ECE 661: Homework 2

Naveen Madapana

September 3, 2018

1 Homography estimation for projective distortion

1.1 Definition

A homography is a linear transformation that maps physical points from domain plane to a range plane. A general homography (H), represented by $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ is a 3×3 nonsingular matrix that transforms 2D physical points from one plane to the other. Homographies map straight lines to straight lines.

Let $x_1 = \begin{bmatrix} u_1 & v_1 & w_1 \end{bmatrix}^T$ and $x_2 = \begin{bmatrix} u_2 & v_2 & w_2 \end{bmatrix}^T$ be the homogeneous coordinates of two points p_1 and p_2. Suppose p_1 and p_2 represent a particular point in the real world captured by two different cameras. Then, we could estimate a homography H such that:

$$x_2 = Hx_1 \quad \longrightarrow \quad \begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix}$$

x_1 and x_2 are homogeneous coordinates. Hence, we can rewrite u_2 and v_2 as u_2/w_2 and v_2/w_2 respectively. Simplifying the above equation would give:

$$u_2 = \frac{H_{11}u_1 + H_{12}v_1 + H_{13}w_1}{H_{31}u_1 + H_{32}v_1 + H_{33}w_1}$$

$$v_2 = \frac{H_{21}u_1 + H_{22}v_1 + H_{23}w_1}{H_{31}u_1 + H_{32}v_1 + H_{33}w_1}$$

Without the loss of generality, it can be assumed that $w_1 = 1$ as we are working with homogeneous coordinates. Simplifying the equation further, we will get,

$$u_2(H_{31}u_1 + H_{32}v_1 + H_{33}) = H_{11}u_1 + H_{12}v_1 + H_{13}w_1$$

$$v_2(H_{31}u_1 + H_{32}v_1 + H_{33}) = H_{21}u_1 + H_{22}v_1 + H_{23}w_1$$

Hence, each pair of matching point will yield two equations (for x and y). We can re-arrange the equations above to get a system of linear equations of the form $Ah = 0$ where h is a 9×1 vector consisting of elements in the homography H and A is a $2m \times 9$ matrix

(m being the number matching pairs of points selected to estimate h). Let a_u and a_v be the coefficients corresponding to x and y dimensions.

$$a_u h = 0, \quad a_v h = 0$$

Where:

$$a_u = \begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u_1u_2 & v_1u_2 & u_2 \end{bmatrix}$$
$$a_v = \begin{bmatrix} 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1v_2 & v_1v_2 & v_2 \end{bmatrix}$$
$$h = \begin{bmatrix} H_{11} & H_{12} & H_{13} & H_{21} & H_{22} & H_{23} & H_{31} & H_{32} & H_{33} \end{bmatrix}^T$$

Now if we choose m pairs of points we will get the system $Ah = 0$:

$$\begin{bmatrix} -u_{11} & -v_{11} & -1 & 0 & 0 & 0 & u_{11}u_{12} & v_{11}u_{12} & u_{12} \\ 0 & 0 & 0 & -u_{11} & -v_{11} & -1 & u_{11}v_{12} & v_{11}v_{12} & v_{12} \\ -u_{21} & -v_{21} & -1 & 0 & 0 & 0 & u_{21}u_{22} & v_{21}u_{22} & u_{22} \\ 0 & 0 & 0 & -u_{21} & -v_{21} & -1 & u_{21}v_{22} & v_{21}v_{22} & v_{22} \\ & & & & \vdots & & & & \\ -u_{n1} & -v_{n1} & -1 & 0 & 0 & 0 & u_{n1}u_{n2} & v_{n1}u_{n2} & u_{n2} \\ 0 & 0 & 0 & -u_{n1} & -v_{n1} & -1 & u_{n1}v_{n2} & v_{n1}v_{n2} & v_{n2} \end{bmatrix} \begin{bmatrix} H_{11} \\ H_{12} \\ H_{13} \\ H_{21} \\ H_{22} \\ H_{23} \\ H_{31} \\ H_{32} \\ H_{33} \end{bmatrix}$$

1.2 Homography estimation

Obtaining H involves estimating the values of nine variables. Since H is represented in homogeneous coordinate system, there are only eight variables to estimate (only ratios will matter). Hence we require **atleast** four point to point correspondences in order to estimate eight variables.
SVD (Singular Value Decomposition) was used to estimate h. In SVD, we can express the matrix $A_{m\times n}$ as $A = U\Sigma V^T$ where:

- $U_{m\times m}$ is a unitary matrix
- $\Sigma_{m\times n}$ is a diagonal matrix with non-negative values
- $V^T_{n\times n}$ is also a unitary matrix

The vector h is nothing but the last column of V^T. The intuition for this is the following. The last column of V^T is the eigen vector corresponding to the smallest eigen value. We want to solve for h such that $Ah = \lambda h$, where, $\lambda = 0$. Assuming that A is positive definite (i.e. all eigen values are non negative), eigen vector with smallest eigen value minimizes the value of $|Ah|^2$

2 Tasks

2.1 Task 1

In this task, the image of Jackie Chan needs to projected on to the frame PQRS in various images. The first step was to select four points from each image to estimate the homography. Hence, the corners P, Q, R and S of each image were selected. Figure 1 shows the selected points for task 1. Overall, following steps were followed to estimate the homography.

1. Select the point to point correspondences (atleast four points on both domain and range images)
2. Create the matrix A using the the pairs of points
3. Perform SVD decomposition $[U, \Sigma, V] = svd(A)$
4. Find the last column of V^T
5. Now, we have $(h = V^T[:, end])$. Normalize $(h = h/h[end])$
6. Reshape $h_{9\times1}$ into 3×3 matrix (H), which acts the homography

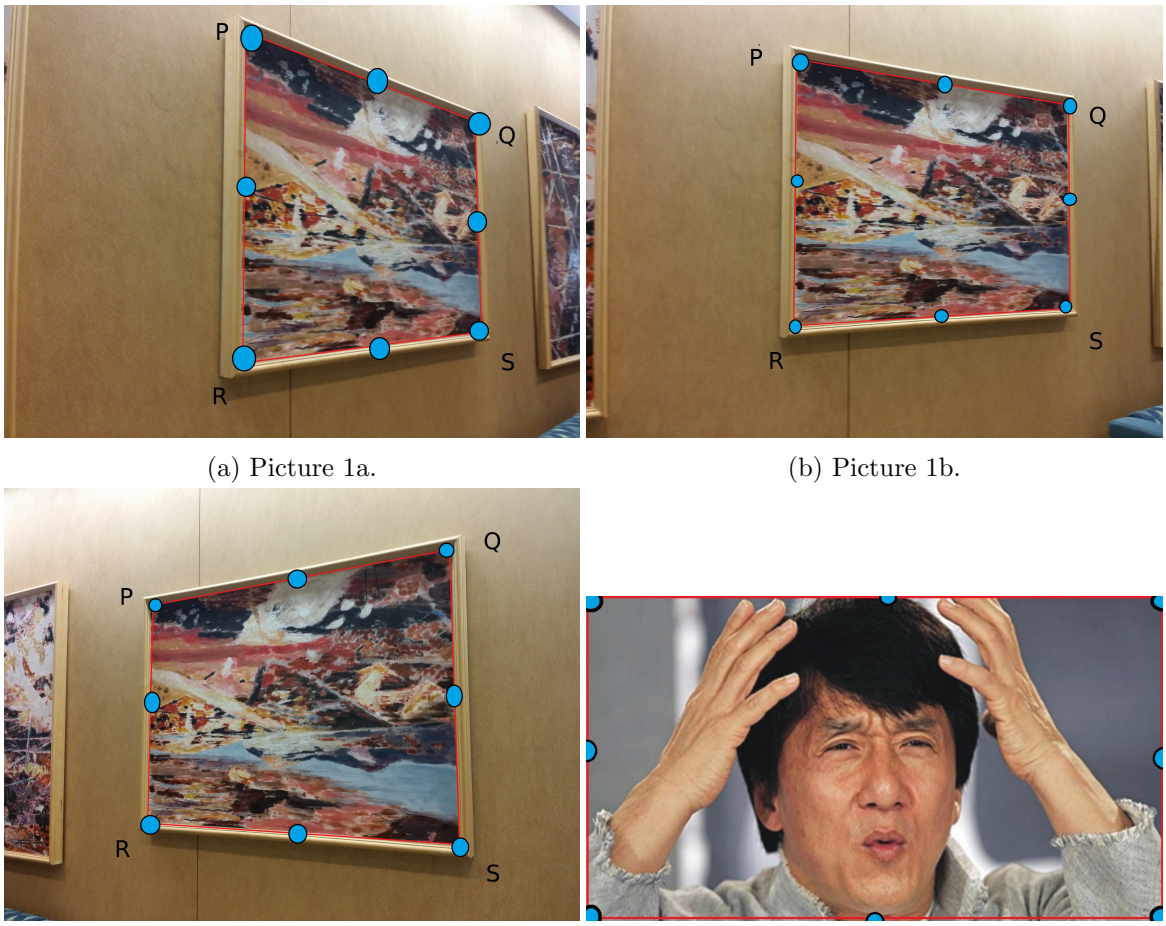
Figures 2 shows the results of projecting Jackie Chan's picture on to the given images. Four point correspondences (corner points) have been used to estimate the homography.

2.1.1 Implementation Details

The estimated homography needs to be applied to Jackie Chan's image, so that the Jackie Chan's image can fit into the frame PQRS in the original image. The idea is that every pixel in the PQRS frame has a corresponding pixel in Jackie Chan's image. We need to use the estimated homography to find this pixel to pixel correspondence. Note that, corners of Jackie Chan's image correspond to the corners of PQRS frame. First, the coordinates of pixels (x, y) inside the PQRS frame are identified in the following way.

1. Compute the homogeneous representation of the lines PQ, QS, SR, RP
2. Compute the centroid (c) of the PQRS frame
3. For each of the line i, find out 2D boolean matrix B_i (of same size as image i), where True indicates that the corresponding pixel and centroid are on the same side of the line, False indicates that they are on the opposite side of the line.
4. Find a final Boolean matrix B which is an intersection of all of the $B_i \forall i \in 1, 2, 3, 4$. The pixels that lie in inside the frame PQRS will take a value of True.
5. For each pixel coordinate that has a value of True in B, find the corresponding pixel coordinates in Jackie Chan's image using H^{-1}

6. Now, replace the intensities of pixels (in the PQRS frame) with the corresponding intensity in the Jackie Chan's image.



(a) Picture 1a.

(b) Picture 1b.

(c) Picture 1c

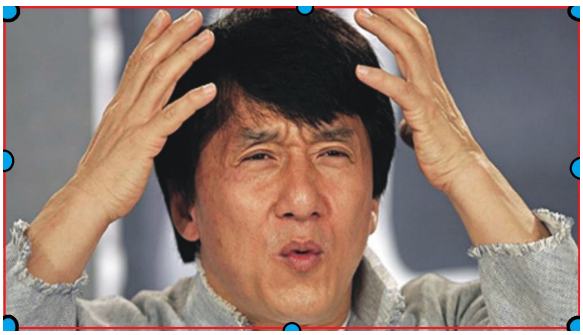
(d) Jackie Chan

Figure 1: Task 1 Pictures

(a) Picture 1a. with Jackie Chan

(b) Picture 1b. with Jackie Chan

(c) Picture 1c with Jackie Chan



(d) Jackie Chan

Figure 2: Projected Task 1 Pictures

2.2 Task 2

Assume that there are three images: image1, image2 and image3 (with .jpg extensions). In this task, the idea is to verify a property of homography: $H_{13} = H_{12}H_{23}$. Note that H_{ij} maps image i to image j.

Next, following homographies are computed. It was ensured that each homography is normalized (element at last row and last column is 1).

1. H_{12} : image 1 to image 2
2. H_{23} : image 2 to image 3
3. H_{13} : image 1 to image 3
4. $\tilde{H}_{13} = H_{12}H_{23}$

Next, the homographies are applied in the following way:

1. H_{13} applied to image 1
2. $\tilde{H}_{13}$ applied to image 1
3. H_{13} was applied to image 1. H_{23} was applied to the resulting image.

2.2.1 Implementation Details

Let I_1, I_2 and I_3 be the RGB images of a particular scene (PQRS frame) from three distinct perspectives. Though the images are three dimensional, let $I_1(i, j)$ imply the three intensities each corresponding to R, G and B channels. Also, let H_{ij} be the homography that maps image i to image j.

In this task, H_{12}, H_{23} and H_{13} are estimated using singular value decomposition. Now, the objective of this task is to verify if the image obtained by transforming image 1 using H_{13} is same as the image obtained by two successive transformations (first, H_{12} and then, H_{23}). Now, let us discuss how to transform an image I using any homography H.

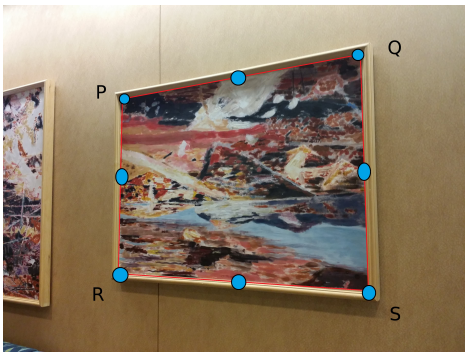
1. Let M, N be the no. of rows and columns in image I. Let (x, y) be the pixel coordinates, where $x \in 1, 2, ..., N$ and $y \in 1, 2, ..., M$
2. Let $S(I)$ be a two dimensional matrix of size $MN \times 2$. It contains the coordinates of all pixels in I
3. Let $C(S(I))$ be a two dimensional matrix of size $MN \times 3$. It contains the coordinates of all pixels in I, except that last column is filled with 1
4. Now compute, $T = (HC(S(I))^T)^T$. Normalize T by dividing all columns with the last column.
5. Now, identify the number of rows and number of columns in the resulting image $\tilde{I}$ from $S(\tilde{I})$. No. of rows $\tilde{N}$ in $\tilde{I} = max(T[:, 2]) - min(T[:, 2])$ and No. of columns $\tilde{M}$ in $\tilde{I} = max(T[:, 1]) - min(T[:, 1])$
6. Initialize the image $\tilde{I}$ of size $(\tilde{M}, \tilde{N}, 3)$. Also, compute $\tilde{H} = H^{-1}$. Make sure that $\tilde{H}$ is normalized, i.e. $\tilde{H}[3, 3] = 1$
7. Find out $C(S(\tilde{I}))$. Now, the idea is to find a corresponding pixel for each of the pixel present in $C(S(\tilde{I}))$
8. $L = (\tilde{H}C(S(\tilde{I}))^T)^T$. Divide all columns of L with the last column and convert the resulting L into integers.
9. We need to make sure that entries in L lie in the original image, i.e. we need to clip the entries accordingly. $L[:, 1] = clip(L[:, 1], min = 1, max = N)$. Similarly, $L[:, 2] = clip(L[:, 2], min = 1, max = M)$

10. , Fill the intensity values in $\tilde{I}$ using the one to one correspondence between L and $S(\tilde{I})$. $\tilde{I}[S(\tilde{I})[:,2], S(\tilde{I})[:,1], :] = I[L[:,2], L[:,1], :]$

(a) H_{13} applied to image 1

(b) $(H_{12}H_{23})$ applied to image 1

(c) H_{12} applied to image 1, and H_{23} applied to resulting image



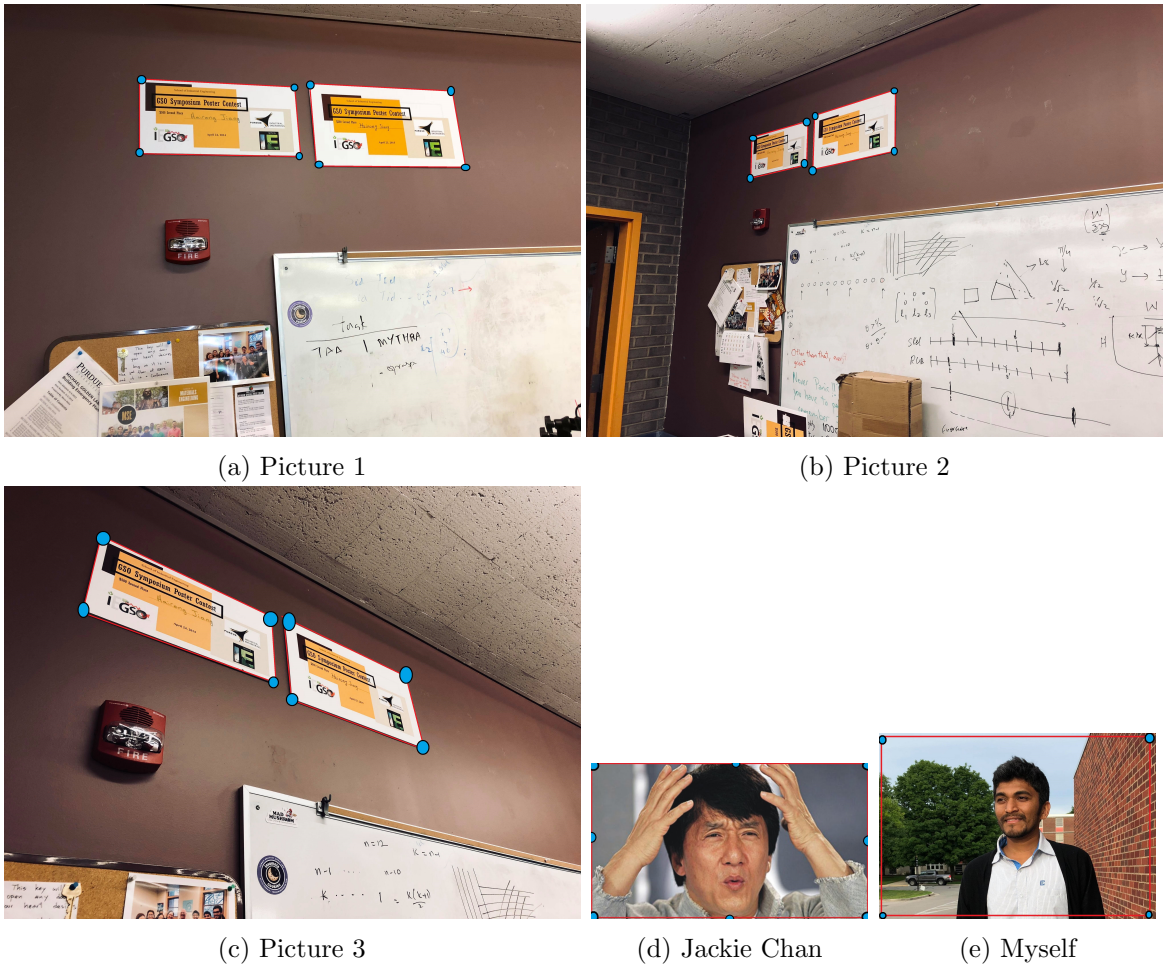
(d) Original image 3.jpg

Figure 3: Image 1 transformed to look like image 3

2.3 Task 3

In this task, the image of my own and the image of Jackie Chan are projected on to the frames in the images that were taken by me. The first step was to select four points from each picture to estimate the homographies. Hence, the corners P, Q, R and S of the frame were selected. Figure 4 shows the selected points for task 3.

Figures 5 shows the results of projecting my own picture on to the given images. Four point correspondences (corner points) have been used to estimate the homography. The implementation details of task 3 are exactly similar to task 1.



(a) Picture 1 (b) Picture 2

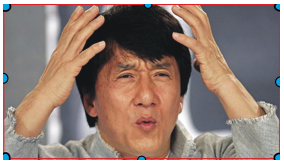
(c) Picture 3 (d) Jackie Chan (e) Myself

Figure 4: Task 3 Pictures

(a) Picture with myself and Jackie Chan

(b) Picture with myself and Jackie Chan

(c) Picture with myself and Jackie Chan



(d) Jackie Chan

(e) Myself

Figure 5: Projected Task 3 Pictures

2.4 Source Code

```
import cv2
import numpy as np
import os, time, sys

##############################
#    HELPER FUNCTIONS        #
##############################

def get_pts(shap, corners = False, H=None, targ_shap = None):
    #####
    #
```

```
    # Description:
    #
    # Input:
    #   shap: tuple (num_rows, num_cols, optional) - for given image
    #   corners: if True, only corner points, if False, all points
    #   H: 3 x 3 ndarray - given image to target image
    #   targ_shap: tuple (num_rows, num_cols, optional) - for target image
    #
    #   if H is not None, apply the homography
    #   if targ_shap is not None, clip the transformed pts accordingly.
    #
    # Return:
    #   trasformed points. In (x, y) format. It depends on if H, targ_shap are N
    #
    #####

    M, N = shap[0], shap[1]
    if(corners):
        pts = np.array([[0, 0],[N-1, 0],[N-1, M-1], [0, M-1]])
    else:
        xv, yv = np.meshgrid(range(N), range(M))
        pts = np.array([xv.flatten(), yv.flatten()]).T
    if H is None: return pts, None

    # else
    t_pts = np.dot(H, real_to_homo(pts).T)
    t_pts = homo_to_real(t_pts.T).astype(int)
    if(targ_shap is None): return pts, t_pts

    #else
    t_pts[:,0] = np.clip(t_pts[:,0], 0, targ_shap[1]-1)
    t_pts[:,1] = np.clip(t_pts[:,1], 0, targ_shap[0]-1)
    return pts, t_pts

def find_homography_2d(pts1, pts2):
    # Assertion
    assert pts1.shape[1] == 2, 'pts1 should have two columns'
    assert pts2.shape[1] == 2, 'pts2 should have two columns'
    assert pts1.shape[0] == pts2.shape[0], 'pts1 and pts2 should have same numbe

    # Forming the matrix A (8 x 9)
    A = []
```

```
    for (x1, y1), (x2, y2) in zip(pts1, pts2):
        A.append([x2, y2, 1, 0, 0, 0, -1*x1*x2, -1*x1*y2, -1*x1])
        A.append([0, 0, 0, x2, y2, 1, -1*y1*x2, -1*y1*y2, -1*y1])
    A = np.array(A)

    [U, S, V] = np.linalg.svd(A, full_matrices = True)
    h = V.T[:,-1]
    h = h / h[-1]

    ## Finding the homography. H[3,3] is assumed 1.
    # h = np.dot(np.linalg.pinv(A[:,:-1]), -1*A[:,-1])
    # h = np.append(h, 1)

    H = np.reshape(h, (3, 3))
    return H, hinv(H)

def apply_homography(img_path, H, num_partitions = 1, suff = ''):
    img = cv2.imread(img_path)
    img[0,:], img[:,0], img[-1,:], img[:,-1] = 0, 0, 0, 0
    xv, yv = np.meshgrid(range(0, img.shape[1], img.shape[1]-1), range(0, img.sh
    img_pts = np.array([xv.flatten(), yv.flatten()]).T
    trans_img_pts = np.dot(H, real_to_homo(img_pts).T)
    trans_img_pts = homo_to_real(trans_img_pts.T).astype(int)

    xmin, ymin = np.min(trans_img_pts[:,0]), np.min(trans_img_pts[:,1])
    xmax, ymax = np.max(trans_img_pts[:,0]), np.max(trans_img_pts[:,1])
    W_new = xmax - xmin
    H_new = ymax - ymin

    img_new = np.zeros((H_new+1, W_new+1, 3), dtype = np.uint8)

    x_batch_sz = int(W_new/float(num_partitions))
    y_batch_sz = int(H_new/float(num_partitions))
    for x_part_idx in range(num_partitions):
        for y_part_idx in range(num_partitions):
            x_start, x_end = x_part_idx*x_batch_sz, (x_part_idx+1)*x_batch_sz
            y_start, y_end = y_part_idx*y_batch_sz, (y_part_idx+1)*y_batch_sz
            xv, yv = np.meshgrid(range(x_start, x_end), range(y_start, y_end))
            xv, yv = xv + xmin, yv + ymin
            img_new_pts = np.array([xv.flatten(), yv.flatten()]).T
            trans_img_new_pts = np.dot(hinv(H), real_to_homo(img_new_pts).T)
            trans_img_new_pts = homo_to_real(trans_img_new_pts.T).astype(int)
```

```
            trans_img_new_pts[:,0] = np.clip(trans_img_new_pts[:,0], 0, img.shap
            trans_img_new_pts[:,1] = np.clip(trans_img_new_pts[:,1], 0, img.shap
            img_new_pts = img_new_pts - [xmin, ymin]
            # This is the bottle nect step. It takes the most time.
            img_new[img_new_pts[:,1].tolist(), img_new_pts[:,0].tolist(), :] = i

    fname, ext = tuple(os.path.basename(img_path).split('.'))
    write_filepath = os.path.join(os.path.dirname(img_path), fname+suff+'.'+ext)
    print write_filepath
    cv2.imwrite(write_filepath, img_new)

def apply_trans_patch(base_img_path, template_img_path, H, suff = '_fnew'):
    ## Read images
    base_img = cv2.imread(base_img_path)
    temp_img = cv2.imread(template_img_path)

    ## Find corners in base that correspond to corners in template
    temp_cpts, trans_temp_cpts = get_pts(temp_img.shape, corners=True, H=H, targ
    _cpts = real_to_homo(trans_temp_cpts) # homo. representation
    _cent_cpts = np.mean(_cpts, axis = 0)# centroid of four points
    # Find the four lines of quadrilateral
    lines = [np.cross(_cpts[0], _cpts[1]), np.cross(_cpts[1], _cpts[2]), np.cros

    ## Finding points in the base that are present in the quadrilateral
    base_bool = np.zeros(base_img.shape[:-1]).flatten() == 0 # True -> inside th
    base_all_pts, _ = get_pts(base_img.shape) # get all pts
    for line in lines:
        line = line / line[-1]
        sn = int(np.sign(np.dot(_cent_cpts, line)))
        nsn = np.int8(np.sign(np.dot(real_to_homo(base_all_pts), line)))
        base_bool = np.logical_and(base_bool, nsn==sn)
    base_bool = base_bool
    base_bool = np.reshape(base_bool, (base_img.shape[0], base_img.shape[1]))

    row_ids, col_ids = np.nonzero(base_bool)
    des_base_pts = np.array([col_ids, row_ids])

    trans_des_base_pts = homo_to_real(np.dot(hinv(H), real_to_homo(des_base_pts.

    # Clip the points
    trans_des_base_pts[:, 0] = np.clip(trans_des_base_pts[:, 0], 0, temp_img.sha
    trans_des_base_pts[:, 1] = np.clip(trans_des_base_pts[:, 1], 0, temp_img.sha
```

```
    base_img[des_base_pts[1].tolist(), des_base_pts[0].tolist(), :] = temp_img[t

    fname, ext = tuple(os.path.basename(base_img_path).split('.'))
    write_filepath = os.path.join(os.path.dirname(base_img_path), fname+suff+'.'
    print write_filepath
    cv2.imwrite(write_filepath, base_img)

def real_to_homo(pts):
    # pts is a 2D numpy array of size _ x 2/3
    # This function converts it into _ x 3/4 by appending 1
    if(pts.ndim == 1):
        return np.append(pts, 1)
    else:
        return np.concatenate((pts, np.ones((pts.shape[0], 1))), axis = 1)

def homo_to_real(pts):
    # pts is a 2D numpy array of size _ x 3/4
    # This function converts it into _ x 2/3 by removing last column
    if(pts.ndim == 1):
        pts = pts / pts[-1]
        return pts[:-1]
    else:
        pts = pts.T
        pts = pts / pts[-1,:]
        return pts[:-1,:].T

def save_mps(event, x, y, flags, param):
    fac, mps = param
    if(event == cv2.EVENT_LBUTTONUP):
        mps.append([int(fac*x), int(fac*y)])
        print(int(fac*x), int(fac*y))

def create_matching_points(img_path):
    npz_path = img_path[:-4] + '.npz'
    flag = os.path.isfile(npz_path)
    if(not flag):
        img = cv2.imread(img_path)
        fac = max(float(int(img.shape[1]/960)), float(int(img.shape[0]/540)))
        if(fac < 1.0): fac = 1.0
        resz_img = cv2.resize(img, None, fx=1.0/fac, fy=1.0/fac, interpolation =
        cv2.namedWindow('TempImage')
```

```
        mps = []
        cv2.setMouseCallback('TempImage', save_mps, param=(fac, mps))
        cv2.imshow('TempImage', resz_img)
        cv2.waitKey(0)
        np.savez(npz_path, mps = np.array(mps))
    return np.load(npz_path)

def nmlz(x):
    assert isinstance(x, np.ndarray), 'x should be a numpy array'
    assert x.ndim > 0 and x.ndim < 3, 'dim of x >0 and <3'
    if(x.ndim == 1 and x[-1]!=0): return x/float(x[-1])
    if(x.ndim == 2 and x[-1,-1]!=0): return x/float(x[-1,-1])
    return x

############################
#          MAIN            #
############################
## Initialization
all_img_paths = [os.path.join('.','images','1.jpg'), os.path.join('.','images','

## Task 1 and 3
img_path = all_img_paths[0]
img_path_ref = all_img_paths[-1]

pts = create_matching_points(img_path)['mps']
pts_ref = create_matching_points(img_path_ref)['mps']

## Find homography from 2 --> 1
H, _ = find_homography_2d(pts, pts_ref)

apply_trans_patch2(img_path, img_path_ref, H)


## Task 2
pts1 = create_matching_points(all_img_paths[0])['mps']
pts2 = create_matching_points(all_img_paths[1])['mps']
pts3 = create_matching_points(all_img_paths[2])['mps']

## Find homography from 2 --> 1
H12, _ = find_homography_2d(pts2, pts1) # 2 --> 1
H23, _ = find_homography_2d(pts3, pts2) # 3 --> 2
H13, _ = find_homography_2d(pts3, pts1) # 3 --> 1
```

```
_H13 = nmlz(np.dot(H12, H23))

apply_homography(all_img_paths[0], H13, num_partitions = 4, suff = '_H13')
apply_homography(all_img_paths[0], _H13, num_partitions = 4, suff = '_H13p')

apply_homography(all_img_paths[0], H12, num_partitions = 4, suff = '_H12')
apply_homography('.\\images\\1_H12.jpg', H23, num_partitions = 4, suff = '_H12_H
```