

# Introduction to Image Understanding Homework 3

Name: Anass Belcaid

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

## Problem 1

- (a) **Feature Extraction:** Write a function to extract *Sift features*.

---

```
def get_sift_features(Img):
    """
    Wrapper function to get the SIFT features on the image
    """

    if Img.ndim == 3:
        #convert the image to grayscale
        Img = cv2.cvtColor(Img, cv2.COLOR_BGR2GRAY)

        #creating the sift instance
        sift = cv2.xfeatures2d.SIFT_create()

        #getting the keypoints
        kp = sift.detect(Img, None)

        return kp

def q1():
    """
    Question 1 compute the SIFT keypoints
    """

    #filenames
    names = ["reference.png", "test.png", "test2.png"]
    sift_names = ['ref_sift.png', 'test_sift.png', 'test2_sift.png']

    #images
    Imgs = list(map(cv2.imread, names))

    for img, siftname in zip(Imgs, sift_names):
        print("Processing img : {}".format(siftname))
        kp = get_sift_features(img)
        out = img[0].copy()
        cv2.drawKeypoints(img, kp[-100:], outImage = out, flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

---

- (b) Propose a simple **matching** algorithm:

I will use the simple matching algorithm that compute the **distances** between referenced image and the template. A match is detected if the ratio between the **closest** and next **closest** is inferior to a given constant  $C$ . Also the matches will be sorted according to distances to their associated points.

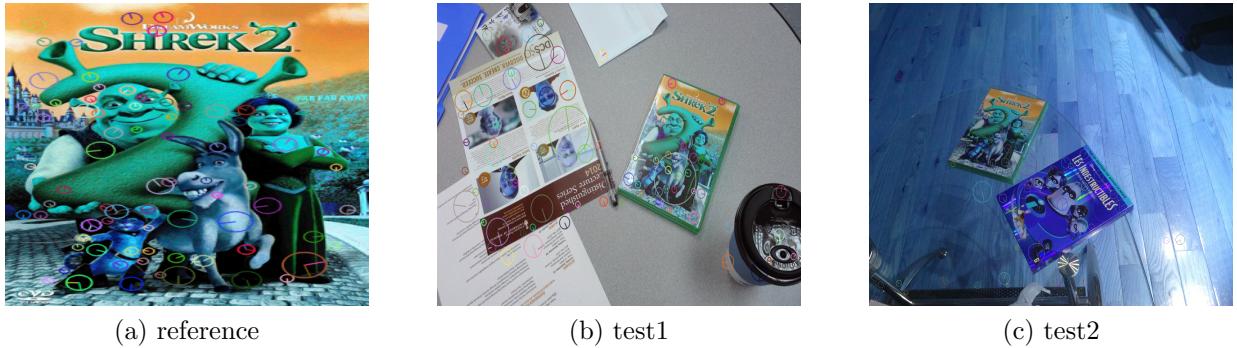


Figure 1: Features Extraction by the SIFT method

---

```

def match_descriptors(desc1, desc2, threshold=0.5):
    """
    Match the feature descriptors by finding distances between them. A match is formed
    when the distance to the closest vector is much smaller than the distance to the
    second-closest, that is, the ratio of the distances should be smaller
    than the threshold. Return the matches as pairs of vector indices.

    Args:
        desc1: an array of shape (M, P) holding descriptors of size P about M keypoints
        desc2: an array of shape (N, P) holding descriptors of size P about N keypoints

    Returns:
        matches: an array of shape (Q, 2) where each row holds the indices of one pair
        of matching descriptors
    """
    matches = []

    N = desc1.shape[0]
    dists = cdist(desc1, desc2)

    for i in range(N):
        #getting the indices for the smallest distance
        m1,m2 = np.argsort(dists[i,:])[:2]

        # computing the ratio
        ratio = dists[i,m1]/dists[i,m2]

        #adding the match
        if(ratio


---



```

(c) Find the affine transform between the matches

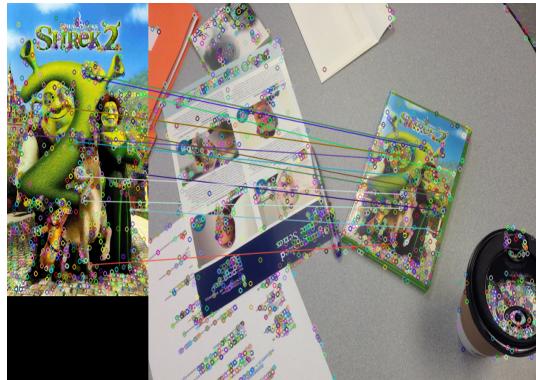
---

```

def find_affine_transform(kp1, kp2, matchs):
    """
    Function to find the affine transform between the set of kp1 and kp2
    """
    X = np.array([kp1[i].pt for i in matchs[:,0]]).T
    #padding X

```

---



(a) Test1



(b) Test2

Figure 2: Matching with distances

```

X = np.r_[X, np.ones((1,3))]
Y = np.array([kp1[i].pt for i in matchs[:,3,1]]).T
Y = np.r_[Y, np.ones((1,3))]

#compute the transformation matrix
H = np.linalg.lstsq(X,Y, rcond=None)[0]
return H

def q3():
    """
    Find affine transformation between template and figure
    """

    template = cv2.imread("./reference.png")
    kp1,desc1 = get_sift_features(template)

    #reference
    ref = cv2.imread("./test.png")
    kp2, desc2 = get_sift_features(ref)

    #matching descriptors
    matchs = match_descriptors(desc1, desc2)

    #getting the keypoints as (x, y)
    H = find_affine_transform(kp1, kp2, matchs)

    return H

```

---

(d) Image stitching:



(a) Test 1

(b) Test 2

Figure 3: Transformation by the first matches

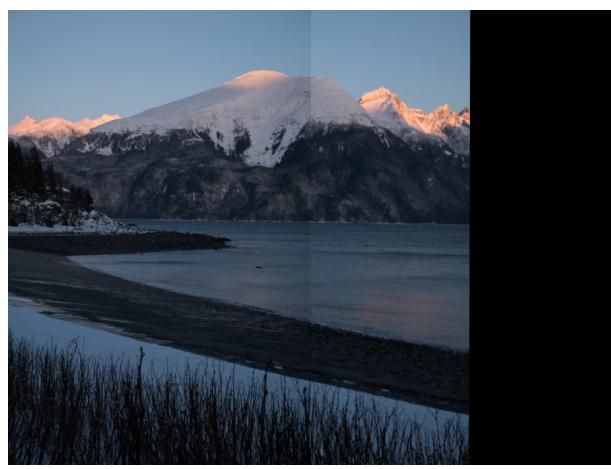


Figure 4: Stitching of the two landscape