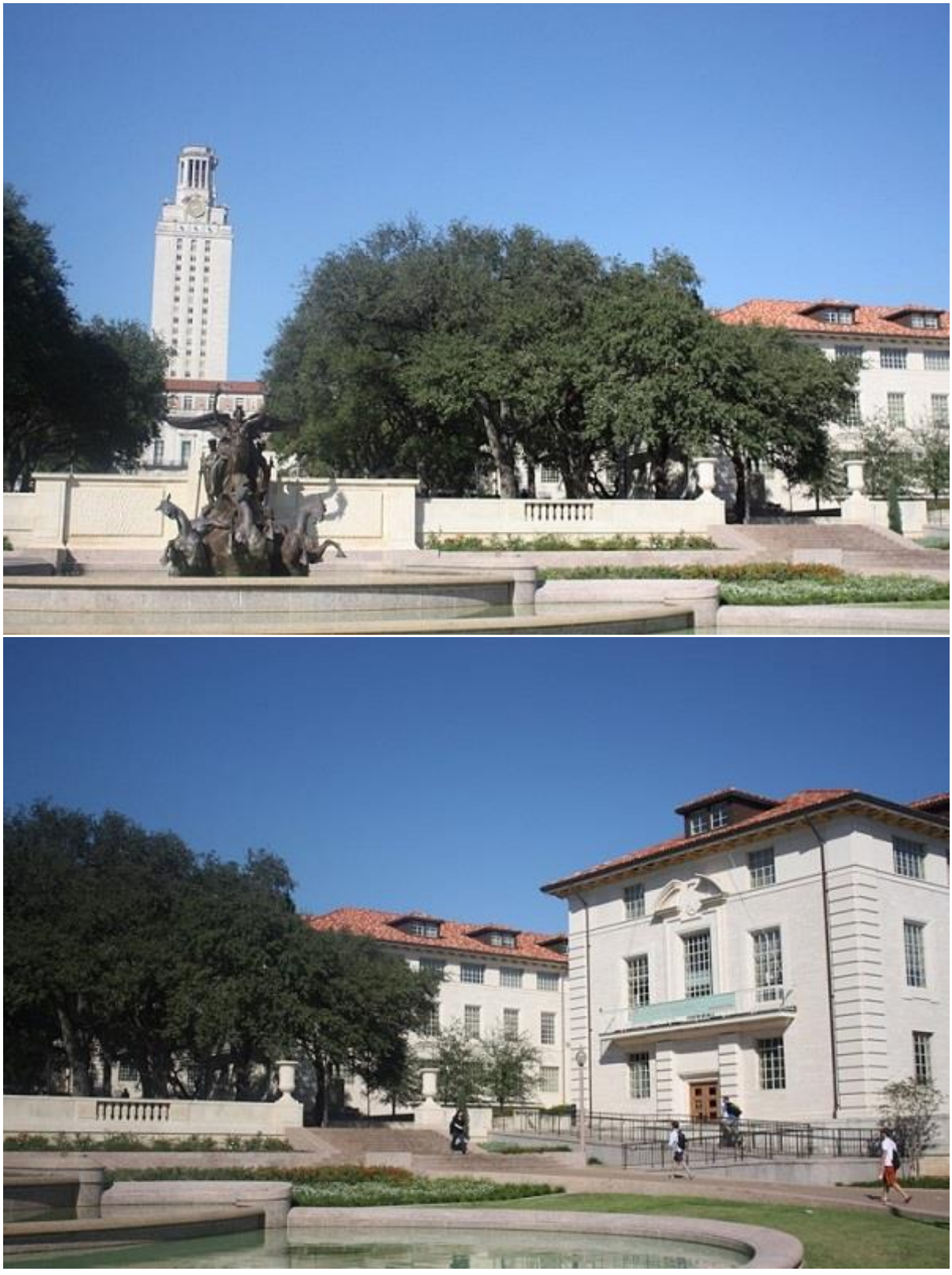
Explanation and Images:

Implement the Harris corner detector for a gray value image. You should leverage the code developed so far in the course and compute the "cornerness" response function of each point based on the second moment matrix. The final feature selection will then pick the 1000 strongest points as the features of the image. Execute this detector on greyscale versions of the two images in the TwoViewAlignment data subdirectory. Include the results in your report.

Original Image:

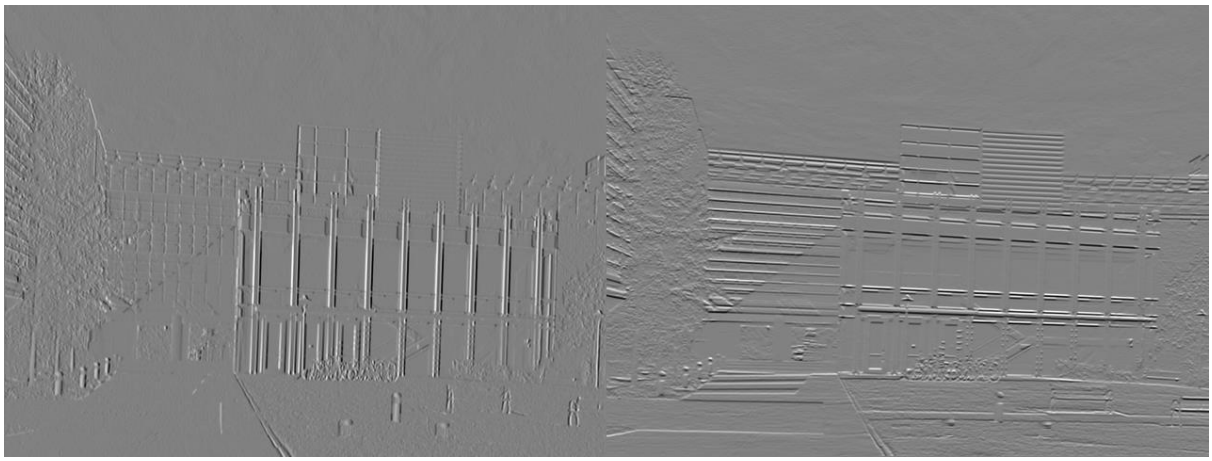
Kunal Goyal
CS558 – Computer Vision
Homework3

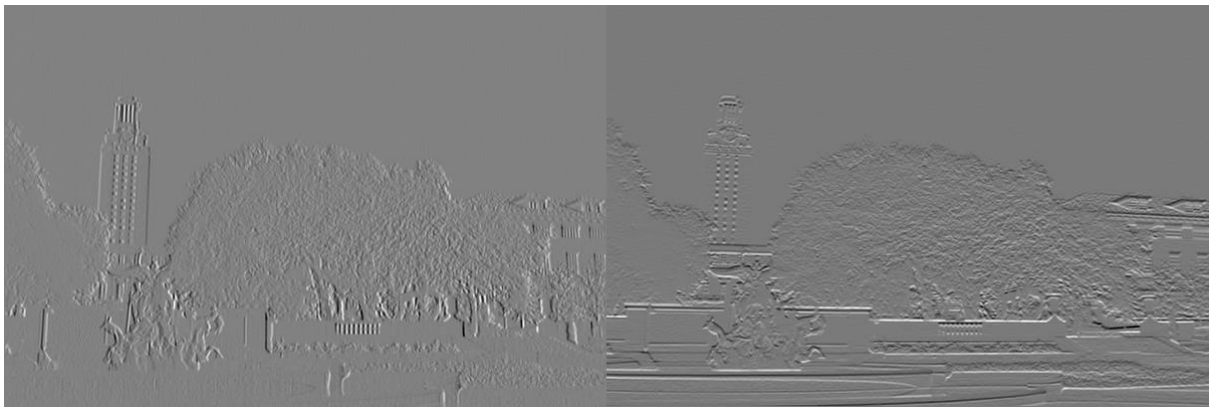
Kunal Goyal
CS558 – Computer Vision
Homework3



RGB Image:

Kunal Goyal
CS558 – Computer Vision
Homework3





Harris Corner Detector

Overview of Harris Corner Detection:

Before SIFT and SURF operators were invented, Harris Corner Detector was widely used in digital image interest points detection. The idea of Harris corner detection is based on that the characterization of a corner pixel should be invariant to rotations of images. Although scale was not introduced when Harris corner detector was first introduced, we now can implement the Harris corner detector with variable scale.

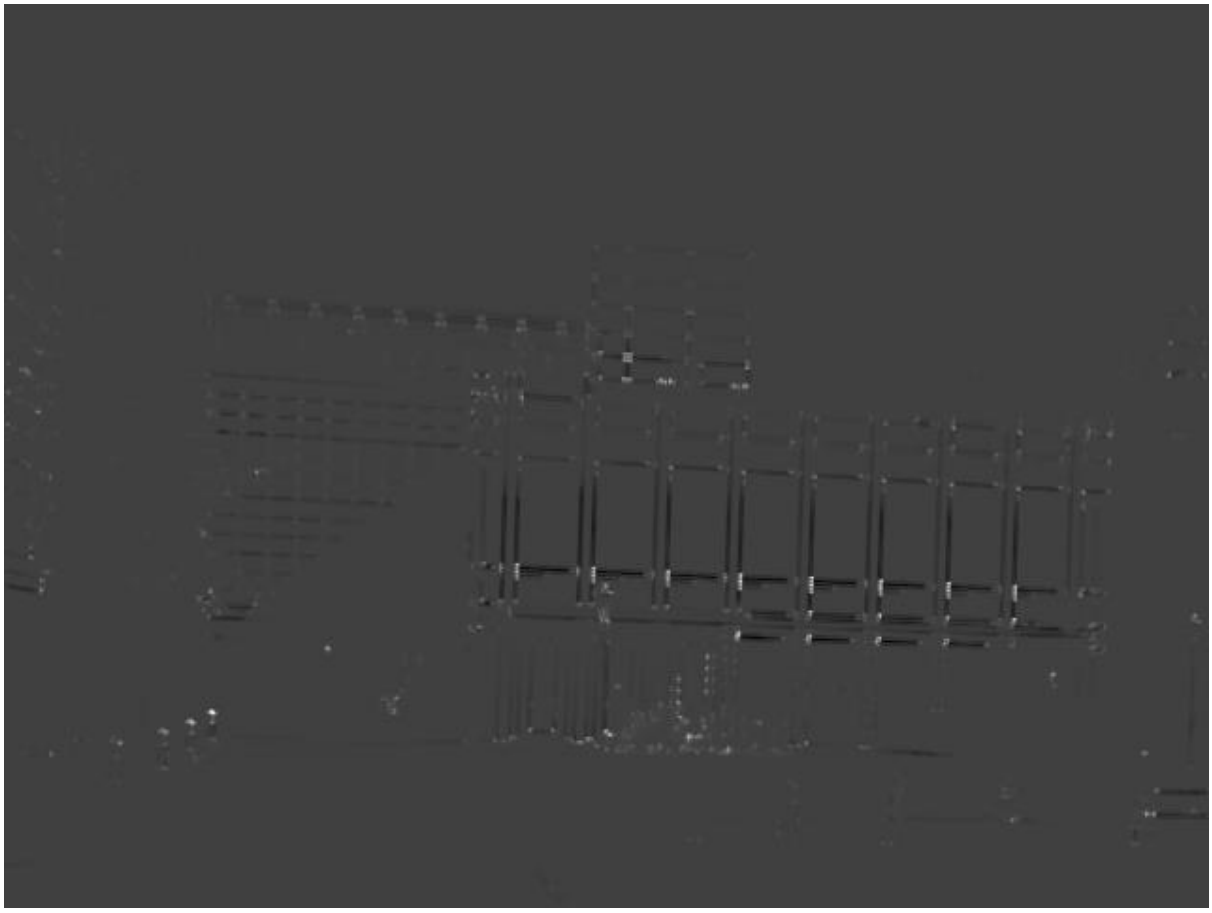
The Harris Corner Detector is a mathematical operator that finds features (what are features?) in an image. It is simple to compute and is fast enough to work on computers. Also, it is popular because it is rotation, scale and illumination variation independent. However, the Shi-Tomasi corner detector, the one implemented in OpenCV, is an improvement of this corner detector.

To define the Harris corner detector, we must go into a bit of math. We'll get into a bit of calculus, some matrix math, but trust me, it won't be tough. I'll make everything easy to understand!

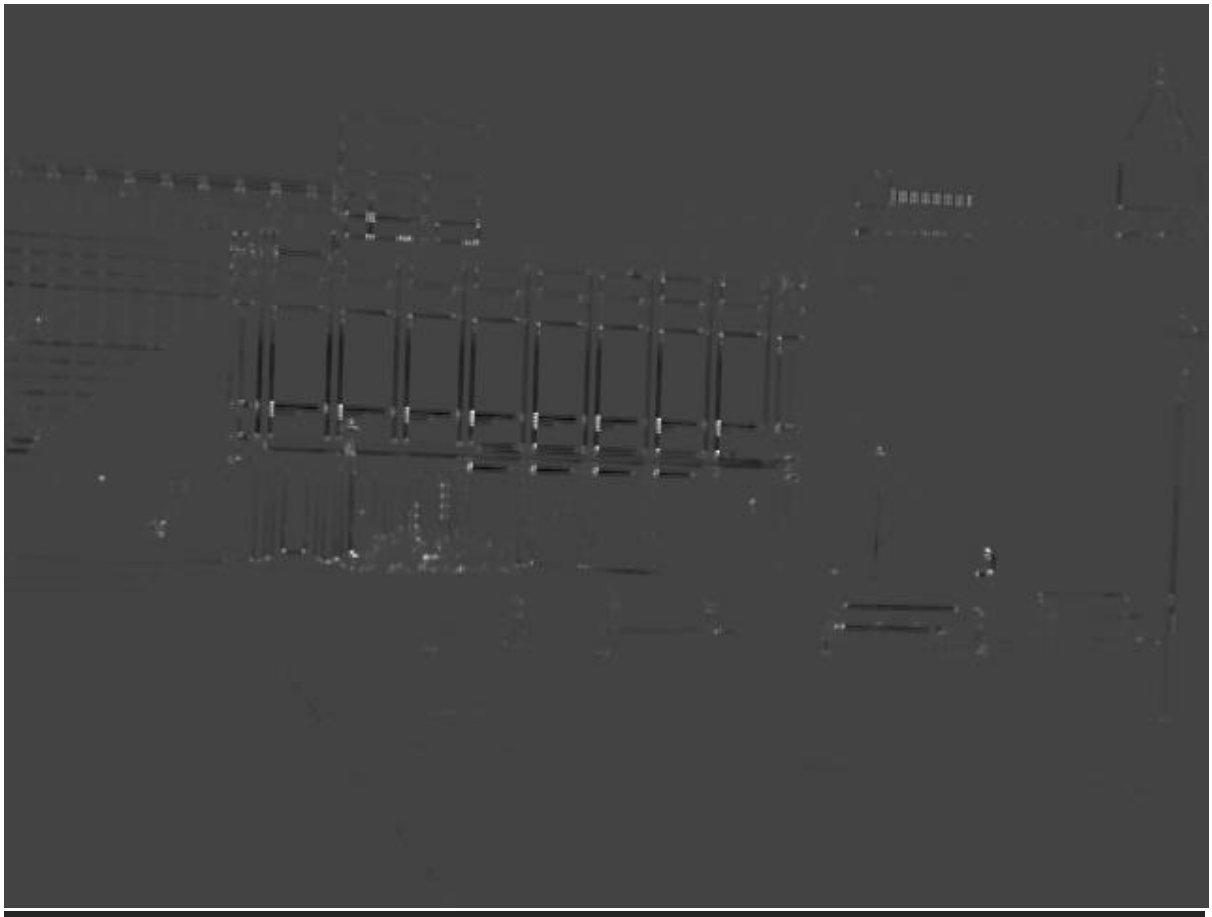
Kunal Goyal
CS558 – Computer Vision
Homework3

Our aim is to find little patches of image (or "windows") that generate a large variation when moved around. Have a look at above image, we can see the corners with highlighted part of image. All the corners can be detected so easily in the image.

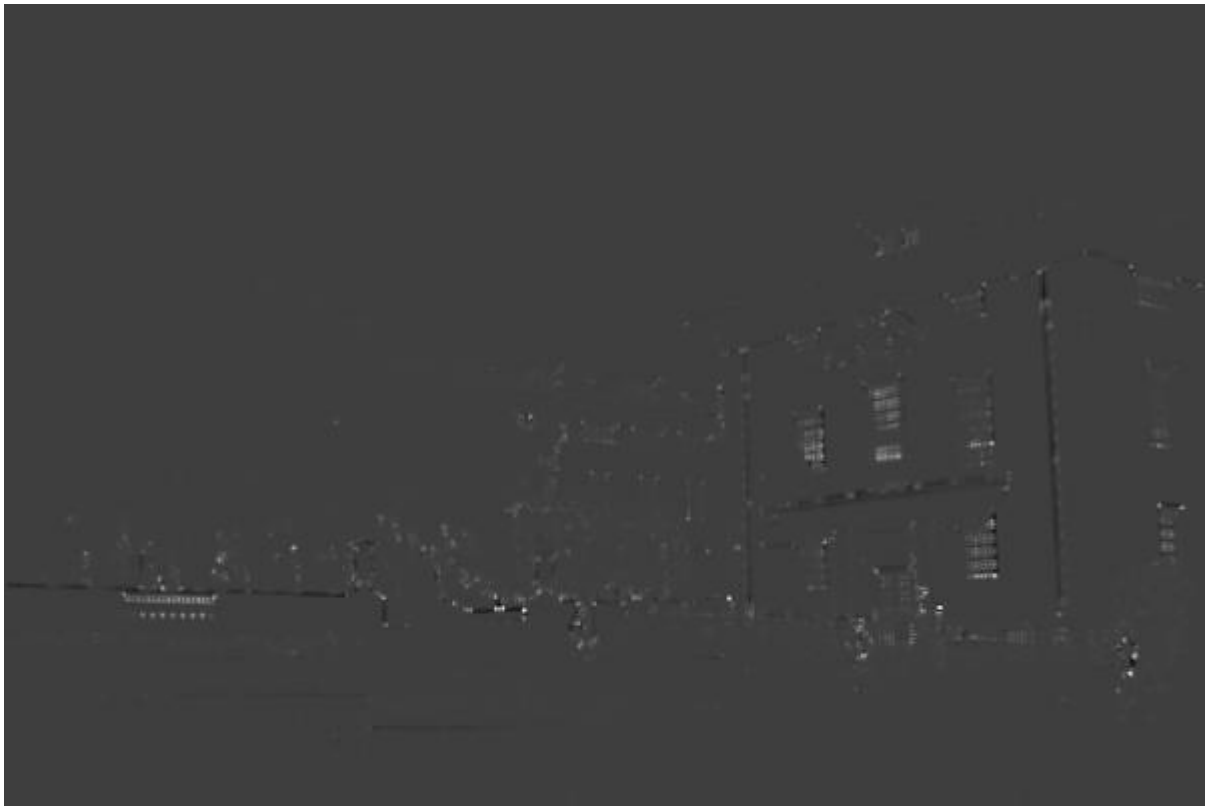
Intermediate Results: Gradient,Corners:



Kunal Goyal
CS558 – Computer Vision
Homework3



Kunal Goyal
CS558 – Computer Vision
Homework3



The x-gradient and y-gradient for pic1.jpg and pic2.jpg
This part is done in the code with the help of Harris corner module, which is developed, and we got the results as follows:

Kunal Goyal
CS558 – Computer Vision
Homework3

Kunal Goyal
CS558 – Computer Vision
Homework3

Kunal Goyal
CS558 – Computer Vision
Homework3

These 4 Images indicating the Harris corner detected as red Dot in the pictures. we used both pictures to perform this task.

Kunal Goyal
CS558 – Computer Vision
Homework3

Non-Maximum Suppression (NMS) is a technique used in numerous computer vision tasks. It is a class of algorithms to select one entity (e.g., bounding boxes) out of many overlapping entities. We can choose the selection criteria to arrive at the desired results. The criteria are most commonly some forms of probability number and some form of overlap measure (e.g., Intersection over Union).

How NMS Works:

NMS is used to make sure that in object detection, a particular object is identified only once. Consider a 100X100 image with a 9X9 grid and there is a car that we want to detect. If this car lies in multiple cells of grid, NMS ensures we identify the optimal cell among all candidates where this car belongs.

The way NMS works is:
→ It first discards all those cells where probability of object being present (calculated in final softmax layer) is <= 0.6
→ Then it takes the cell with largest probability among candidates for object as a prediction
→ Finally, we discard any remaining cell with Intersection over union value >= 0.5 with the prediction cell.

SSD matching algorithm: The SSD algorithm is like the SAD algorithm, and its formula is:

SSD (u, v) = Sum {[Left (u, v)-Right (u, v)] * [Left (u, v)-Right (u, v)]} select the maximum value

NCC matching algorithm: The NCC algorithm is to calculate the cross-correlation of the matching area of two images, and its calculation formula is:

NCC (u, v) = [(wl-w)/(| wl-w |)] * [(wr-w)/(| wr-w |)] select the maximum value

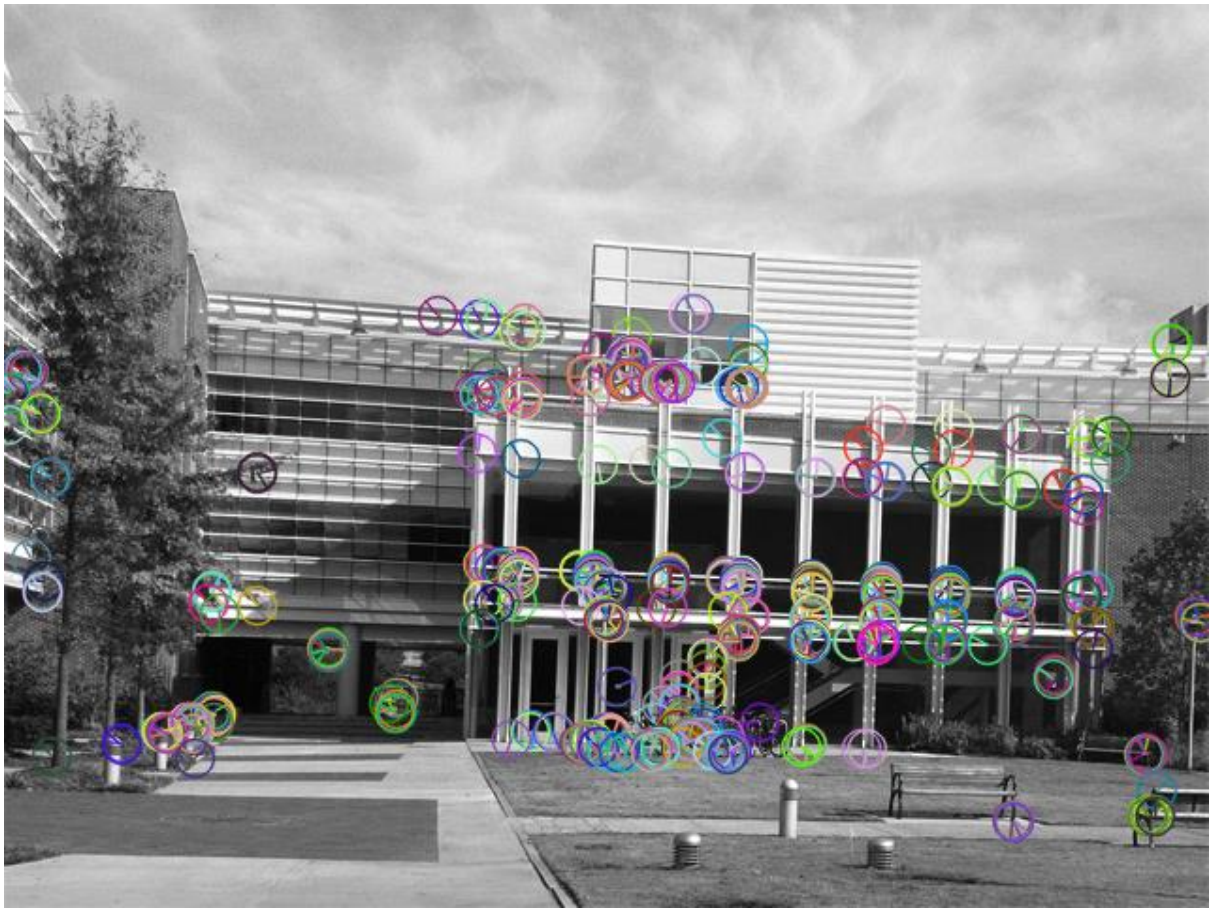
NCC is a standard method for matching two windows around a pixel of interest. The nor-malization within the window compensates differences in gain and bias. NCC is statistically the optimal method for compensating Gaussian noise. However, NCC tends to blur depth discontinuities more than many other matching costs, because outliers lead to high errors within the NCC calculation.

SSD:
1) The most popular matching score.
2) We used it when deriving Harris corners
3) T&V claim it works better than cross-correlation

Kunal Goyal
CS558 – Computer Vision
Homework3

NCC:

Highest score also coincides with correct match. Also, looks like less chances of getting a wrong match.
Important point about NCC:
Score values range from 1 (perfect match) to -1 (completely anti-correlated).
Intuition: treating the normalized patches as vectors, we see they are unit vectors. Therefore, correlation becomes dot product of unit vectors, and thus must range between -1 and 1.

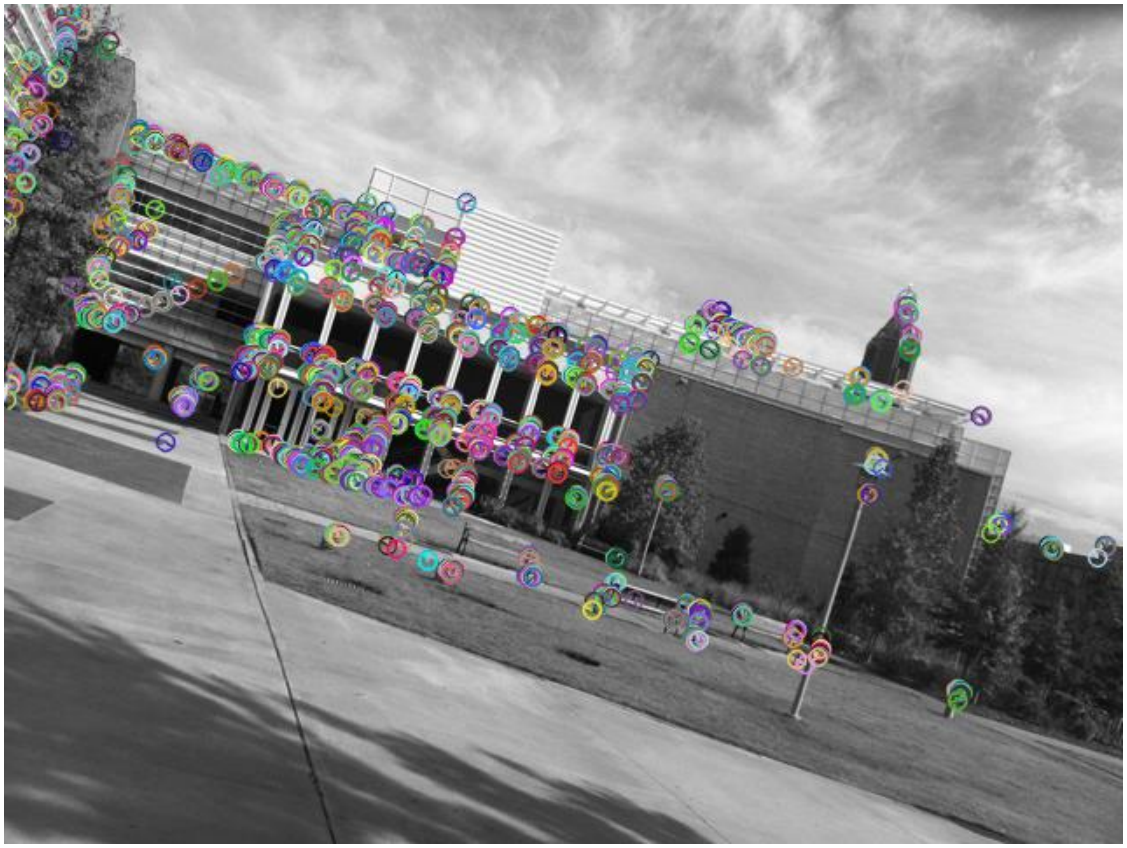


Kunal Goyal
CS558 – Computer Vision
Homework3

Kunal Goyal
CS558 – Computer Vision
Homework3

Rotate (in plane) one of the two images by 45 degrees and compare the results. Explain the difference in performance. Moreover, experiment with different rotation angles to determine what is the level of sensitivity of the similarity measures being used. Finally, discuss what can be done to remedy the situation. You may use MATLAB imrotate() function to transform one of the images

Kunal Goyal
CS558 – Computer Vision
Homework3



Here above is the result from the rotated image and we see that both the results are same. there is no other deviation even if we are using rotated image.

We are getting same corners as we are getting with the normal image. So, we can say here that these are unchanged even with the rotated image. This code is already there in the code section.

Problem2:

Using the feature detection code in the problem 1, build a set of putative feature correspondences using the following rules: a1) Selected the top 20 features based on the similarity measure score; a2) Select 30 random correspondences.

```
Sim=
[[  0.97083118  -0.2934951   41.02013423]
 [  0.2934951    0.97083118 -60.11409396]]
 with consensus=14 or 2%

Executing task: 3c
==================
Best match:
Sim=
[[  0.95391061  -0.216946    33.77327747]
 [  0.29469274   0.97299814 -62.38873371]]
 with consensus=9 or 4%
```

Kunal Goyal
CS558 – Computer Vision
Homework3

This above matrix is the result for the Problem 2a. We are getting the score as above for the random correspondences.

Implement a RANSAC-based method to estimate an affine transformation between uttower_left.jpg and uttower_right.jpg in the TwoViewAlignment directory. First, run using as input only the putative correspondences defined in a1. Second use the aggregate putative correspondences of BOTH a1 AND a2. Show the inlier matches determined by RANSAC procedure (with adaptive iteration number) and compute their average feature reprojection error between the images. Discuss what is the expected number of RANSAC iterations foe each experiment and what is the actual number observed in practice. Include the results in your report

```
Executing task: 3d
==================
Best match:
Sim=
[[  0.9770231   -0.28620124  40.6265623 ]
 [  0.28620124   0.9770231  -59.84294912]]
 with consensus=12 or 2%

Executing task: 3e
==================
Best match:
Sim=
[[  0.99362041  -0.28601808  36.29558745]
 [  0.26794258   1.01275917 -61.41467305]]
 with consensus=16 or 8%
```

When we implemented the ransac algo and tried to find same similarity measure score then we got the results as above.

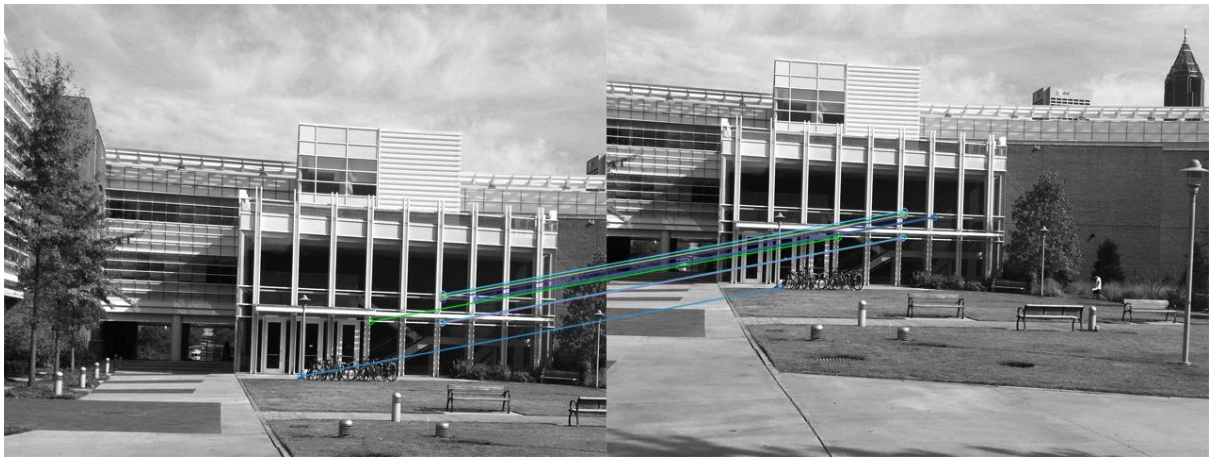
This is chosen as per different criteria which is mentioned in the problem,
This is score matrix which we got with the RANSAC-based method and having the best match for some percentage.

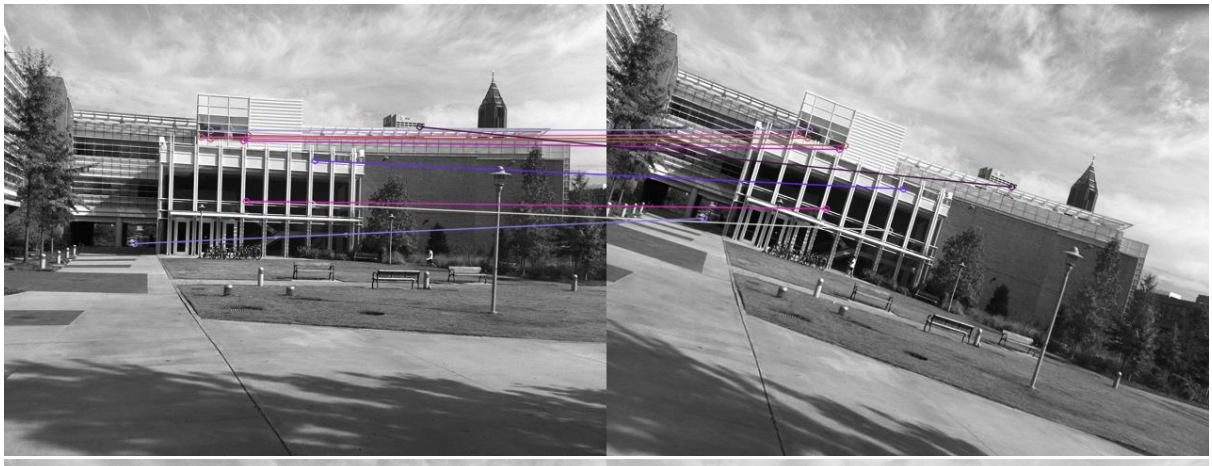
Warp one image onto the other using the affine transformation estimates. To do this, you will need to learn about MATLAB maketform and imtransform functions (or their python/C counterparts). Create a new image big enough to hold the panorama and composite the two images into it. You may composite by simply averaging the pixel values where the two images overlap. Include the results in your report.

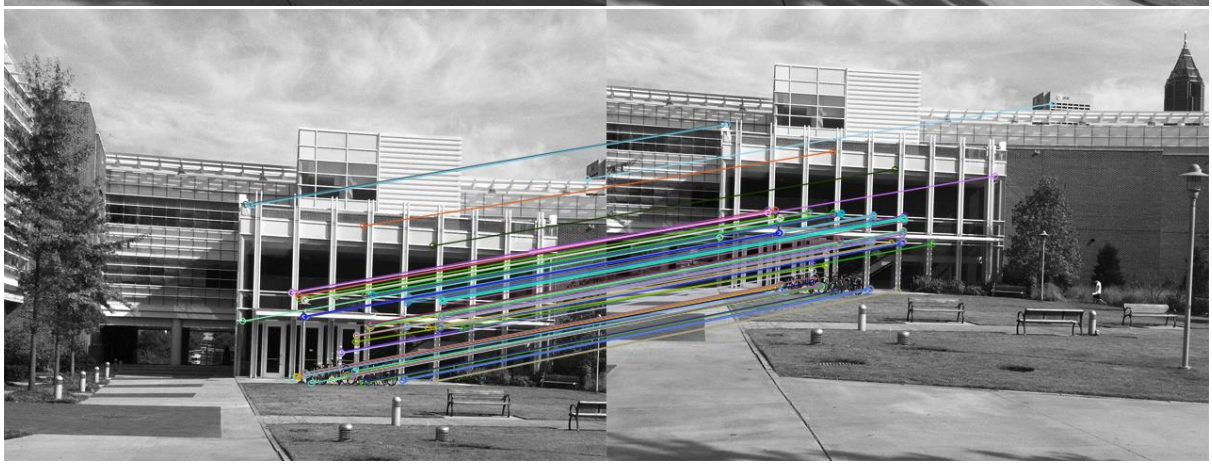
This is the panorama creation process which we have to cover. while applying above criteria we got the below results.

Harris Operator/SIFTComparison:

Kunal Goyal
CS558 – Computer Vision
Homework3







Kunal Goyal
CS558 – Computer Vision
Homework3

Conclusion for above Set :
For this set of images as the view angle (also the lighting conditions, color saturation, etc) didn't change that much, Harris corner detector works well. For the correspondences established based on SSD and NCC, except for a very few mismatch the overall correct matching rate is very high.

It can also be concluded that larger the σ, less sensitive the Harris Corner detector is (less interest points is not necessarily bad). Those points in the lower part of the image could always be detected by Harris Corner detector. As the Harris Corner detector already work well, SIFT operator would not improve our result that much. (of course, we will easily have a lot more interest points).

Dynamic Threshold for Euclidean Distance, SSD and NCC
It has already proven useful and convenient in this experiment using dynamic threshold. The idea of dynamic threshold is to avoid manually change each threshold value for every single experiment. Because the threshold values would vary hugely based on the image quality, illumination, feature descriptors strength, corner response, etc.

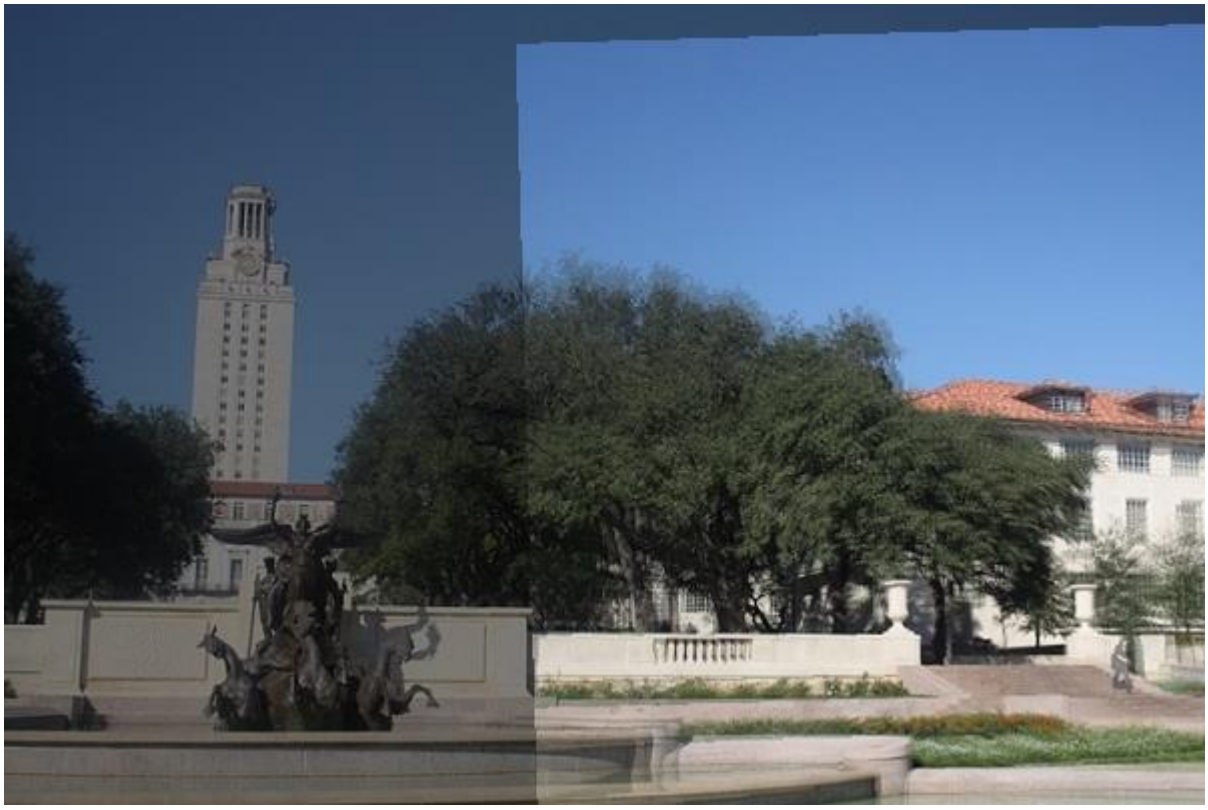
For example, in order to qualify for a match SSD value should be at least smaller than 5 times the smallest SSD value. Or, similarly, in order to qualify for a match NCC value should be at least larger than 0.9 times the largest NCC value.

Great Result: NCC for SIFT

Kunal Goyal
CS558 – Computer Vision
Homework3

From the experiment, we have concluded that when the images/features are robust, NCC based on SIFT features can still work great.

Compensate for the color variation among both images in order to mitigate the apparent "color edge" among the two images in the composite RGB image. Include the results in your report.



Kunal Goyal
CS558 – Computer Vision
Homework3

Above both the images are composite images with RGB gradient.