Homework 2

Fangda Li
ECE 661 - Computer Vision

September 5, 2016

1 Methodology

1.1 Obtaining Transformation Matrix

Suppose we have two images, M and N, which contain the same planar surface but from different view points, we want to find the transformation matrix, H_{MN}. Subsequently, in order to transform pixels of M to the perspective of N, H_{MN} must satisfy the following equation:

$$p_N = H_{MN} p_M \tag{1}$$

where p_M is a pixel of M in HC[1] and p_N is the corresponding pixel in the perspective of N. More specifically, equation (1) can be rewritten as the following:

$$\begin{bmatrix} x_N \\ y_N \\ z_N \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_M \\ y_M \\ 1 \end{bmatrix} \tag{2}$$

Since point pair correspondences are obtained in HC, we need to convert p_N into its HC format, p'_N, as shown in equation (3).

$$p'_N = \begin{bmatrix} x'_N \\ y'_N \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{x_N}{z_N} \\ \frac{y_N}{z_N} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{h_{11}x_M + h_{12}y_M + h_{13}}{h_{31}x_M + h_{32}y_M + 1} \\ \frac{h_{21}x_M + h_{22}y_M + h_{23}}{h_{31}x_M + h_{32}y_M + 1} \\ 1 \end{bmatrix} \tag{3}$$

The first two rows in equation (3) provide us two equations for solving H_{MN}. Since we have 8 unknowns in H_{MN}, we need at least four pairs of corresponding points, with each pair supplying two equations. Equation (4) illustrates the eight equations from four point pairs

[1] HC refers to Homogeneous Coordinate

in matrix format.

$$\begin{bmatrix} x_M^1 & y_M^1 & 1 & 0 & 0 & 0 & -x_M^1 x_N^{1'} & -y_M^1 x_N^{1'} \\ 0 & 0 & 0 & x_M^1 & y_M^1 & 1 & -x_M^1 y_N^{1'} & -y_M^1 y_N^{1'} \\ x_M^2 & y_M^2 & 1 & 0 & 0 & 0 & -x_M^2 x_N^{2'} & -y_M^2 x_N^{2'} \\ 0 & 0 & 0 & x_M^2 & y_M^2 & 1 & -x_M^2 y_N^{2'} & -y_M^2 y_N^{2'} \\ x_M^3 & y_M^3 & 1 & 0 & 0 & 0 & -x_M^3 x_N^{3'} & -y_M^3 x_N^{3'} \\ 0 & 0 & 0 & x_M^3 & y_M^3 & 1 & -x_M^3 y_N^{3'} & -y_M^3 y_N^{3'} \\ x_M^4 & y_M^4 & 1 & 0 & 0 & 0 & -x_M^4 x_N^{4'} & -y_M^4 x_N^{4'} \\ 0 & 0 & 0 & x_M^4 & y_M^4 & 1 & -x_M^4 y_N^{4'} & -y_M^4 y_N^{4'} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_M^{1'} \\ y_M^{1'} \\ x_M^{2'} \\ y_M^{2'} \\ x_M^{3'} \\ y_M^{3'} \\ x_M^{4'} \\ y_M^{4'} \end{bmatrix} \tag{4}$$

Multiplying the two sides with the inverse of the 8 by 8 matrix gives the values of elements in H_{MN}. Note that four point pairs are sufficient in our case to give good visual results.

1.2 Projecting World Plane onto Image Plane

In order to project world plane image (i.e. face image), W, onto a certain image (i.e. wall image), I, the following steps are applied:

1. Obtain the transformation matrix H_{WI} with the method described in subsection 1.1.
2. Calculate the HC of each pixel in I projected onto W, using H_{WI}^{-1}.
3. For every pixel in I whose projection falls within the boundaries of W, replace its value with the value of its corresponding pixel in W.
4. Return image I.

In cases where the projected pixel coordinates are not integers, use the value of the pixel's nearest neighbor instead. The coordinates of its nearest neighbor are obtained by rounding the non-integer coordinates.

1.3 Projecting Image Plane onto World Plane

In order to project a certain image, I, back onto world plane, W, the following steps are applied:

1. Obtain the transformation matrix H_{IW} with the method described in subsection 1.1.
2. Since the transformed image might not be a rectangle, compute the bounding box of I after being transformed onto W.
3. Use the bounding box to initiate an new blank image N on W plane.
4. Calculate the HC of each pixel in N projected back onto I, using H_{IW}^{-1}.
5. For every pixel in N whose projection falls within the boundaries of I, replace its value with the value of its corresponding pixel in I.

6. For every pixel in N whose projection does not fall within the boundaries, replace its value with the background color value.

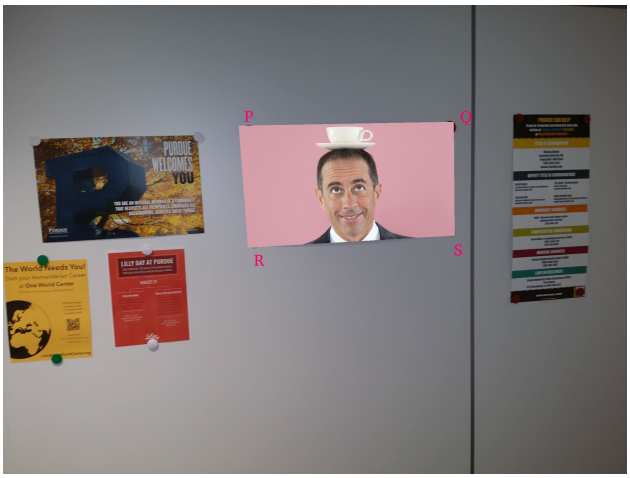
7. Return image N

In cases where the projected pixel coordinates are not integers, use the value of the pixel's nearest neighbor instead. The coordinates of its nearest neighbor are obtained by rounding the non-integer coordinates.

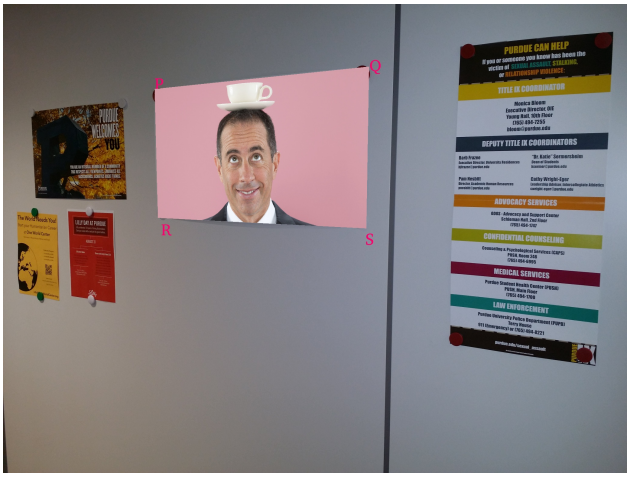
2 Results

2.1 Task 1

(a) Projecting 1(d) onto 1(a)



(b) Projecting 1(d) onto 1(b)



(c) Projecting 1(d) onto 1(c)

2.2 Task 2

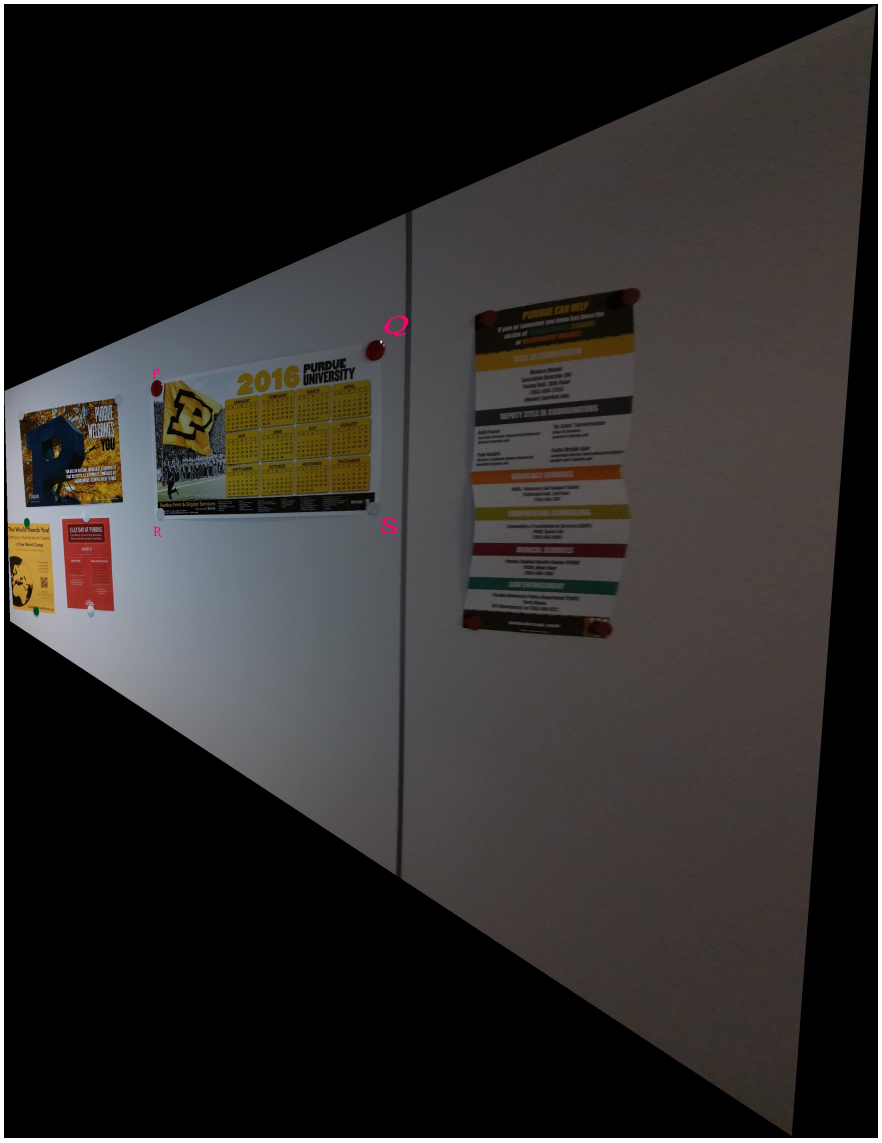
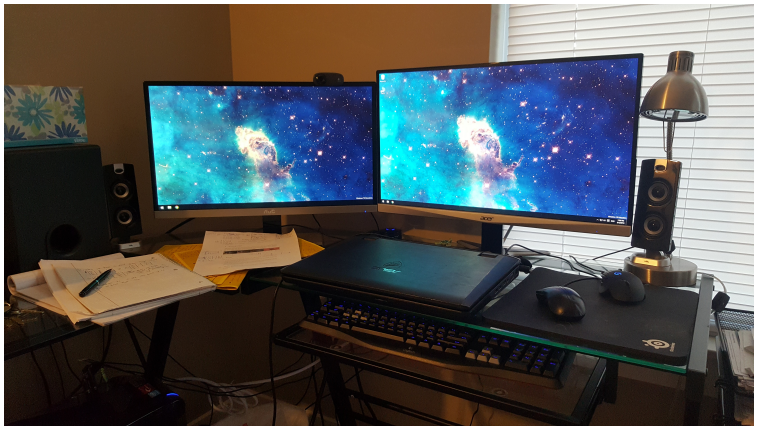
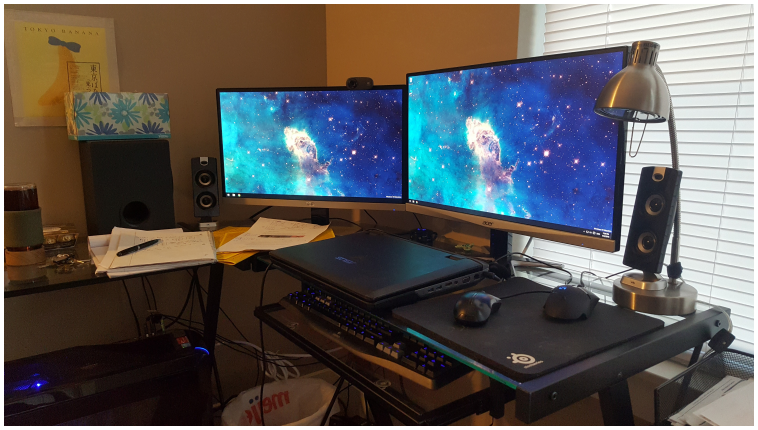


Figure 2: Transforming 1(a) to the perspective of 1(c)

2.3 Task 3



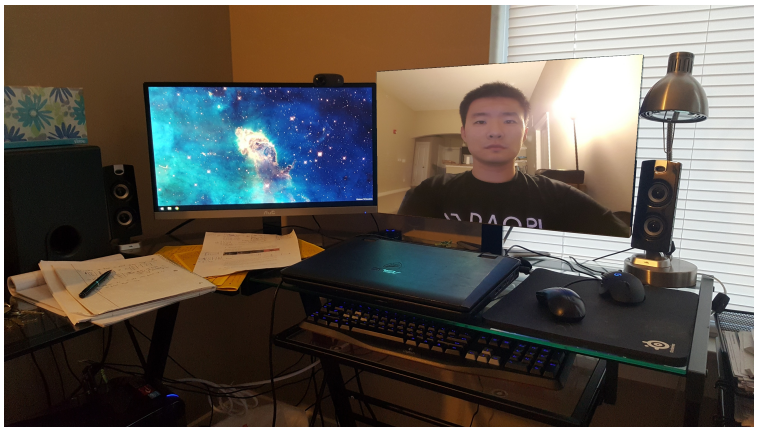
(a) View 1



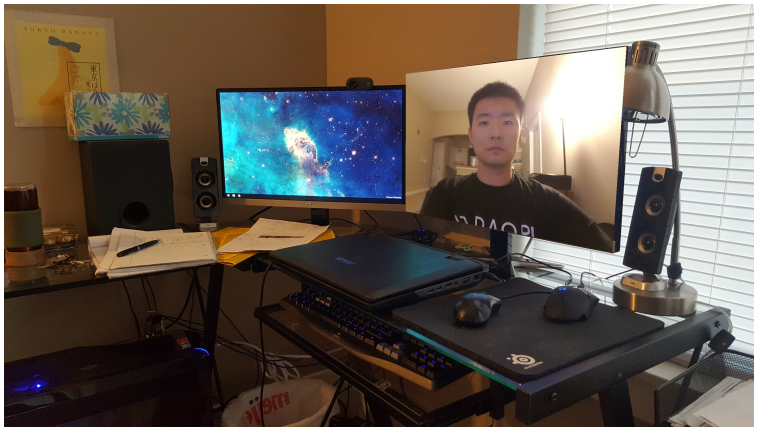
(b) View 2

(c) Me

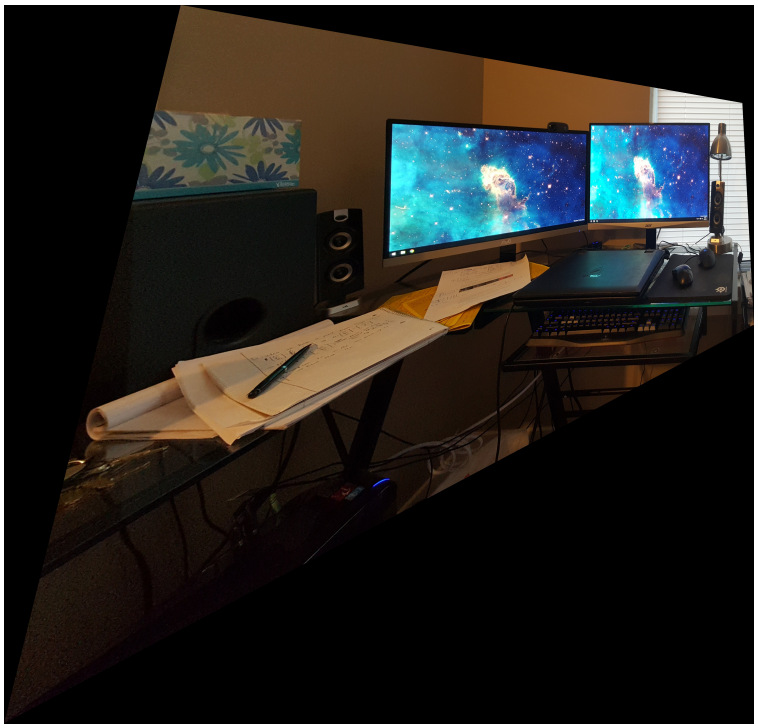
Figure 3: My own images



(a) Result 1



(b) Result 2



(c) Result 3

Figure 4: Task 3

3 Source Code

```
#!/usr/bin/python
import numpy as np
import cv2

RESIZE_RATIO = 0.5
BACKGROUND_COLOR = (0, 0, 0)
WORLD_HC = [(0,0), (0,2560), (1536,0), (1536,2560)]
IMAGE_HC_A = [(428,2110), (541,3306), (1496,2150), (1359,3309)]
IMAGE_HC_B = [(796,1558), (771,2983), (1602,1620), (1523,2988)]
IMAGE_HC_C = [(560,1002), (412,2422), (1413,1024), (1482,2404)]
IMAGE_HC_D = [(472,2637), (348,4526), (1468,2646), (1644,4426)]
IMAGE_HC_E = [(492,2846), (300,4408), (1464,2842), (1756,4308)]
IMAGE_HC_F = [(0,0), (0,2591), (1943,0), (1943,2591)]

def get_transformation_matrix(world_hc, image_hc):
        '''
                Given corresponding HC points in world view and image view,
                find the homography matrix between two coordinate systems.
                If sucessful, return a 3x3 numpy matrix
                @world_hc: list of HC point tuples
                @image_hc: list of HC point tuples
                @return: 3x3 numpy matrix H
        '''
        # We need at least for point pairs
        if len(world_hc) != 4 or len(image_hc) != 4:
                print "ERROR: not enough HC point pairs!"
                return None
        # Ah = c --> h = inv(A)c
        A = np.matrix( np.zeros((8,8)) )
        c = np.matrix( np.zeros((8,1)) )
        for i in range( len(world_hc) ):
                (x_w, y_w) = world_hc[i]
                (x_ip, y_ip) = image_hc[i]
                A[2*i] = [x_w, y_w, 1, 0, 0, 0, -x_w*x_ip, -y_w*x_ip]
                A[2*i+1] = [0, 0, 0, x_w, y_w, 1, -x_w*y_ip, -y_w*y_ip]
                c[2*i] = x_ip
                c[2*i+1] = y_ip
        try:
                h = A.I * c
        except np.linalg.LinAlgError:
                print "ERROR: A is singular!"
                return None
        H = np.concatenate( (h,[[1]]), 0)
```

```
        return H.reshape(3,3)

def project_world_into_image(world_img, image_img, H):
        '''
                Project world_img into image_img, given their homography H.
                @world_img: np.ndarray of the image to be projected
                @image_img: np.ndarray of the image to be projected on
                @H: the transformation matrix from world plane to image plane
                @return: np.ndarrary of the image with projection
        '''
        proj_img = image_img.copy()
        for r in range(image_img.shape[0]):
                for c in range(image_img.shape[1]):
                        p_i = np.array([[r,c,1]])
                        (r_w, c_w, z_w) = H.I * p_i.T
                        r_w = r_w / z_w
                        c_w = c_w / z_w
                        if 0 <= r_w < world_img.shape[0]-1 and 0 <= c_w < world_img.shap
                                proj_img[r][c] = get_pixel_by_nearest_neighbor(world_img
        return proj_img

def transform_image_into_image(image_img_1, H):
        '''
                Transform image_img_1 into the view of image_img_2, given their homogr
                @image_img_1: np.ndarray of the image to be tranformed
                @H: the transformation matrix
                @return: np.ndarrary of the transformed image
        '''
        (num_row, num_col, off_row, off_col) = get_bounding_box_after_transformation(ima
        trans_img = np.ndarray( (num_row, num_col, 3) )
        for r in range(trans_img.shape[0]):
                for c in range(trans_img.shape[1]):
                        p_t = np.array([[r+off_row,c+off_col,1]])
                        (r_1, c_1, z_1) = H.I * p_t.T
                        r_1 = r_1 / z_1
                        c_1 = c_1 / z_1
                        if 0 <= r_1 < image_img_1.shape[0]-1 and 0 <= c_1 < image_img_1.
                                trans_img[r][c] = get_pixel_by_nearest_neighbor(image_im
                        else:
                                trans_img[r][c] = BACKGROUND_COLOR
        return trans_img

def get_pixel_by_nearest_neighbor(image, row_f, col_f):
        '''
                Get the pixel value based on float row and column numbers.
```

```
                @image: image to be find pixels in
                @row_f, col_f: float row and column numbers
                @return: pixel value from image
        '''
        row = int(round(row_f))
        col = int(round(col_f))
        return image[row][col]

def get_bounding_box_after_transformation(image, H):
        '''
                Given an image and the transformation matrix to be applied,
                calculate the bounding box of the image after transformation.
                @image: image to transform
                @H: transformation matrix to be applied
                @return: (num_row, num_col, off_row, off_col)
        '''
        (h, w, c) = image.shape
        corners_1 = [(0,0), (0,w), (h,0), (h,w)]
        corners_2_row = []
        corners_2_col = []
        for corner in corners_1:
                (r,c) = corner
                p_1 = np.array([[r,c,1]])
                (r_2, c_2, z_2) = H * p_1.T
                corners_2_row.append( int(r_2 / z_2) )
                corners_2_col.append( int(c_2 / z_2) )
        return (max(corners_2_row)-min(corners_2_row)+1, max(corners_2_col)-min(corners_
                        min(corners_2_row), min(corners_2_col))


def resize_image_by_ratio(image, ratio):
        '''
                Resize an image by a given ratio, used for faster debug.
        '''
        return cv2.resize(image, (int(image.shape[1]*ratio),int(image.shape[0]*ratio)))

def task1():
        '''
                Transform a front view image onto another image.
        '''
        world_img = cv2.imread('images/Seinfeld.jpg')
        image_img_a = cv2.imread('images/1.jpg')
        image_img_b = cv2.imread('images/2.jpg')
        image_img_c = cv2.imread('images/3.jpg')
```

```
    world_img = resize_image_by_ratio(world_img, RESIZE_RATIO)
    image_img_a = resize_image_by_ratio(image_img_a, RESIZE_RATIO)
    image_img_b = resize_image_by_ratio(image_img_b, RESIZE_RATIO)
    image_img_c = resize_image_by_ratio(image_img_c, RESIZE_RATIO)

    world_hc = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in WORLD_HC]
    image_hc_a = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC
    image_hc_b = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC
    image_hc_c = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC

    H_wa = get_transformation_matrix(world_hc, image_hc_a)
    H_wb = get_transformation_matrix(world_hc, image_hc_b)
    H_wc = get_transformation_matrix(world_hc, image_hc_c)

    proj_img = project_world_into_image(world_img, image_img_a, H_wa)
    cv2.imwrite('images/task1_a.jpg', proj_img)
    proj_img = project_world_into_image(world_img, image_img_b, H_wb)
    cv2.imwrite('images/task1_b.jpg', proj_img)
    proj_img = project_world_into_image(world_img, image_img_c, H_wc)
    cv2.imwrite('images/task1_c.jpg', proj_img)

def task2():
    '''
        Cascaded transformations.
    '''
    image_img_a = cv2.imread('images/1.jpg')
    image_img_b = cv2.imread('images/2.jpg')
    image_img_c = cv2.imread('images/3.jpg')

    image_img_a = resize_image_by_ratio(image_img_a, RESIZE_RATIO)
    image_img_b = resize_image_by_ratio(image_img_b, RESIZE_RATIO)
    image_img_c = resize_image_by_ratio(image_img_c, RESIZE_RATIO)

    image_hc_a = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC
    image_hc_b = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC
    image_hc_c = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC

    H_ab = get_transformation_matrix(image_hc_a, image_hc_b)
    H_bc = get_transformation_matrix(image_hc_b, image_hc_c)
    H_ac = H_ab * H_bc

    trans_img = transform_image_into_image(image_img_a, H_ac)
    cv2.imwrite('images/task2.jpg', trans_img)

def task3():
```

```
    '''
        Repeat task 1 and 2 with my own images.
    '''
    image_img_d = cv2.imread('images/4.jpg')
    image_img_e = cv2.imread('images/5.jpg')
    image_img_f = cv2.imread('images/fangda.jpg')

    image_img_d = resize_image_by_ratio(image_img_d, RESIZE_RATIO)
    image_img_e = resize_image_by_ratio(image_img_e, RESIZE_RATIO)
    image_img_f = resize_image_by_ratio(image_img_f, RESIZE_RATIO)

    image_hc_d = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC
    image_hc_e = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC
    image_hc_f = [( int(x*RESIZE_RATIO), int(y*RESIZE_RATIO) ) for (x,y) in IMAGE_HC

    # Repeat task 1
    H_fd = get_transformation_matrix(image_hc_f, image_hc_d)
    H_fe = get_transformation_matrix(image_hc_f, image_hc_e)

    proj_img = project_world_into_image(image_img_f, image_img_d, H_fd)
    cv2.imwrite('images/task3_a.jpg', proj_img)
    proj_img = project_world_into_image(image_img_f, image_img_e, H_fe)
    cv2.imwrite('images/task3_b.jpg', proj_img)

    # Repeat task 2
    H_de = get_transformation_matrix(image_hc_d, image_hc_e)
    H_ef = get_transformation_matrix(image_hc_e, image_hc_f)
    H_df = H_de * H_ef

    trans_img = transform_image_into_image(image_img_d, H_df)
    cv2.imwrite('images/task3_c.jpg', trans_img)

def main():
    task1()
    task2()
    task3()

if __name__ == '__main__':
    main()
```