hw1

October 27, 2017

1 CSE 252A Computer Vision I Fall 2017

1.1 Assigment 1

This assignment contains theoretical and programming exercises. If you plan to submit hand written answers for theoretical exercises, please be sure your writing is readable and merge those in order with the final pdf you create out of this notebook.

1.2 Problem 1: Perspective Projection [5 pts]

Consider a perspective projection where a point

$$P = [\text{x y z}]^T$$

is projected onto an image plane Π' represented by $k = f' > 0$ as shown in the following figure. The first second and third coordinate axes are denoted by i,j,k respectively.

Consider the projection of a ray in the world coordinate system

$$Q = [2\ \text{-}3\ 0] + t[0\ 2\ 1]$$

where $-\infty \leq t \leq -1$.

Calculate the coordinates of the endpoints of the projection of the ray onto the image plane.

1.3 Problem 2: Thin Lens Equation [5 pts]

An illuminated arrow forms a real inverted image of itself at a distance of w = 90cm, measured along the optical axis of a convex thin lens as shown above. The image is half the size of the object

1. How far from the object must the lens be placed?
2. What is the focal length of the lens?
3. Suppose the arrow moves x cm to the right while the lens and image plane remain fixed. This will result in an out of focus image; what is the radius of the corresponding blur circle formed from the tip of the arrow on the image plane assuming the diameter of the lens is d?

1.4 Problem 3: Affine Projection [5 pts]

Consider an affine camera and a line in 3D space. Consider three points (A, B, and C) on that line, and the image of those three points (a, b and c). Now consider the distance between a and b and the distance between a and c. Show that the ratio of the distance is independent of the direction of the line.

1.5 Problem 4: Image Formation and Rigid Body Transformations [10 points]

In this problem we will practice rigid body transformations and image formations through the projective and affine camera model. The goal will be to photograph the following four points

$$ {}^{A}P_1 = [\text{-1 -0.5 2}]^T $$

,

$$ {}^{A}P_2 = [\text{1 -0.5 2}]^T $$

,

$$ {}^{A}P_3 = [\text{1 0.5 2}]^T $$

,

$$ {}^{A}P_4 = [\text{-1 0.5 2}]^T $$

To do this we will need two matrices. Recall, first, the following formula for rigid body transformation

$$ {}^{B}P = {}^{B}_{B}R\, {}^{A}P + {}^{B}O_A $$

Where ${}^{B}P$ is the point coordinate in the target (B) coordinate system. ${}^{A}P$ is the point coordinate in the source (A) coordinate system. ${}^{B}_{A}R$ is the rotation matrix from A to B, and ${}^{B}O_A$ is the origin of the coordinate system A expressed in B coordinates.

The rotation and translation can be combined into a single 4X4 extrinsic parameter matrix, P_e, so that ${}^{B}P = P_e * {}^{A}P$.

Once transformed, the points can be photographed using the intrinsic camera matrix, P_i which is a 3X4.

Once these are found, the image of a point, ${}^{A}P$, can be calculated as $P_i * P_e * {}^{A}P$.

We will consider four different settings of focal length, viewing angles and camera positions below. For each of these calculate:

a). the extrinsic transformation matrix,

b). intrinsic camera matrix under the perspective camera assumption.

c). intrinsic camera matrix under the affine camera assumption. In particular, around what point do you do the taylor series expansion?

d).Calculate the image of the four vertices and plot using the supplied **plotsquare** function (see e.g. output in figure below). 1. [No rigid body transformation]. Focal length = 1. The optical axis of the camera is aligned with the z-axis. 2. [Translation]. ${}^{B}O_A = [\text{0 0 1}]^T$. The optical axis of the camera is aligned with the z-axis. 3. [Translation and Rotation]. Focal length = 1. ${}^{B}_{A}R$ encodes a 30 degrees around the z-axis and then 60 degrees around the y-axis. ${}^{B}O_A = [\text{0 0 1}]^T$. 4. [Translation and Rotation, long distance]. Focal length = 5. ${}^{B}_{A}R$ encodes a 30 degrees around the z-axis and then 60 degrees around the y-axis. ${}^{B}O_A = [\text{0 0 13}]^T$.

We will not use a full intrinsic camera matrix (e.g. that maps centimeters to pixels, and defines the coordinates of the center of the image), but only parameterize this with f, the focal length. In other words: the only parameter in the intrinsic camera matrix under the perspective assumption is f, and the only ones under the affine assumption are: f, x_0, y_0, z_0, where x_0, y_0, z_0 is the center of the taylor series expansion.

Include a image like above. Note that the axis are the same for each row, to facilitate comparison between the two camera models. Also include: 1. The actual points around which you did the taylor series expansion for the affine camera models. 2. How did you arrive at these points? 3. How do the projective and affine camera models differ? Why is this difference smaller for the last image compared to the second last?

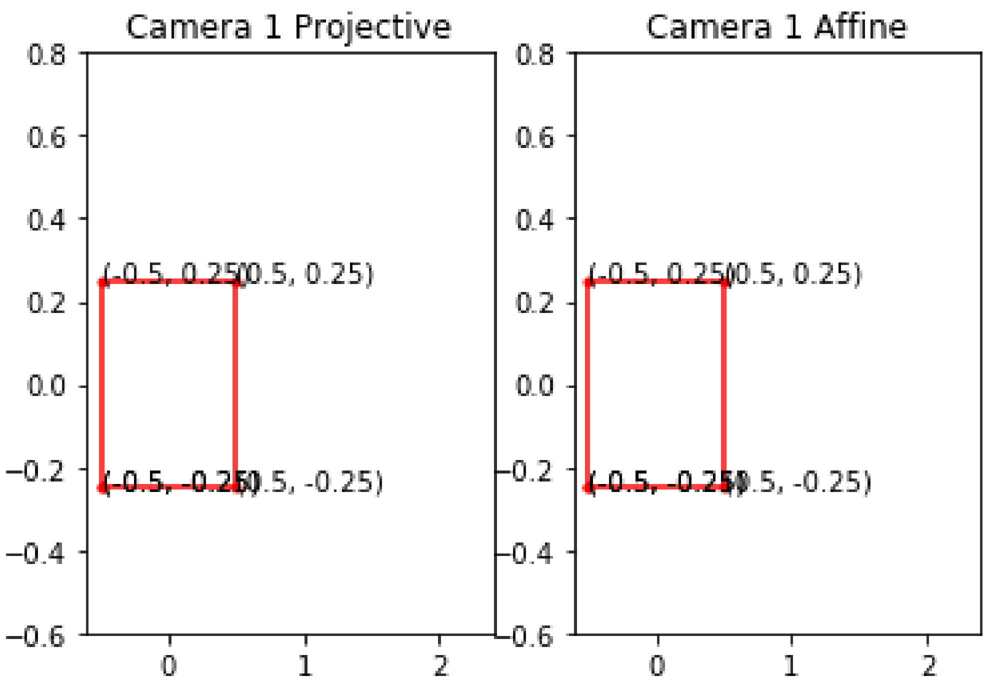
Note: this is a programming exercise.

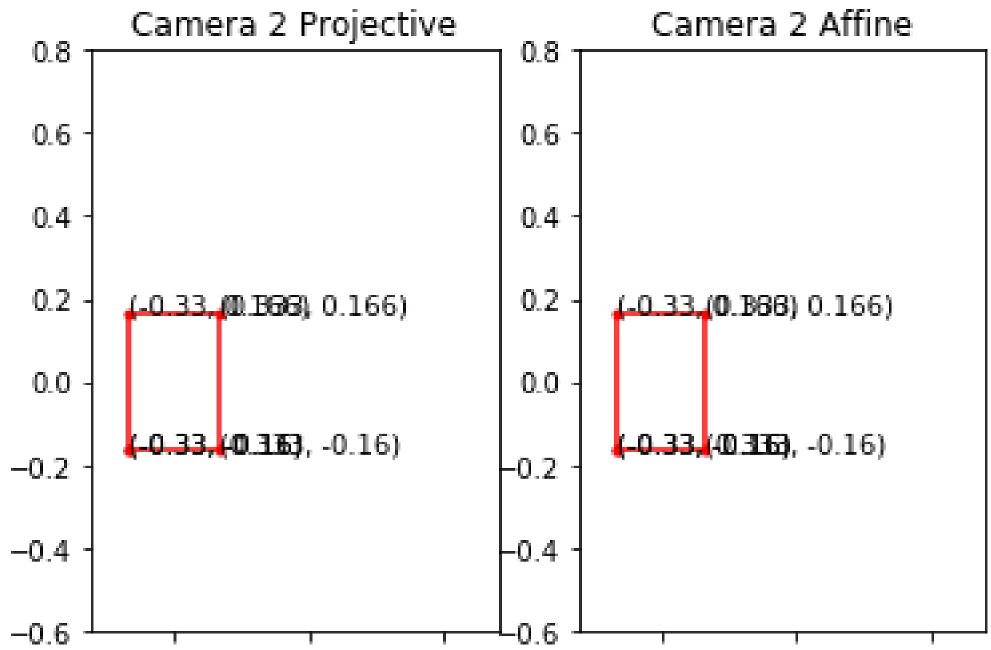
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt

# convert points from euclidian to homogeneous
def to_homog(points):

    points_homog = np.row_stack((points, [1]*points.shape[1]))

    return points_homog

# convert points from homogeneous to euclidian
def from_homog(points_homog):

    last_row = points_homog[-1]
    points = points_homog[0:points_homog.shape[0]-1,:]
    for i in range(points.shape[1]):
        points[:,i] = points[:,i]/last_row[i]

    return points

# project 3D euclidian points to 2D euclidian
def project_points(P_int, P_ext, pts):

    pts = to_homog(pts)
    temp = np.dot(np.dot(P_int, P_ext), pts)
    Project_pts = from_homog(temp)

    return Project_pts

def camera1():

    P_int_proj = np.eye(3,4)
    P_ext = np.eye(4,4)
    f = 1
    ref = np.array([[0, 0 ,2]]).T
    P_int_affine = np.array([[f/ref[2][0]                , 0                      ,\
                             -f*ref[0][0]/(ref[2][0]**2), f*ref[0][0]/ref[2][0]],\
                            [0                          , f/ref[2][0]            ,\
                             -f*ref[1][0]/(ref[2][0]**2), f*ref[1][0]/ref[2][0]],\
                            [0                          , 0                      ,\
                             0                          , 1                    ]])

    return P_int_proj, P_int_affine, P_ext

def camera2():

    P_int_proj = np.eye(3,4)
    P_ext = np.eye(4,4)
```

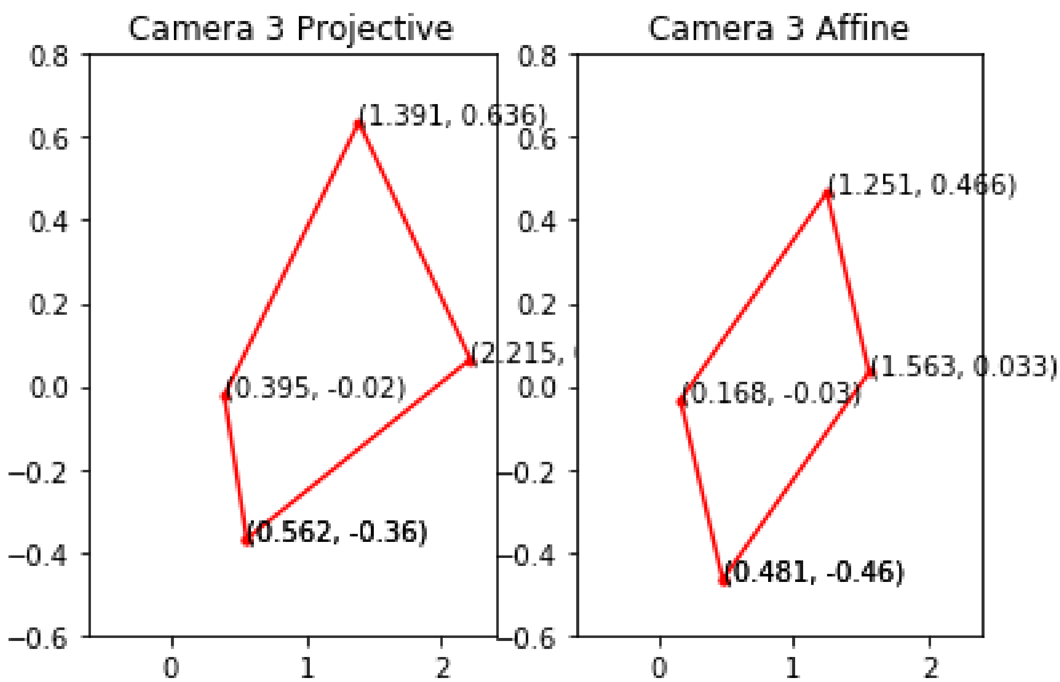
```
    P_ext[:,3] += [0, 0, 1, 0]
    f = 1
    ref = np.array([[0, 0 ,2]]).T
    ref = from_homog(np.dot(P_ext, to_homog(ref)))
    P_int_affine = np.array([[f/ref[2][0]              , 0                     ,\
                              -f*ref[0][0]/(ref[2][0]**2), f*ref[0][0]/ref[2][0]],\
                             [0                          , f/ref[2][0]           ,\
                              -f*ref[1][0]/(ref[2][0]**2), f*ref[1][0]/ref[2][0]],\
                             [0                          , 0                     ,\
                              0                          , 1                    ]])

    return P_int_proj, P_int_affine, P_ext

def camera3():

    P_int_proj = np.eye(3,4)
    P_int_affine = np.eye(3,4)
    rotz = np.array([[np.cos(np.pi/6), -np.sin(np.pi/6), 0, 0],\
                     [np.sin(np.pi/6), np.cos(np.pi/6) , 0, 0],\
                     [0              , 0               , 1, 0],\
                     [0              , 0               , 0, 1]])

    roty = np.array([[np.cos(np.pi/3) , 0, np.sin(np.pi/3), 0],\
                     [0               , 1, 0              , 0],\
                     [-np.sin(np.pi/3), 0, np.cos(np.pi/3), 0],\
                     [0               , 0, 0              , 1]])
    P_ext = np.dot(roty, rotz)
    P_ext[:,3] += [0, 0, 1, 0]

    f = 1
    ref = np.array([[0, 0 ,2]]).T
    ref = from_homog(np.dot(P_ext, to_homog(ref)))
    P_int_affine = np.array([[f/ref[2][0]              , 0                     ,\
                              -f*ref[0][0]/(ref[2][0]**2), f*ref[0][0]/ref[2][0]],\
                             [0                          , f/ref[2][0]           ,\
                              -f*ref[1][0]/(ref[2][0]**2), f*ref[1][0]/ref[2][0]],\
                             [0                          , 0                     ,\
                              0                          , 1                    ]])


    return P_int_proj, P_int_affine, P_ext

def camera4():

    P_int_proj = np.eye(3,4)
    P_int_affine = np.eye(3,4)
    rotz = np.array([[np.cos(np.pi/6), -np.sin(np.pi/6), 0, 0],\
                     [np.sin(np.pi/6), np.cos(np.pi/6) , 0, 0],\
```

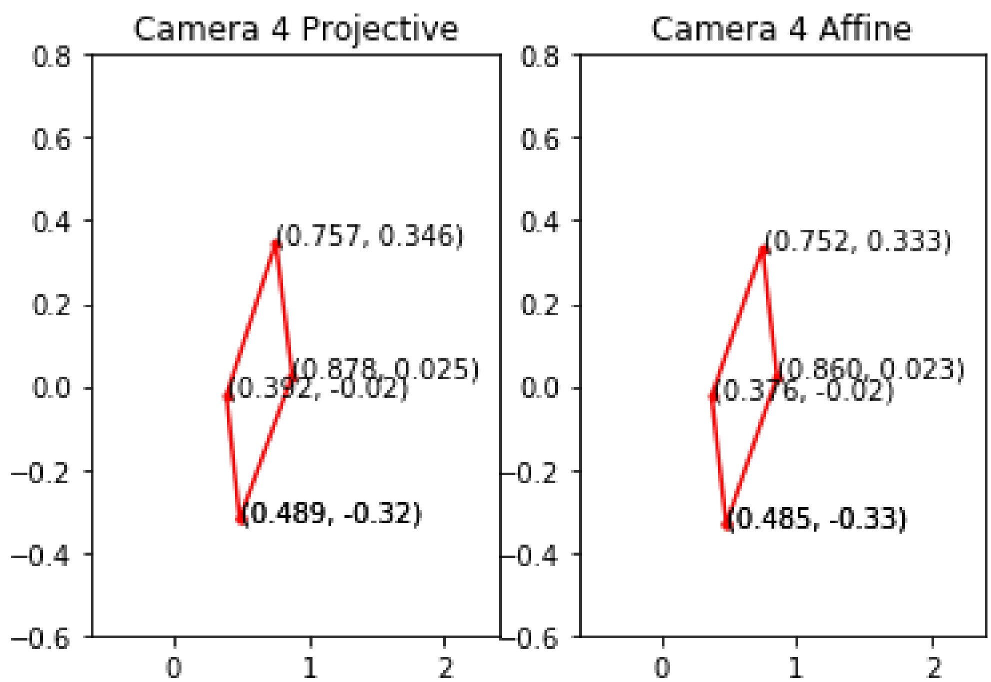
```
                    [0                , 0                , 1, 0],\
                    [0                , 0                , 0, 1]])

    roty = np.array([[np.cos(np.pi/3) , 0, np.sin(np.pi/3), 0],\
                    [0                 , 1, 0               , 0],\
                    [-np.sin(np.pi/3), 0, np.cos(np.pi/3), 0],\
                    [0                 , 0, 0               , 1]])
    P_ext = np.dot(roty, rotz)
    P_ext[:,3] += [0, 0, 13, 0]
    f = 5
    P_int_proj[2][2] = 1/f
    ref = np.array([[0, 0 ,2]]).T
    ref = from_homog(np.dot(P_ext, to_homog(ref)))
    P_int_affine = np.array([[f/ref[2][0]                , 0                        ,\
                              -f*ref[0][0]/(ref[2][0]**2), f*ref[0][0]/ref[2][0]],\
                             [0                          , f/ref[2][0]              ,\
                              -f*ref[1][0]/(ref[2][0]**2), f*ref[1][0]/ref[2][0]],\
                             [0                          , 0                        ,\
                              0                          , 1                        ]])


    return P_int_proj, P_int_affine, P_ext


#######################################################
# test code. Do not modify
#######################################################

def plot_points(points, title='', style='.-r', axis=[]):
    inds = list(range(points.shape[1]))+[0]
    plt.plot(points[0,inds], points[1,inds],style)
    for xy in zip(points[0,inds], points[1,inds]):
        plt.annotate('(%0.5s, %0.5s)' % xy, xy=xy, textcoords='data')
    if title:
        plt.title(title)
    if axis:
        plt.axis(axis)

def main():
    point1 = np.array([[-1,-.5,2]]).T
    point2 = np.array([[1,-.5,2]]).T
    point3 = np.array([[1,.5,2]]).T
    point4 = np.array([[-1,.5,2]]).T
    points = np.hstack((point1,point2,point3,point4))

    for i, camera in enumerate([camera1, camera2, camera3, camera4]):
        P_int_proj, P_int_affine, P_ext = camera()
        plt.subplot(1, 2, 1)
```

```
        plot_points(project_points(P_int_proj, P_ext, points), title='Camera %d Projec
        plt.subplot(1, 2, 2)
        plot_points(project_points(P_int_affine, P_ext, points), title='Camera %d Affi
        plt.show()

main()
```




Camera 2 Projective
Camera 2 Affine
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
1
2


Camera 3 Projective
(1.391, 0.636)
(0.395, -0.02)
(0.562, -0.36)
Camera 3 Affine
(1.251, 0.466)
(1.563, 0.033)
(0.168, -0.03)
(0.481, -0.46)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
0
1
2



1.6 Problem 5: Image warping and merging [10 pts]

You may use eig or svd routines in python for this part of the assignment.

We consider a vision application in which components of the scene are replaced by components from another image scene.

Consider for a moment that you are watching a sporting event on television whose audience, which has a broad, possibly multinational audience. During these events, it is advantageous and profitable to use different advertisements in different markets to target more directly viewers in those regions.

In this assignment you will implement a simple version of this algorithm using multiple advertisements on a single scene. Given two scenes, the natural thing to do would be to compute how your object of interest is observed in the current scene and warp it to match the destination scene. This would allow you to paste together the two images where your new advertisement overlaps the particular billboard in the new scene even though these two images were obtained in different locations.

This digital replacement is accomplished by a set of points for each advertisement in both the target (scene) and advertisement images. The task then consists of mapping the points from the advertisement to their respective points in the target image. In the most general case, there would be no constraints on the scene geometry, making the problem quite hard to solve. If, however, the scene can be approximated by a plane in 3D, a solution can be formulated much more easily even without the knowledge of camera calibration parameters.

To solve this section of the homework, you will begin by deriving the transformation that maps one image onto another in the planar scene case. Then you will write a program that implements this transformation and uses it to warp some UCSD logos into an art gallery.

To begin with, we consider the projection of planes in images. imagine two cameras C_1 and C_2 looking at a plane π in the world. Consider a point P on the plane π and its projection $p = [\text{u1, v1, 1}]^T$ in the image 1 and $q = [\text{u2, v2, 1}]^T$ in image 2.

Fact 1: These exists a unique, upto scale, 3X3 matrix H such that, for any point P:

$$q \approx Hp$$

Here $\approx$ denotes equality in homogeneous coordinates, meaning that the left and right hand sides are proportional. Note that H only depends on the plane and the projection matrices of the two cameras.

The interesting thing about this result is that by using H we can compute the image of P that would be seen in the camera with center C_2 from the image of the point in the camera with center at C_1, without knowing the three dimensional location. Such an H is a projective transformation of the plane, called a homography.

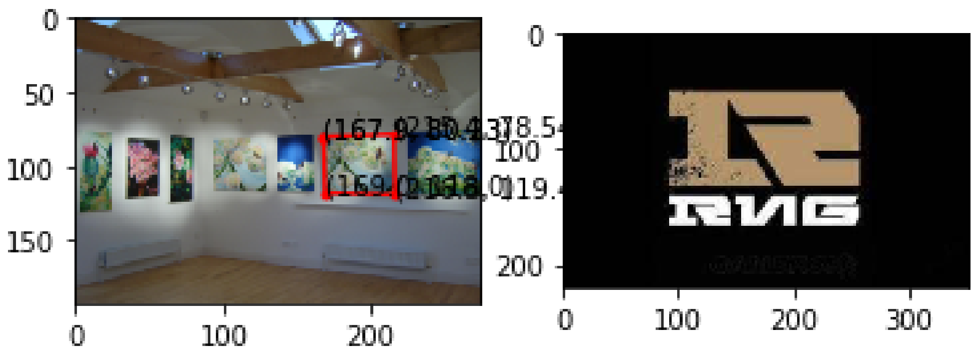
In this problem, complete the code for computeH and warp functions that can be used in the skeletal code that follows. 1. In the code snippets, the source image refers to the image(default: ucsd_logo.png) that needs to be replaced into the target image(joan_clancy_gallery.jpg). You may use any other source image for of your choice for this exercise. 2. The skeletal code has a composite function which is complete and will be used to merge the target_image and the warped source image so that the region in the target image is replaced by the source image. This function is fully implemented. 3. You will have to implement the computeH function that computes a homography from the target image to the source image. It takes in the point correspondences between the source image and target image in homogeneous coordinates respectively and returns a 3X3 homography matrix. 4. You will also have to implement the warp function that maps the source image into the target image plane. It takes in the source image, the points enclosing the region in the target to be replaced and the size of the target plane respectively. It returns the warped source image onto the target plane and a mask that indicates the regions to be filled in by the original target image. This mask is used in the composite function later.

Note: We have provided test code to check if your implementation for computeH is correct. All the code to plot the results needed is also provided along with the code to read in the images and other data required for this problem. Please try not to modify that code.

You may find following python built-ins helpful: numpy.linalg.svd, numpy.meshgrid

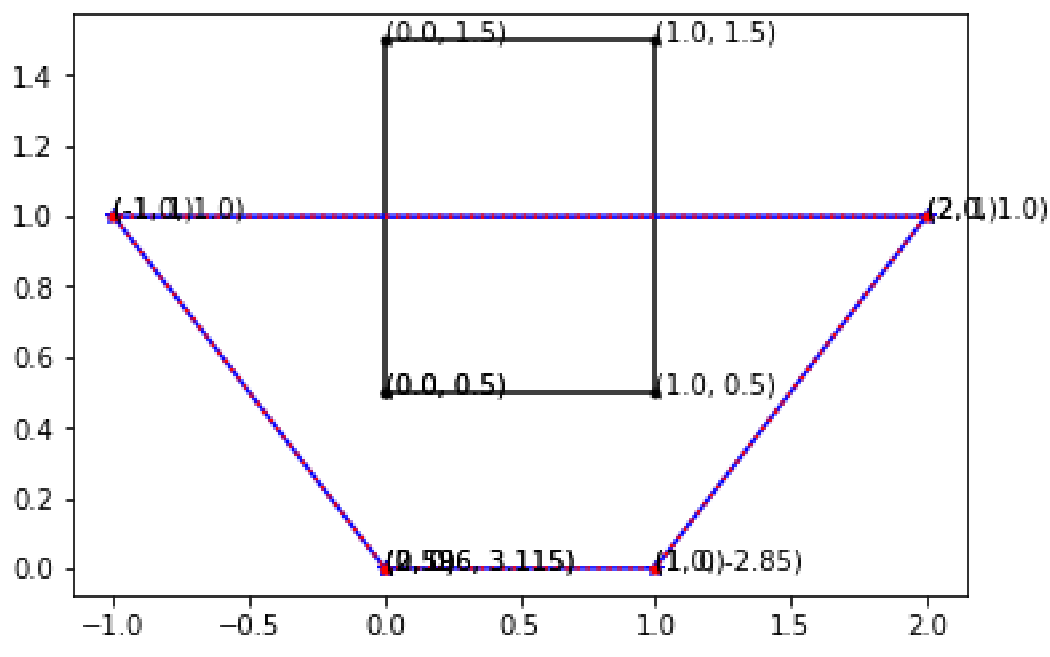
In [2]:
```
import numpy as np
from scipy.misc import imread, imresize
from scipy.io import loadmat
import matplotlib.pyplot as plt

# load images to be used
I1 = imresize(imread('joan_clancy_gallery.jpg'),.1)[:,:,:3]/255.

#You can use any image here. We would love to see your face in that gallery!!
I2 = imread('rng.jpg')[:,:,:3]/255.

# load target points
target_points = np.load('gallerypoints.npy')/10

# display images
plt.subplot(1, 2, 1)  # first plot
```

```
plt.imshow(I1)
#plt.plot(points[0,0],points[1,0],'.r')
plot_points(target_points)

plt.subplot(1, 2, 2)  # first plot
plt.imshow(I2)
plt.show()
```




```
In [3]: def computeH(source_points, target_points):
            # returns the 3x3 homography matrix such that:
            # np.matmul(H, source_points) ~ target_points
            # where source_points and target_points are expected to be in homogeneous

            # Please refer the note on DLT algorithm given at:
            # https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimation/homography_es

            # make sure points are 3D homogeneous
            assert source_points.shape[0]==3 and target_points.shape[0]==3

            row = 0
            A = np.zeros((8, 9))
            for i in range(source_points.shape[1]):
                x1 = source_points[0][i]
                y1 = source_points[1][i]

                x2_p = target_points[0][i] / target_points[2][i]
                y2_p = target_points[1][i] / target_points[2][i]

                A[row] = [-x1, -y1, -1, 0, 0, 0, x2_p*x1, x2_p*y1, x2_p]
                A[row + 1] = [0, 0, 0, -x1, -y1, -1, y2_p*x1, y2_p*y1, y2_p]

                row += 2

            U, sigma, V = np.linalg.svd(A, full_matrices=True)
```

```python
    #print(sigma)
    H = V[-1].reshape(3, 3)

    return H

#########################################################
# test code. Do not modify
#########################################################
def test_computeH():
    source_points = np.array([[0,0.5],[1,0.5],[1,1.5],[0,1.5]]).T
    target_points = np.array([[0,0],[1,0],[2,1],[-1,1]]).T
    H = computeH(to_homog(source_points), to_homog(target_points))
    mapped_points = from_homog(np.matmul(H,to_homog(source_points)))

    plot_points(source_points,style='.-k')
    plot_points(target_points,style='*-b')
    plot_points(mapped_points,style='.:r')
    plt.show()
    print('The red and blue quadrilaterals should overlap if ComputeH is implemented co
test_computeH()
```




```
The red and blue quadrilaterals should overlap if ComputeH is implemented correctly.
```

```python
In [4]: def warp(source_img, target_points, target_size):
            # make sure the new image (of size target_size) has the same number of color chann
```

```
    assert target_size[2]==source_img.shape[2]

    source_img = imresize(source_img, target_size)
    source_points = np.array([[0,0],[source_img.shape[1],0],[source_img.shape[1],source
    H = computeH(to_homog(source_points), to_homog(target_points))

    mask = np.ones(source_img.shape, dtype = 'float64')
    warp_img = np.zeros(source_img.shape, dtype = 'float64')

    for i in range(source_img.shape[0]):
        for j in range(source_img.shape[1]):
            temp = np.array([j,i])
            temp = temp.reshape(2,1)
            mapped_points = from_homog(np.matmul(H,to_homog(temp)))
            x = int(mapped_points[0])
            y = int(mapped_points[1])
            warp_img[y][x] = source_img[i][j]
            mask[y][x] = 0

    warp_img = warp_img[:,:,:3]/255

    return warp_img, mask

def composite(img1, img2, mask):
    # returns a new image that contains pixels from img1 where mask is true and img2 wh
    return img1*mask + img2*(1-mask)



#######################################################
# test code. Do not modify
#######################################################
I2_warp, mask = warp(I2, target_points, I1.shape)
I3 = composite(I1, I2_warp, mask)
plt.subplot(1, 2, 1)  # first plot
plt.imshow(I2_warp)
plt.subplot(1, 2, 2)  # first plot
plt.imshow(I3)
plt.show()
```


0
50
100
150
0
100
200
0
50
100
150
0
100
200

Homework 1

Wenjun Zhang

A53218995

Problem 1

The ray in the world coordinate system is

$$Q = [2 \quad -3 \quad 0] + t[0 \quad 2 \quad 1]$$

which we can also express as

$$Q = [x \quad y \quad z] = [2 \quad 2t-3 \quad t]$$

where $-\infty \leq t \leq -1$.

We have, by similar triangles, that $[x \quad y \quad z] \rightarrow [f'x/z \quad f'y/z \quad f']$

As for this ray, $Q' = [\frac{2f'}{t} \quad f'(2-\frac{3}{t}) \quad f']$.

The coordinates of the endpoints of the projection of the ray onto the image plane is:

When $t = -\infty$, $Q' = f'[0 \quad 2 \quad 1]$

When $t = -1$, $Q' = f'[-2 \quad 5 \quad 1]$

Problem 2

---

1. Assume the distance between the object and lens is u cm, and the distance between the image and lens is v cm, we have

$$\begin{cases} \frac{v}{u} = \frac{1}{2} \\ u + v = 90 \end{cases} \quad \text{leads to} \quad \begin{cases} u = 60 \\ v = 30 \end{cases}$$

So the object is placed 60 cm away from the lens

2. According to the equation below:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}$$

The focal length of the lens $f = 20$ cm.

3. According to the previous equation, we have

$$\frac{1}{60+x} + \frac{1}{v\prime} = \frac{1}{20} \quad \text{leads to} \quad v' = 20 + \frac{400}{40+x}.$$

While the lens and image plane remain unchanged, let's assume the radius of the blur circle is r, then it satisfies

$$\frac{2r}{d} = \frac{30 - (20 + \frac{400}{40 + x})}{20 + \frac{400}{40 + x}}$$

So the radius is $r = \frac{xd}{240+4x}$.

Problem 3

We can assume the three points on the line are A, B and C, and they can be expressed as:

$$\boldsymbol{A} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \lambda_A \cdot \boldsymbol{u} \qquad \boldsymbol{B} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \lambda_B \cdot \boldsymbol{u} \qquad \boldsymbol{C} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \lambda_C \cdot \boldsymbol{u}$$

where $\boldsymbol{u}$ is an unit vector in that line.

From the equation of Affine model, we have

$$\begin{bmatrix} u \\ v \end{bmatrix} \approx \frac{f}{z_0} \begin{bmatrix} x_0 \\ y_0 \end{bmatrix} + \begin{bmatrix} \frac{f}{z_0} & 0 & -\frac{f x_0}{z_0^2} \\ 0 & \frac{f}{z_0} & -\frac{f y}{z_0^2} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \boldsymbol{Mp} + \boldsymbol{k}$$

Assume the three points on the image based on Affine model are a, b and c. The distance between a and b and the distance between a and c are:

$$\begin{cases} |\boldsymbol{ab}| = |\boldsymbol{MA} + \boldsymbol{k} - \boldsymbol{MB} - \boldsymbol{k}| = |(\lambda_A - \lambda_B)\boldsymbol{Mu}| \\ |\boldsymbol{ac}| = |\boldsymbol{MA} + \boldsymbol{k} - \boldsymbol{MC} - \boldsymbol{k}| = |(\lambda_A - \lambda_C)\boldsymbol{Mu}| \end{cases}$$

Thus, the ratio of the distance is

$$\frac{|\boldsymbol{ab}|}{|\boldsymbol{ac}|} = \left|\frac{\lambda_A - \lambda_B}{\lambda_A - \lambda_c}\right| \left|\frac{\boldsymbol{Mu}}{\boldsymbol{Mu}}\right| = \left|\frac{\lambda_A - \lambda_B}{\lambda_A - \lambda_c}\right|$$

which is independent of the direction of the line.

Problem 4

The actual points around which we do the Taylor series expansion for the four models are $[0 \quad 0 \quad 2]$, $[0 \quad 0 \quad 3]$, $[1.73205081 \quad 0 \quad 2]$, $[1.73205081 \quad 0 \quad 14]$.

We derive the original reference point $P_0 = [0 \quad 0 \quad 2]$ since they are the center of the 4 source points. Then for the each of the four models, we first compute the 4×4 extrinsic parameter matrix P_e. So the reference point now becomes $P_0' = P_e * P_0$.

The difference between the affine model and projective model is that, considering their expressions:

$$P_{proj} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{pmatrix} \qquad P_{affine} = \begin{pmatrix} f/z_0 & 0 & -fx_0/z_0^2 & fx_0/z_0 \\ 0 & f/z_0 & -fy_0/z_0^2 & fy_0/z_0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The affine camera model takes perspective projection equation, and perform Taylor series expansion about some point (x_0, y_0, z_0) and drop terms with higher order than linear.

The reason why the difference of the last image is smaller than the second last image is that, the focal length is longer, but the last image takes a reference point whose z_0 is quite large, which means the distance is pretty far. And the rotation and movement of an object in a long distance will make less influence in vision. And that leads to the trivial difference in the last image.