CSE 252A Computer Vision I Fall 2018 - Assignment 1

Instructor: David Kriegman

Assignment Published On: Tuesday, October 9, 2018

Due On: Tuesday, October 23, 2018 11:59 pm

Instructions

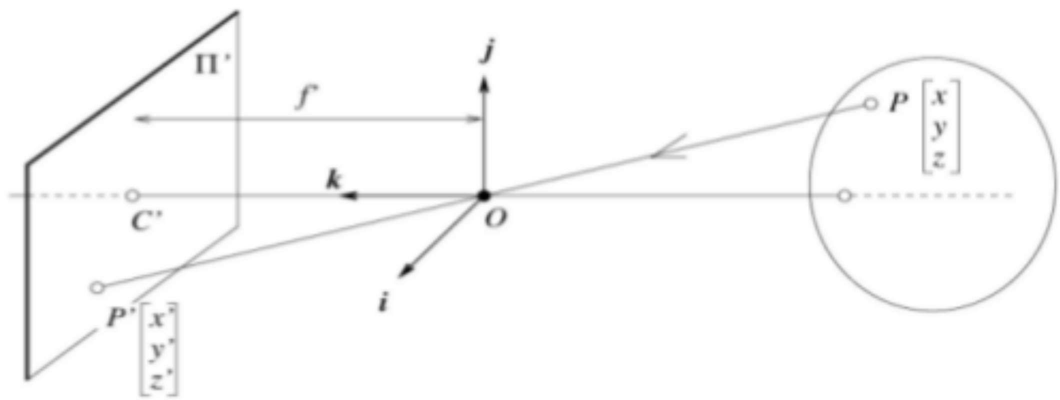
- Review the academic integrity and collaboration policies on the course website.
- This assignment must be completed individually.
- This assignment contains theoretical and programming exercises. If you plan to submit hand written answers for theoretical exercises, please be sure your writing is readable and merge those in order with the final pdf you create out of this notebook. You could fill the answers within the notebook iteself by creating a markdown cell.
- Programming aspects of this assignment must be completed using Python in this notebook.
- If you want to modify the skeleton code, you can do so. This has been provided just to provide you with a framework for the solution.
- You may use python packages for basic linear algebra (you can use numpy or scipy for basic operations), but you may not use packages that directly solve the problem.
- If you are unsure about using a specific package or function, then ask the instructor and teaching assistants for clarification.
- You must submit this notebook exported as a pdf. You must also submit this notebook as .ipynb file.
- You must submit both files (.pdf and .ipynb) on Gradescope. You must mark each problem on Gradescope in the pdf.
- **Late policy** - 10% per day late penalty after due date up to 3 days.

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 two rays in the world coordinate system

$$Q1 = [7 \text{ -3 } 1] + t[8 \ 2 \ 4]$$
$$Q2 = [2 \text{ -5 } 9] + t[8 \ 2 \ 4]$$

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

Calculate the coordinates of the endpoints of the projection of the rays onto the image plane. Identify the vanishing point based on the coordinates.

P1_Solution

(1)

The points in line Q1,Q2can be denoted as

$$Q1 = \begin{bmatrix} 7+8t \\ -3+2t \\ 4t+1 \end{bmatrix}, Q2 = \begin{bmatrix} 2+8t \\ -5+2t \\ 9+4t \end{bmatrix}$$

We can convert the Euclidean cordinates into homogenous cordinate by inserting 1 element,

$$Q1 = \begin{bmatrix} 7+8t \\ -3+2t \\ 4t+1 \\ 1 \end{bmatrix}, Q2 = \begin{bmatrix} 2+8t \\ -5+2t \\ 9+4t \\ 1 \end{bmatrix}$$

Therefore, the projectie transformation for points in line Q1 and Q2 are,

$$\begin{bmatrix} u_1' \\ v_1' \\ w_1' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f'} & 0 \end{bmatrix} \times \begin{bmatrix} 7+8t \\ -3+2t \\ 4t+1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7+8t \\ -3+2t \\ \frac{4t+1}{f'} \end{bmatrix}$$

$$\begin{bmatrix} u_2' \\ v_2' \\ w_2' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f'} & 0 \end{bmatrix} \times \begin{bmatrix} 2+8t \\ -5+2t \\ 9+4t \\ 1 \end{bmatrix} = \begin{bmatrix} 2+8t \\ -5+2t \\ \frac{9+4t}{f'} \end{bmatrix}$$

Then the projection points on image π' are,

$$\begin{bmatrix} \frac{(7+8t)f'}{4t+1} \\ \frac{(-3+2t)f'}{4t+1} \end{bmatrix} \text{and} \begin{bmatrix} \frac{(2+8t)f'}{9+4t} \\ \frac{(-5+2t)f'}{9+4t} \end{bmatrix}$$

The endpoints means $t = -1$. Therefore,we can calculate endpoints on image plane are

$$\begin{bmatrix} \frac{f'}{3} \\ \frac{5f'}{3} \end{bmatrix} \text{and} \begin{bmatrix} \frac{-6f'}{5} \\ \frac{-7f'}{5} \end{bmatrix}$$

(2)

First of all, we can choose 2 points on line Q1 and 2 points on line Q2.
Line Q1:
When $t = -1, A_1 = [-1, -5, -3, 0]^T$
When $t = -\frac{1}{2}, A_2 = [3, -4, -1, 0]^T$
Link Q2:
When $t = -1, A_1 = [-6, -7, 5, 0]^T$
When $t = -\frac{1}{2}, A_2 = [-2, -6, 7, 0]^T$
Then, after transforming the 4 points onto the plane Π', they are dipicted as follow

$$A_1' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f'} & 0 \end{bmatrix} \times A_1 = \begin{bmatrix} -1 \\ -5 \\ -\frac{3}{f} \end{bmatrix} \rightarrow \begin{bmatrix} \frac{f}{3} \\ \frac{5f}{3} \end{bmatrix} \text{(Euclidean Cordinates)}$$

$$A_2' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f'} & 0 \end{bmatrix} \times A_2 = \begin{bmatrix} 3 \\ -4 \\ -\frac{1}{f} \end{bmatrix} \rightarrow \begin{bmatrix} -3f \\ 4f \end{bmatrix} \text{(Euclidean Cordinates)}$$

$$B_1' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f'} & 0 \end{bmatrix} \times B_1 = \begin{bmatrix} -6 \\ -7 \\ -\frac{5}{f} \end{bmatrix} \rightarrow \begin{bmatrix} \frac{-6f}{5} \\ \frac{-7f}{5} \end{bmatrix} \text{(Euclidean Cordinates)}$$

$$B_2' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{f'} & 0 \end{bmatrix} \times B_2 = \begin{bmatrix} -2 \\ -6 \\ -\frac{7}{f} \end{bmatrix} \rightarrow \begin{bmatrix} \frac{-2f}{7} \\ \frac{-6f}{7} \end{bmatrix} \text{(Euclidean Cordinates)}$$

Further, Line Q1 can be denoted by

$$\frac{4f-\frac{5f}{3}}{-3f-\frac{f}{3}}(x+3f)-4f=y$$

Line Q2 can be denoted by

$$\frac{\frac{-6f}{7}+\frac{7f}{5}}{-\frac{2f}{7}+\frac{6f}{5}}(x+\frac{2f}{7})-\frac{6f}{7}=y$$

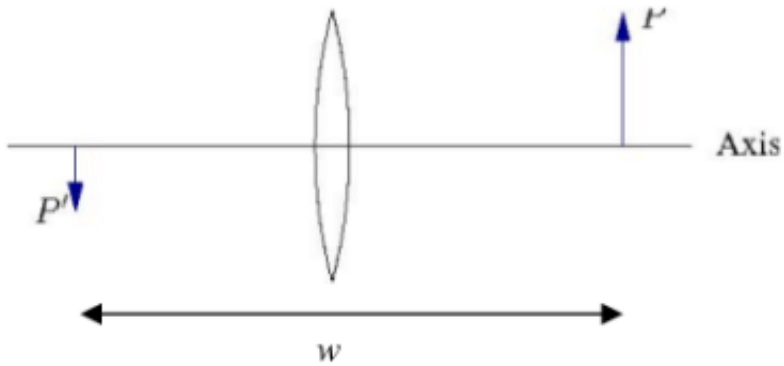
Therefore, we can joint the two equations to produce the intersection which is the vanishing point.

$$\begin{cases} \frac{-7}{10}(x+3f)-4f=y, \\ \frac{19}{32}(x+\frac{2f}{7})-\frac{6f}{7}=y, \end{cases}$$

Finally, the intersection is $(2f, \frac{1}{2}f)$, which is also the vaninsing point.

Problem 2: Thin Lens Equation [5 pts]

An illuminated arrow forms a real inverted image of itself at a distance of $w = 60$ cm, 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? Whats is the focal length of the lens?
2. At what distance from the center of the lens should the arrow be placed so that the height of the image is the same?
3. What would be the type and location of image formed if the arrow is placed at a distance of 5 cm along the optical axis from the optical center?

P2_Solution

(1)
According to thin lens equation,
$\frac{1}{z'} - \frac{1}{z} = \frac{1}{f}$, note that z is negative.
We can describe the equation in scalar which is
$\frac{1}{z'} + \frac{1}{z} = \frac{1}{f}$
Under similarity relation, we also know
$z' = \frac{1}{z}$
And $w = z' + z = 60cm$
Therefore,

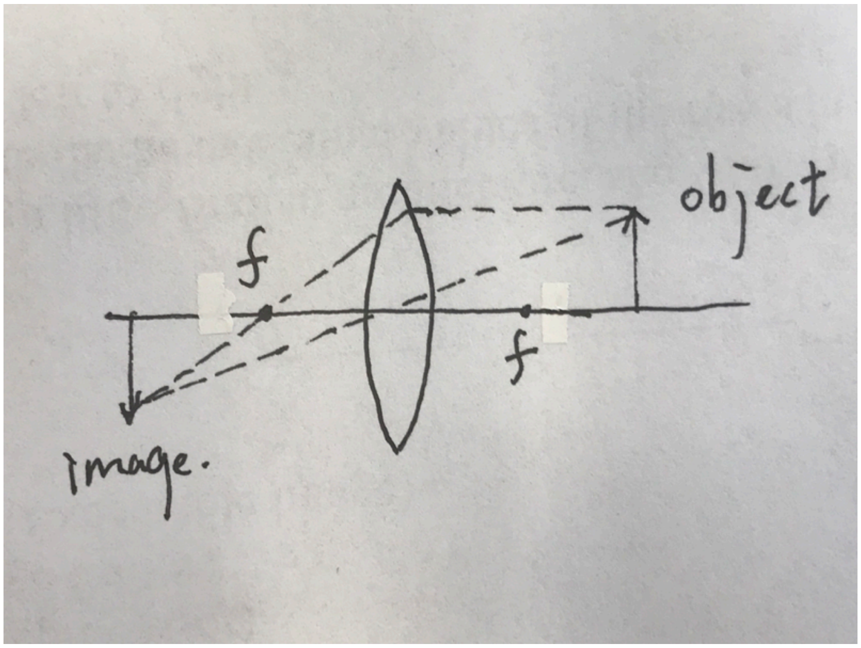
$$z' = 20cm$$
$$z = 40cm$$

Finally,
$\frac{1}{f} = \frac{1}{z'} + \frac{1}{z} => f = \frac{40}{3}cm$

So, the object should be placed 40cm far way from the center of lens and the focal length is $\frac{40}{3}$ cm.

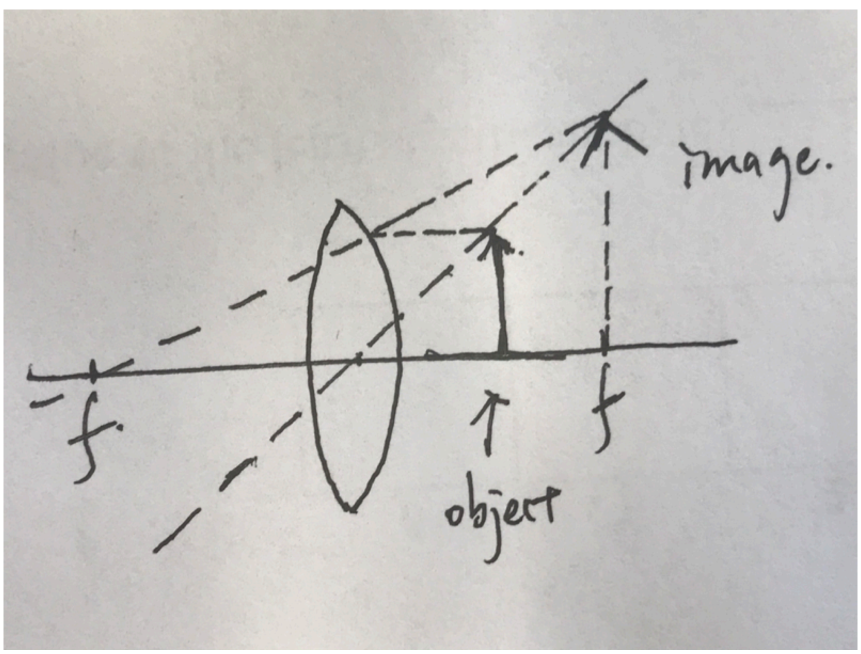
(2)

In order to have the same height image, the object should be placed $\frac{80}{3}$ cm far away from the lens.



(3)

Since the focal length is $\frac{40}{3}$ cm. If we put the object at 5cm< $\frac{40}{3}$ cm away from the lens, the two rays will not converge. However, it will form a virtual image under the reverse extension line as the following figure shows. Therefore,it will form a virtual image



Problem 3: Affine Projection [3 pts]

Show that the image of a pair of parallel lines in 3D space is a pair of parallel lines in an affine camera.

P3_Solution:

Suppose there are 4 points in 3D space named A,B,C,D. And $\vec{AB}$ is paralleled to $\vec{CD}$,where $\vec{A}=\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix}$, $\vec{B}=\begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}$, $\vec{C}=\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$, $\vec{D}=\begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$

We define the Affine transformation matrix to be H, where

$$H = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix}$$

Then the above 4 points can be projected onto the affine plane by the following process

$$A^{'} = H \times A = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix}$$

Also

$$B^{'} = H \times B = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}$$

$$C^{'} = H \times C = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

$$D^{'} = H \times D = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}$$

Then the two lines in affine camera can be denoted by

$$\overrightarrow{A^{'}B^{'}} = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times (\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} - \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix})$$

Also

$$\overrightarrow{C'D'} = \begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times (\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} - \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix})$$

Therefore, the ratio in affine plane is

$$\frac{\overrightarrow{A'B'}}{\overrightarrow{C'D'}} = \frac{\begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times (\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} - \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix})}{\begin{bmatrix} A & t \\ 0 & 1 \end{bmatrix} \times (\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} - \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix})} = \frac{\begin{bmatrix} x_a \\ y_a \\ z_a \end{bmatrix} - \begin{bmatrix} x_b \\ y_b \\ z_b \end{bmatrix}}{\begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} - \begin{bmatrix} x_d \\ y_d \\ z_d \end{bmatrix}} = \frac{\overrightarrow{AB}}{\overrightarrow{CD}}$$

Therefore, the image of a pair of parallel lines in 3D space is a pair of parallel lines in an affine camera.

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

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

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 4 × 4 extrinsic parameter matrix, P_e, so that $^{B}P = P_e \cdot \, ^{A}P$.

Once transformed, the points can be photographed using the intrinsic camera matrix, P_i which is a 3 × 4 matrix.

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

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

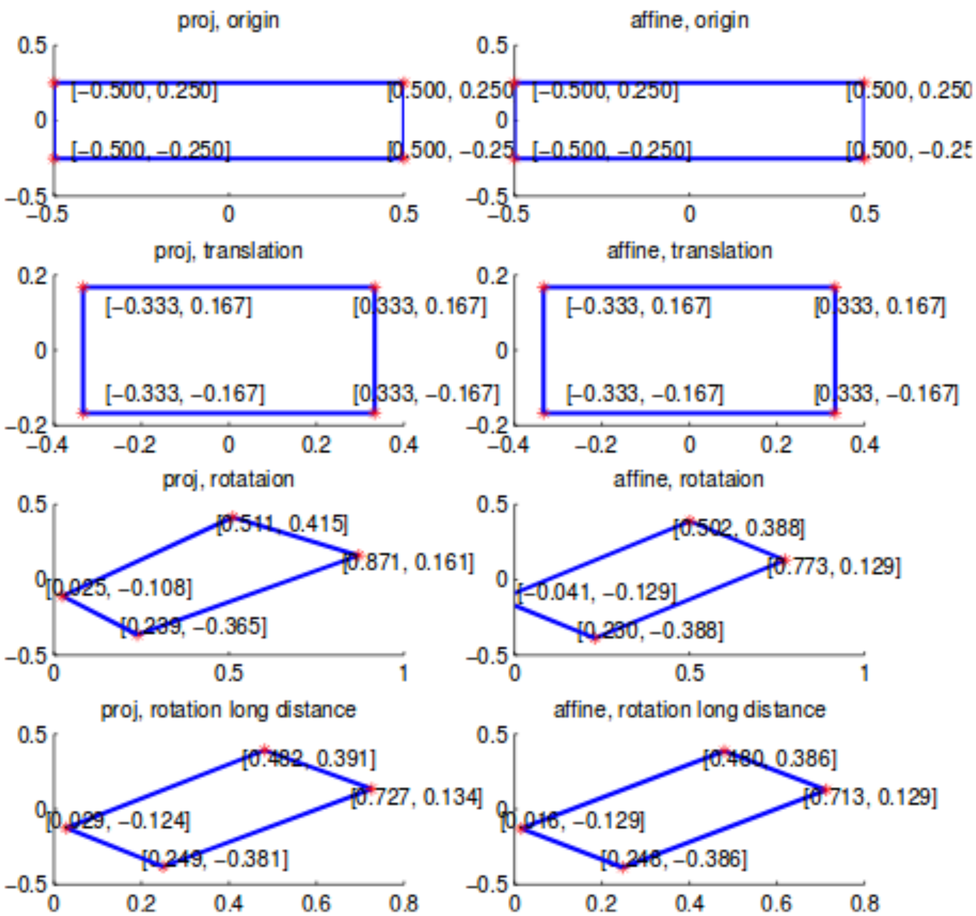
a) Extrinsic transformation matrix,

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 functions

Your output should look something like the following image (Your output values might not match, this is just an example)



1. [No rigid body transformation]. Focal length = 1. The optical axis of the camera is aligned with the z-axis.
2. [Translation]. ${}^{B}O_A = [0\ 0\ 1]^T$. Focal length = 1. 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. ${}^{B}O_A = [0\ 0\ 1]^T$.
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. ${}^{B}O_A = [0\ 0\ 13]^T$.

> You can refer the Richard Szeliski starting page 36 for image formation and the extrinsic matrix.
>
> Intrinsic matrix calculation for perspective and affine camera models was covered in class and can be referred in slide 3 http://cseweb.ucsd.edu/classes/fa18/cse252A-a/lec3.pdf (http://cseweb.ucsd.edu/classes/fa18/cse252A-a/lec3.pdf)

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.

Note that the axis are the same for each row, to facilitate comparison between the two camera models. Also include:

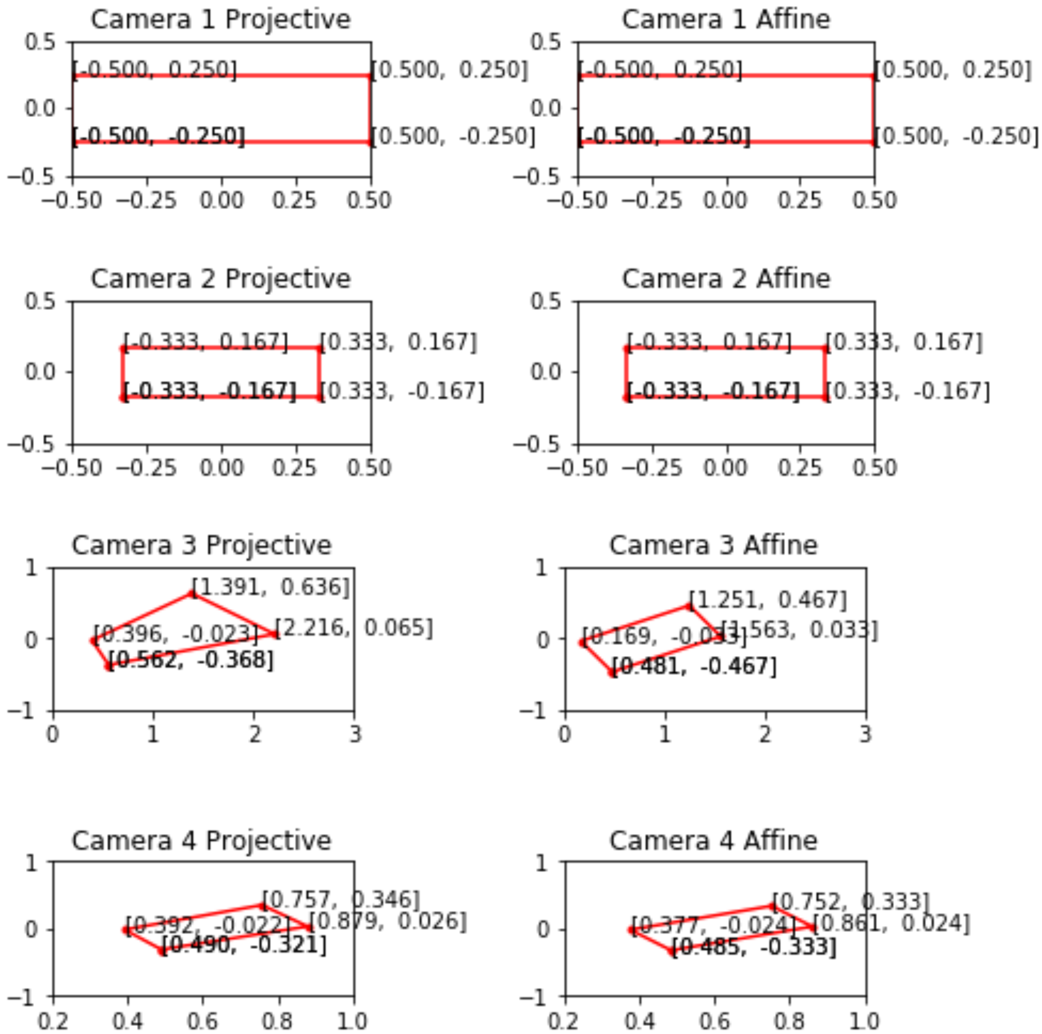
1. The actual points around which you did the taylor series expansion for the affine camera models.
2. How did you arrive at these points?
3. How do the projective and affine camera models differ? Why is this difference smaller for the last image compared to the second last?

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
import math


# convert points from euclidian to homogeneous
def to_homog(points):
    #case1: a point vector
    if points.ndim == 1:
        return np.append(points, 1)
    #case2: points array
    else:
        _, N = points.shape
        return np.vstack((points, np.ones((1,N))))
    return point_homog

# convert points from homogeneous to euclidian
def from_homog(points_homog):
    # write your code here
    points_inhomog = points_homog / points_homog[2]
    return points_inhomog[:2]

# project 3D euclidian points to 2D euclidian
def project_points(P_int, P_ext, pts):
    """
    Using the instruction above, the projection should be P_in*P_ext*p
```

```
ts
    However, we should treat the points in homogenous
    And return the result also in homogenous
    """
    pts_homo = to_homog(pts)
    return from_homog(np.dot(np.dot(P_int,P_ext),pts_homo))


# Change the three matrices for the four cases as described in the pro
blem
# in the four camera functions geiven below. Make sure that we can see
the formula
# (if one exists) being used to fill in the matrices. Feel free to doc
ument with
# comments any thing you feel the need to explain.

def camera1():
    # write your code here
    f = 1
    x0 = 0
    y0 = 0
    z0 = 2

    P_ext = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])
    P_int_proj = np.array([[1,0,0,0],[0,1,0,0],[0,0,1/f,0]])
    P_int_affine = np.array([[f/z0,0,-(f*x0)/(z0**2),(f*x0)/z0],[0,f/z
0,-(f*y0)/(z0**2),(f*y0)/z0],[0,0,0,1]])
    return P_int_proj, P_int_affine, P_ext

def camera2():
    #There's no rotation and only a transormation in z axis by 1
    #So the last element in the the third row is 1.
    f = 1
    x0 = 0
    y0 = 0
    z0 = 3

    P_ext = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,1],[0,0,0,1]])
    P_int_proj = np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0]])
    P_int_affine = np.array([[f/z0,0,-(f*x0)/(z0**2),(f*x0)/z0],[0,f/z
0,-(f*y0)/(z0**2),(f*y0)/z0],[0,0,0,1]])
    return P_int_proj, P_int_affine, P_ext

def camera3():
    f = 1
    x0 = 1.73
    y0 = 0
    z0 = 2

    P_ext=np.dot(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),1],[0,0,0,1]]),
                 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]]))
    P_int_proj= np.array([[1,0,0,0],[0,1,0,0],[0,0,f,0]])
    P_int_affine = np.array([[f/z0,0,-(f*x0)/(z0**2),(f*x0)/z0],[0,f/z
0,-(f*y0)/(z0**2),(f*y0)/z0],[0,0,0,1]])
    return P_int_proj, P_int_affine, P_ext

def camera4():
    # write your code here
    f = 5
    x0 = 1.73
    y0 = 0
    z0 = 14

    P_ext = np.dot(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),13],[0,0,0,1]]),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]]))
    P_int_proj = np.array([[1,0,0,0],[0,1,0,0],[0,0,1/f,0]])
    P_int_affine = np.array([[f/z0,0,-(f*x0)/(z0**2),(f*x0)/z0],[0,f/z
0,-(f*y0)/(z0**2),(f*y0)/z0],[0,0,0,1]])
    return P_int_proj, P_int_affine, P_ext


# Use the following code to display your outputs
# You are free to change the axis parameters to better
# display your quadrilateral but do not remove any annotations

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 i in range(len(points[0,inds])):
        plt.annotate("["+str("{0:.3f}".format(points[0,inds][i]))+",
"+str("{0:.3f}".format(points[1,inds][i])+"]"),(points[0,inds][i], poi
nts[1,inds][i]))

    if title:
        plt.title(title)
    if axis:
        plt.axis(axis)

    plt.tight_layout()

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))

    #camera1
    P_int_proj, P_int_affine, P_ext = camera1()
    plt.subplot(4,2,1)
    plot_points(project_points(P_int_proj, P_ext, points), title='Came
ra %d Projective'%(1), axis=[-0.5,0.5,-0.5,0.5])
    plt.subplot(4,2,2)
    plot_points(project_points(P_int_affine, P_ext, points), title='Ca
mera %d Affine'%(1), axis=[-0.5,0.5,-0.5,0.5])
    plt.subplots_adjust(wspace =0.7, hspace =0)     #adjust the margin
    plt.show()

    #camera2
    P_int_proj, P_int_affine, P_ext = camera2()
    plt.subplot(4,2,3)
    plot_points(project_points(P_int_proj, P_ext, points), title='Came
ra %d Projective'%(2), axis=[-0.5,0.5,-0.5,0.5])
    plt.subplot(4,2,4)
    plot_points(project_points(P_int_affine, P_ext, points), title='Ca
mera %d Affine'%(2), axis=[-0.5,0.5,-0.5,0.5])
    plt.subplots_adjust(wspace =0.7, hspace =0)     #adjust the margin
    plt.show()

    #camera3
    P_int_proj, P_int_affine, P_ext = camera3()
    plt.subplot(4,2,5)
    plot_points(project_points(P_int_proj, P_ext, points), title='Came
ra %d Projective'%(3), axis=[0,3,-1,1])
    plt.subplot(4,2,6)
    plot_points(project_points(P_int_affine, P_ext, points), title='Ca
mera %d Affine'%(3), axis=[0,3,-1,1])
    plt.subplots_adjust(wspace =0.7, hspace =0)     #adjust the margin
    plt.show()

    #camera4
    P_int_proj, P_int_affine, P_ext = camera4()
    print()
    plt.subplot(4,2,7)
    plot_points(project_points(P_int_proj, P_ext, points), title='Came
ra %d Projective'%(4), axis=[0.2,1,-1,1])
    plt.subplot(4,2,8)
    plot_points(project_points(P_int_affine, P_ext, points), title='Ca
mera %d Affine'%(4), axis=[0.2,1,-1,1])
    plt.subplots_adjust(wspace =0.7, hspace =0)     #adjust the margin
    plt.show()
main()
```


Camera 1 Projective
[-0.500, 0.250]
[0.500, 0.250]
[-0.500, -0.250]
[0.500, -0.250]
Camera 1 Affine
[-0.500, 0.250]
[0.500, 0.250]
[-0.500, -0.250]
[0.500, -0.250]
Camera 2 Projective
[-0.333, 0.167]
[0.333, 0.167]
[-0.333, -0.167]
[0.333, -0.167]
Camera 2 Affine
[-0.333, 0.167]
[0.333, 0.167]
[-0.333, -0.167]
[0.333, -0.167]
Camera 3 Projective
[1.391, 0.636]
[0.396, -0.023]
[2.216, 0.065]
[0.562, -0.368]
Camera 3 Affine
[1.251, 0.467]
[0.169, -0.033]
[1.563, 0.033]
[0.481, -0.467]
Camera 4 Projective
[0.757, 0.346]
[0.392, -0.022]
[0.879, 0.026]
[0.490, -0.321]
Camera 4 Affine
[0.752, 0.333]
[0.377, -0.024]
[0.861, 0.024]
[0.485, -0.333]

1. The actual points around which you did the taylor series expansion for the affine camera models.

The points are
$C_1 = (0, 0, 2.0000)^T, C_2 = (0, 0, 3.0000)^T, C_3 = (1.7321, 0, 2.0000)^T, and\ C_4 = (1.7321, 0, 14.0000)^T$

2. How did you arrive at these points?

Since Taylor expansion needs to expand at a point which is near every points arond it. Therefore, it should be the centroid of the region. In our situation, the centroid should to be the center of the 4 given points. However, we should pay attention that it should be the 4 points after multiplying with the extinsic matrix.

3. How do the projective and affine camera models differ? Why is this difference smaller for the last image compared to the second last? The affine camera model uses approximation while the projective camera model doesn't. The affine transformation uses the same focal length f for all points by expanding at the centroid. For projective model, different points have different focal lengths. When f is larger,the relative magnitude of error is smaller so we can overlook the error. Therefore, this difference is smaller for the last image compared to the second last?

Problem 5: Homography [12 pts]

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

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

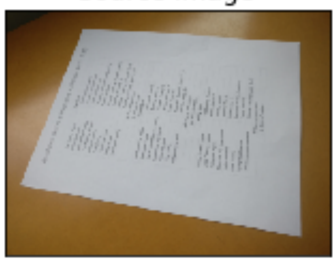
In this problem, we will implement partial functionality of a smartphone camera scanning application (Example: CamScanner) that, in case you've never used before, takes pictures of documents and transforms it by warping and aligning to give an image similar to one which would've been obtained through using a scanner.

The transformation can be visualized by imagining the use of two cameras forming an image of a scene with a document. The scene would be the document you're trying to scan placed on a table and one of the cameras would be your smart phone camera, forming the image that you'll be uploading and using in this assignment. There can also be an ideally placed camera, oriented in the world in such a way that the image it forms of the scene has the document perfectly algined. While it is unlikely you can hold your phone still enough to get such an image, we can use homography to transform the image you take into the image that the ideally placed camera would have taken.

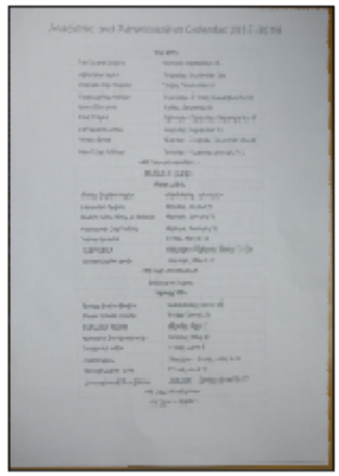
This digital replacement is accomplished by a set of corresponding points for the document in both the source (your picture) and target (the ideal) images. The task then consists of mapping the points from the source to their respective points in the target image. In the most general case, there would 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 understanding the transformation that maps one image onto another in the planar scene case. Then you will write a program that implements this transformation and use it to warp some document into a well aligned document (See the given example to understand what we mean by well aligned).

Source image



Resultant image



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 $p = [\mathrm{u1},\ \mathrm{v1},\ 1]^T$ in the image 1 and $q = [\mathrm{u2},\ \mathrm{v2},\ 1]^T$ in image 2.

There exists a unique, upto scale, 3 × 3 matrix H such that, for any point P:

$$q \approx Hp$$

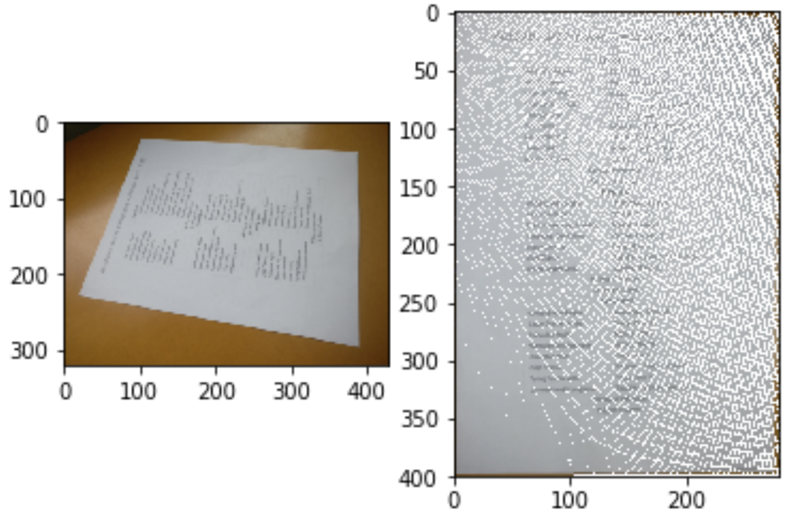
Here $\approx$ denotes equality in homogeneous coordinates, meaning that the left and right hand sides are proportional. Note that H only depends on the plane and the projection matrices of the two cameras.

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.

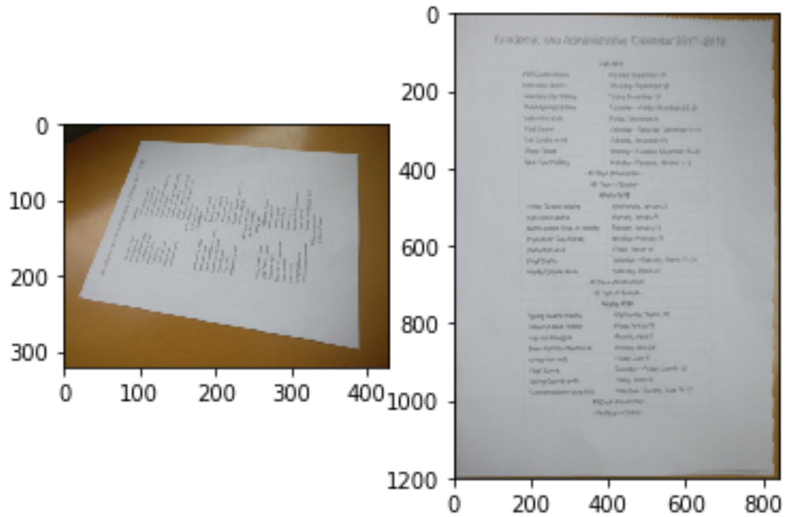
In this problem, complete the code for computeH and warp functions that can be used in the skeletal code that follows.

There are three warp functions to implement in this assignment, example ouputs of which are shown below. In warp1, you will create a homography from points in your image to the target image (Mapping source points to target points). In warp2, the inverse of this process will be done. In warp3, you will create a homography between a given image and your image, replacing your document with the given image.

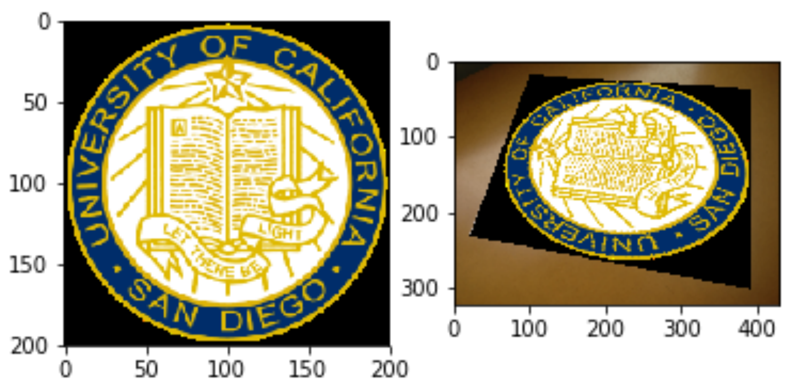
1.



2.



3.



2. In the context of this problem, the source image refers to the image of a document you take that needs to be replaced into the target.
3. The target image can start out as an empty matrix that you fill out using your code.
4. You will have to implement the computeH function that computes a homography. It takes in the point

correspondences between the source image and target image in homogeneous coordinates respectively and returns a 3×3 homography matrix.

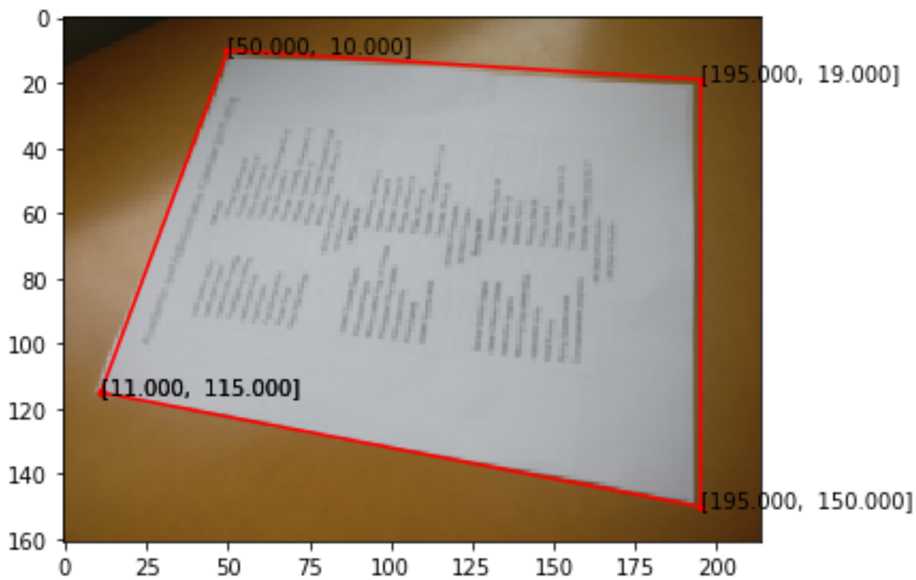
5. You will also have to implement the three warp functions in the skeleton code given and plot the resultant image pairs. For plotting, make sure that the target image is not smaller than the source image.

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 [17]:
```
import numpy as np
from scipy.misc import imread, imresize
from scipy.io import loadmat
import matplotlib.pyplot as plt

# load image to be used - resize to make sure it's not too large
# You can use the given image as well
# A large image will make testing you code take longer; once you're sa
tisfied with your result,
# you can, if you wish to, make the image larger (or till your compute
r memory allows you to)
source_image = imresize(imread("./photo.jpg"),.1)[:,:,:3]/255.
print(source_image.shape)
# display images
plt.imshow(source_image)

# Align the polygon such that the corners align with the document in y
our picture
# This polygon doesn't need to overlap with the edges perftectly, an a
pproximation is fine
# The order of points is clockwise, starting from bottom left.
x_coords = [11,50,195,195]
y_coords = [115,10,19,150]

# Plot points from the previous problem is used to draw over your imag
e
# Note that your coordinates will change once you resize your image ag
ain
source_points = np.vstack((x_coords, y_coords))
plot_points(source_points)

plt.show()
```

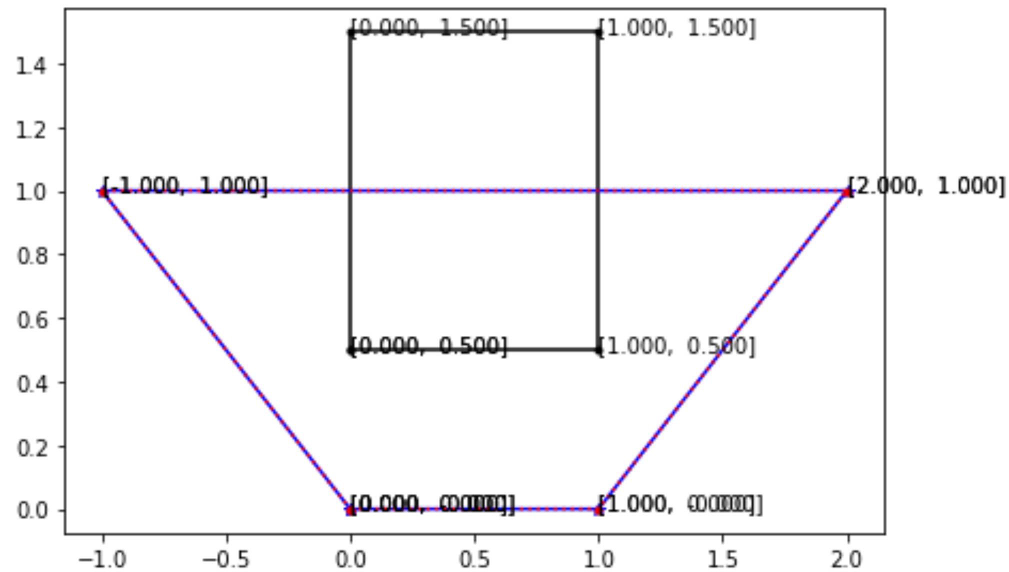
```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:10: Dep
recationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
  # Remove the CWD from sys.path while we load stuff.
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:10: Dep
recationWarning: `imresize` is deprecated!
`imresize` is deprecated in SciPy 1.0.0, and will be removed in 1.2.
0.
Use ``skimage.transform.resize`` instead.
  # Remove the CWD from sys.path while we load stuff.

(161, 214, 3)
```



In [16]:
```python
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 homo
geneous

    # Please refer the note on DLT algorithm given at:
    # https://cseweb.ucsd.edu/classes/wi07/cse252a/homography_estimati
on/homography_estimation.pdf
    # make sure points are 3D homogeneous
    assert source_points.shape[0]==3 and target_points.shape[0]==3
    N = source_points.shape[1]
    A = np.zeros((8,9))
    target_points_inhomog = from_homog(target_points)
    A[0,:] = np.array([(0-source_points[0,0]),(0-source_points[1,0]),-
1,0,0,0,target_points_inhomog[0,0]
                       *source_points[0,0],target_points_inhomog[0,0]*
source_points[1,0],target_points_inhomog[0,0]])
```

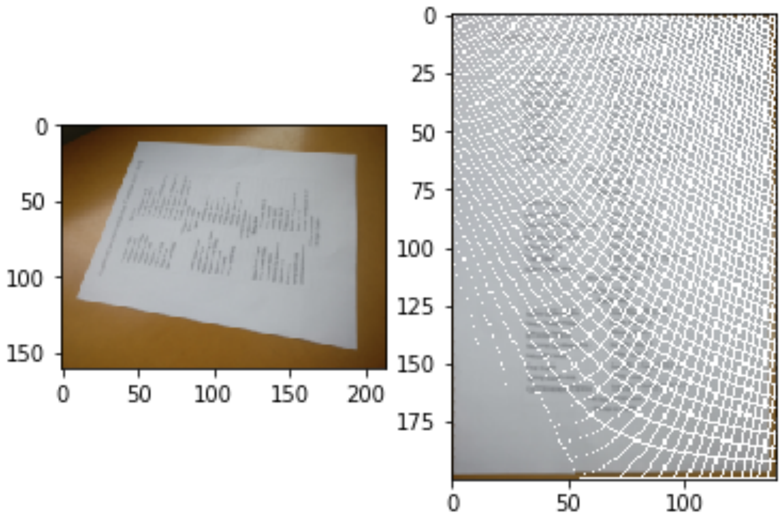
```
    A[1,:] = np.array([0,0,0,(0-source_points[0,0]),(0-source_points[1
,0]),-1,target_points_inhomog[1,0]
                           *source_points[0,0],target_points_inhomog[1,0]*
source_points[1,0],target_points_inhomog[1,0]])
    A[2,:] = np.array([(0-source_points[0,1]),(0-source_points[1,1]),-
1,0,0,0,target_points_inhomog[0,1]
                           *source_points[0,1],target_points_inhomog[0,1]*
source_points[1,1],target_points_inhomog[0,1]])
    A[3,:] = np.array([0,0,0,(0-source_points[0,1]),(0-source_points[1
,1]),-1,target_points_inhomog[1,1]
                           *source_points[0,1],target_points_inhomog[1,1]*
source_points[1,1],target_points_inhomog[1,1]])
    A[4,:] = np.array([(0-source_points[0,2]),(0-source_points[1,2]),-
1,0,0,0,target_points_inhomog[0,2]
                           *source_points[0,2],target_points_inhomog[0,2]*
source_points[1,2],target_points_inhomog[0,2]])
    A[5,:] = np.array([0,0,0,(0-source_points[0,2]),(0-source_points[1
,2]),-1,target_points_inhomog[1,2]
                           *source_points[0,2],target_points_inhomog[1,2]*
source_points[1,2],target_points_inhomog[1,2]])
    A[6,:] = np.array([(0-source_points[0,3]),(0-source_points[1,3]),-
1,0,0,0,target_points_inhomog[0,3]
                           *source_points[0,3],target_points_inhomog[0,3]*
source_points[1,3],target_points_inhomog[0,3]])
    A[7,:] = np.array([0,0,0,(0-source_points[0,3]),(0-source_points[1
,3]),-1,target_points_inhomog[1,3]
                           *source_points[0,3],target_points_inhomog[1,3]*
source_points[1,3],target_points_inhomog[1,3]])
    u,s,vh = np.linalg.svd(A)
    H_vec = vh[8,:]
    H = H_vec.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 correctly.')
test_computeH()
```




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

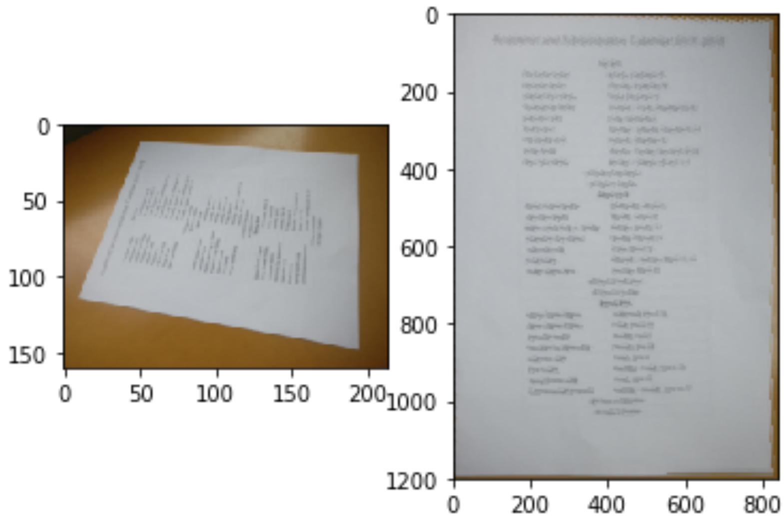
In [18]:
```
def warp(source_img, source_points, target_size):
    # Create a target image and select target points to create a homog
raphy from source image to target image,
    # in other words map all source points to target points and then c
reate
    # a warped version of the image based on the homography by filling
in the target image.
    # Make sure the new image (of size target_size) has the same numbe
r of color channels as source image
    assert target_size[2]==source_img.shape[2]
    target_points = np.array([[0,0],[target_size[1]-1,0],[target_size[
1]-1,target_size[0]-1],[0,target_size[0]-1]]).T
    H = computeH(to_homog(source_points), to_homog(target_points))
    mapped = np.ones(target_size)
    (height,width,c) = source_image.shape
    x_range = range(0,width)
    y_range = range(0,height)
    xv, yv = np.meshgrid(x_range, y_range)
    xv = xv.reshape(-1)
    yv = yv.reshape(-1)
    all_source = np.zeros((2,height*width))
    all_source[0,:] = xv
    all_source[1,:] = yv
    all_source = all_source.astype(np.int64)
    mapped_points = from_homog(np.matmul(H,to_homog(all_source)))
    mapped_points = mapped_points.astype(np.int64)
    for channel in range(c):
        for i in range(width*height):
            if(mapped_points[0,i]<0 or mapped_points[0,i]>=target_size
[1] or mapped_points[1,i]<0 or mapped_points[1,i]>=target_size[0]):
                continue
            else:
                mapped[mapped_points[1,i],mapped_points[0,i],channel]
= source_image[all_source[1,i],all_source[0,i],channel]

    return mapped


# Use the code below to plot your result
result = warp(source_image, source_points, (200,140,3))
plt.subplot(1, 2, 1)
plt.imshow(source_image)
plt.subplot(1, 2, 2)
plt.imshow(result)
plt.show()
```



The output of warp1 of your code probably has some striations or noise. The larger you make your target image, the less it will resemble the document in the source image. Why is this happening?

To fix this, implement warp2, by creating an inverse homography matrix and fill in the target image.

In [19]:
```
def warp2(source_img, source_points, target_size):
    # Create a target image and select target points to create a homog
raphy from target image to source image,
    # in other words map each target point to a source point, and then
create a warped version
    # of the image based on the homography by filling in the target im
age.
    # Make sure the new image (of size target_size) has the same numbe
r of color channels as source image


    target_points = np.array([[0,0],[target_size[1]-1,0],[target_size[
1]-1,target_size[0]-1],[0,target_size[0]-1]]).T
    H = computeH(to_homog(target_points), to_homog(source_points))
    mapped = np.ones(target_size)
    (height,width,c) = source_image.shape
    x_range = range(0,target_size[1])
    y_range = range(0,target_size[0])
    xv, yv = np.meshgrid(x_range, y_range)
    xv = xv.reshape(-1)
    yv = yv.reshape(-1)
    all_source = np.zeros((2,target_size[1]*target_size[0]))
    all_source[0,:] = xv
    all_source[1,:] = yv
    all_source = all_source.astype(np.int64)
    mapped_points = from_homog(np.matmul(H,to_homog(all_source)))
    mapped_points = mapped_points.astype(np.int64)
    for channel in range(c):
        for i in range(target_size[1]*target_size[0]):
            if(mapped_points[0,i]<0 or mapped_points[0,i]>=width
               or mapped_points[1,i]<0 or mapped_points[1,i]>=height):
                continue
            else:
                mapped[all_source[1,i],all_source[0,i],channel] = sour
ce_image[mapped_points[1,i],mapped_points[0,i],channel]

    return mapped


# Use the code below to plot your result
size_factor = 2
result = warp2(source_image, source_points, (600*size_factor,420*size_
factor,3))
plt.subplot(1, 2, 1)
plt.imshow(source_image)
plt.subplot(1, 2, 2)
plt.imshow(result)
plt.show()
```

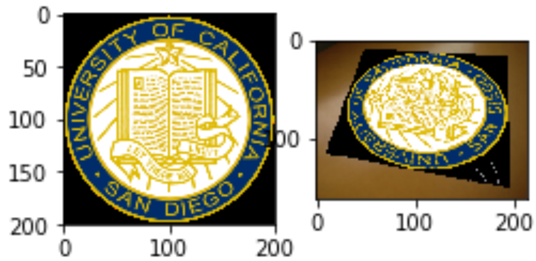


Try playing around with the size of your target image in warp1 versus in warp2, additionally you can also implement nearest pixel interpolation or bi-linear interpolations and see if that makes a difference in your output.

In warp3, you'll be replacing the document in your image with a provided image. Read in "ucsd_logo.png" as the source image, keeping your document as the target.

In [20]:
```
# Load the given UCSD logo image
source_image2 = imread('ucsd_logo.png')[:,:,:3]/255.

def warp3(target_image, target_points, source_image):
    source_points = np.array([[0,0],[source_image.shape[1]-1,0],[source_image.shape[1]-1,source_image.shape[0]-1],[0,source_image.shape[0]-1]]).T
    H = computeH(to_homog(source_points), to_homog(target_points))
    res = target_image
    (height,width,c) = source_image.shape
    x_range = range(0,width)
    y_range = range(0,height)
    xv, yv = np.meshgrid(x_range, y_range)
    xv = xv.reshape(-1)
    yv = yv.reshape(-1)
    all_source = np.zeros((2,width*height))
    all_source[0,:] = xv
    all_source[1,:] = yv
    all_source = all_source.astype(np.int64)
    mapped_points = from_homog(np.matmul(H,to_homog(all_source)))
    mapped_points = mapped_points.astype(np.int64)
    for channel in range(c):
        for i in range(width*height):
            if(mapped_points[0,i]<0 or mapped_points[0,i]>=target_image.shape[1] or mapped_points[1,i]<0 or mapped_points[1,i]>=target_image.shape[0]):
                continue
            else:
                res[mapped_points[1,i],mapped_points[0,i],channel] = source_image[all_source[1,i],all_source[0,i],channel]
    return res
# Use the code below to plot your result
result = warp3(source_image, source_points, source_image2)
plt.subplot(1, 3, 1)
plt.imshow(source_image2)
plt.subplot(1, 3, 2)
plt.imshow(result)
plt.show()
```

```
/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:2: Depr
ecationWarning: `imread` is deprecated!
`imread` is deprecated in SciPy 1.0.0, and will be removed in 1.2.0.
Use ``imageio.imread`` instead.
```



In [ ]: