CSC420 Assignment 3

1. Interest point detection

(a) Harris corner detection implementation

```python
import numpy as np
from matplotlib import pyplot as plt
from skimage import io
from scipy import ndimage


def harris_corner(image, std_dev, threshold, window_size):
    """
    Given an input image, perform the Harris corner detection, and plot
    the corners as red dots.
    """
    # load grayscale image
    img = io.imread(image, as_gray=True)

    # Compute sobel gradients Ix and Iy
    Ix = ndimage.sobel(img, axis=0)
    Iy = ndimage.sobel(img, axis=1)

    # Compute Ixx, Iyy, Ix * Iy
    Ixx, Iyy, Ixy = Ix * Ix, Iy * Iy, Ix * Iy

    # Average(Gaussian) -> gives M
    M_11 = ndimage.gaussian_filter(Ixx, std_dev)
    M_12 = ndimage.gaussian_filter(Iyy, std_dev)  # M_12 = M_21
    M_22 = ndimage.gaussian_filter(Ixy, std_dev)

    # Compute "cornerness score"
    # R = det(M) - alpha * trace(M)^2 for each image window
    alpha = 0.04
    R = np.zeros(img.shape)
    for row in range(R.shape[0]):
        for col in range(R.shape[1]):
            first = M_11[row][col]
            second = third = M_12[row][col]
            last = M_22[row][col]
            M = np.array([[first, second], [third, last]])
            determinant = np.linalg.det(M)
            trace = np.matrix.trace(M)
            R[row][col] = determinant - (alpha * trace ** 2)

    # Find points with large R: R > threshold
    R[R < threshold] = 0
```

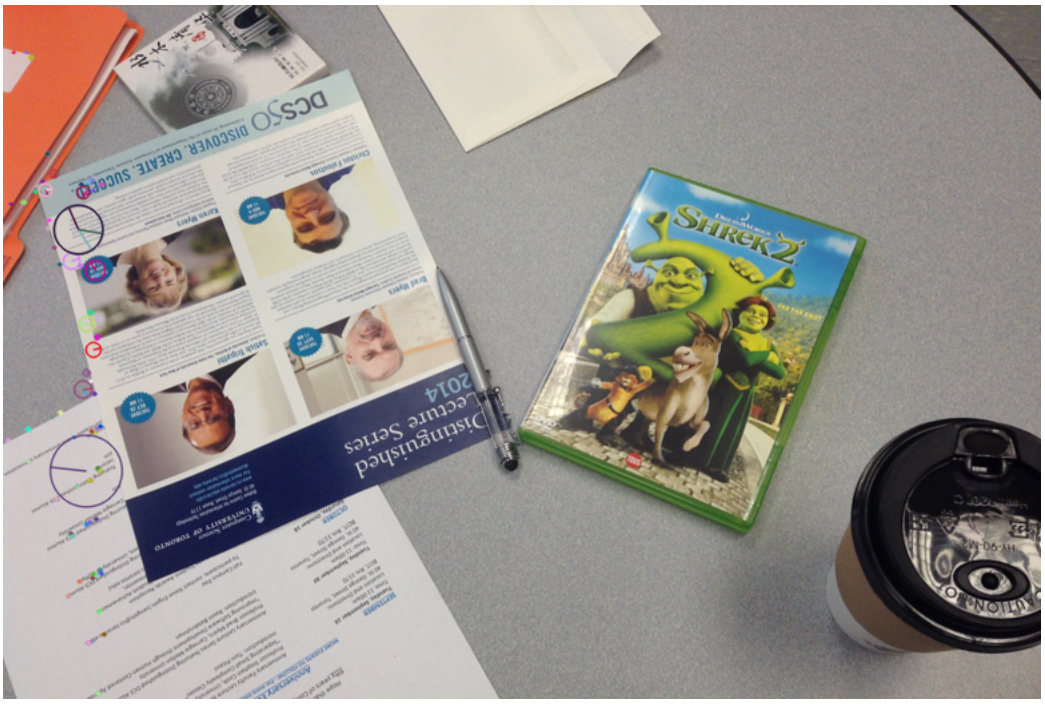
```
43
44      # Perform non-maximum suppression
45      # Take only points of local maxima
46      corners = []
47      for row in range(0, R.shape[0] - window_size, window_size):
48          for col in range(0, R.shape[1] - window_size, window_size):
49              patch = R[row:row + window_size, col:col + window_size]
50              # find max index value
51              biggest, biggest_i, biggest_j = 0, 0, 0
52              for i in range(window_size):
53                  for j in range(window_size):
54                      if patch[i][j] > biggest:
55                          biggest = patch[i][j]
56                          biggest_i, biggest_j = row + i, col + j
57              corners.append((biggest_i, biggest_j))
58
59      plt.figure(1)
60      plt.axis("off")
61      plt.imshow(img, cmap='gray')
62      for corner in corners:
63          # plot the corners with a red dot
64          plt.plot(corner[1], corner[0], 'o', markeredgecolor='r',
65                   markerfacecolor='none', markersize=1)
66      plt.show()
```

(b) Plot the result for building.jpg using
gaussian std dev sigma=0.1, window size=20, threshold=0.1

2. (a) Feature extraction

```python
import numpy as np
import cv2


def feature_extraction(image_path):
    """
    Compute SIFT features for input image and save output image with key points.
    """
    # load image
    image = cv2.imread(image_path)

    # compute key points and descriptors
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    sift = cv2.xfeatures2d.SIFT_create()
    kp, des = sift.detectAndCompute(gray, None)

    # plot 100 key points
    image_keypoints = np.copy(image)
    cv2.drawKeypoints(image, kp[:100], image_keypoints,
                      flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    # save output image with key points
    cv2.imwrite(image_path.split('.')[0] + '_keypoints.png', image_keypoints)

    return kp, des
```

Compute SIFT features and visualize the detected keypoints for reference.png, test.png, and test2.png.


DCSSO DISCOVER. CREATE. SUCCEED
Distinguished
Lecture Series
2014
SHREK 2

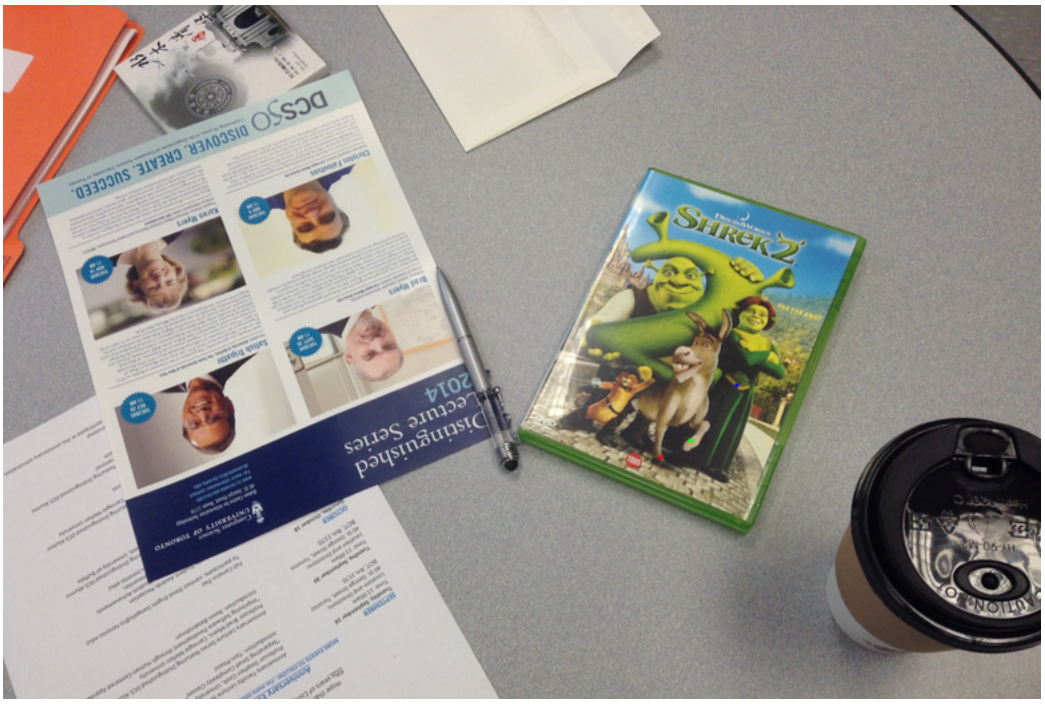

SHREK 2
LES INDESTRUCTIBLES
PAR LES CRÉATEURS DU MONDE DE NEMO

2. (b) Matching

Calculating the Euclidean distance between pairs of descriptors of keypoints as the criteria to evaluate best matches.

```
29  def match_images(img1, img2):
30      """
31      A matching algorithm to find the top 3 feature matches (correspondences)
32      in two input images.
33      """
34      kp1, des1 = feature_extraction(img1)
35      kp2, des2 = feature_extraction(img2)
36      matches = []
37      kp_diff = []
38
39      for i in range(0, len(kp1)):
40          for j in range(0, len(kp2)):
41              dist = distance.euclidean(des1[i], des2[j])
42              if dist < 100:
43                  kp_diff.append(dist)
44                  matches.append([kp1[i], kp2[j]])
45
46      top_three = np.argpartition(kp_diff, 3)[:3]
47
48      match1, match2, match3 = \
49          matches[top_three[0]], matches[top_three[1]], matches[top_three[2]]
50
51      # plot matched keypoints
52      img1_show = cv2.imread(img1)
53      img2_show = cv2.imread(img2)
54      cv2.drawKeypoints(img1_show, [match1[0]], img1_show, color=(255, 0, 0),
55                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
56      cv2.drawKeypoints(img1_show, [match2[0]], img1_show, color=(0, 255, 0),
57                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
58      cv2.drawKeypoints(img1_show, [match3[0]], img1_show, color=(0, 0, 255),
59                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
60
61      cv2.drawKeypoints(img2_show, [match1[1]], img2_show, color=(255, 0, 0),
62                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
63      cv2.drawKeypoints(img2_show, [match2[1]], img2_show, color=(0, 255, 0),
64                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
65      cv2.drawKeypoints(img2_show, [match3[1]], img2_show, color=(0, 0, 255),
66                        flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
67
68      # save the images with correspondences
69      cv2.imwrite(img1.split('.')[0] + '_matches.png', img1_show)
70      cv2.imwrite(img2.split('.')[0] + '_matches.png', img2_show)
```

Visualize results for reference.png and test.png.



Visualize results for reference.png and test2.png.

2. (c) Affine transformation

```python
def visualize_affine_trans(img1, img2):
    """
    Use the top 3 correspondences from part (b) to solve for the
    affine transformation between the features in the two images.
    """
    # top 3 correspondences from part (b)
    kp1, kp2, kp3 = match_images(img1, img2)

    # keypoints for img1
    x1, y1 = kp1[0].pt
    x3, y3 = kp2[0].pt
    x5, y5 = kp3[0].pt

    # keypoints for img2
    x2, y2 = kp1[1].pt
    x4, y4 = kp2[1].pt
    x6, y6 = kp3[1].pt

    a = np.array([x2, y2, x4, y4, x6, y6])
    A = np.array([[x1, y1, 1, 0, 0, 0], [0, 0, 0, x1, y1, 1],
                  [x3, y3, 1, 0, 0, 0], [0, 0, 0, x3, y3, 1],
                  [x5, y5, 1, 0, 0, 0], [0, 0, 0, x5, y5, 1]])
    trans = np.linalg.solve(A, a)

    img1_show = cv2.imread(img1)
    height, width = img1_show.shape[0], img1_show.shape[1]

    # apply affine transformation
    p1 = affine_transformation(0, 0, trans)
    p2 = affine_transformation(0, height, trans)
    p3 = affine_transformation(width, 0, trans)
    p4 = affine_transformation(width, height, trans)

    # visualize the affine transformation with red lines
    img2_show = cv2.imread(img2)
    plt.imshow(img2_show[:, :, ::-1])

    plt.plot((p1[0], p2[0]), (p1[1], p2[1]), c='r', linewidth=2)
    plt.plot((p1[0], p3[0]), (p1[1], p3[1]), c='r', linewidth=2)
    plt.plot((p2[0], p4[0]), (p2[1], p4[1]), c='r', linewidth=2)
    plt.plot((p3[0], p4[0]), (p3[1], p4[1]), c='r', linewidth=2)

    plt.show()


def affine_transformation(x, y, tran):
    """
    Helper function for affine transformation.
    """
    A = np.array([[x, y, 1, 0, 0, 0], [0, 0, 0, x, y, 1]])
    res = np.dot(A, tran)

    return res[0], res[1]
```

2. (d) Visualize the affine transformation