Rept_raymo_BA_20667930

March 3, 2021

Task 5: Homography

Name: Raymond Ren
Degree: Bachelors
ID: 20667930

[2]:
```
# import libraries
import numpy as np
from itertools import combinations
import matplotlib.pyplot as plt
from scipy.linalg import null_space
from PIL import Image
import cv2
```

Problem 1: Homogeneous Coordinate (Lines and Points)

The intersection of two lines l1 and l2, with l1 passing through the points (5,4) and (1,7), and l2 passing through the points (6,3) and (-3,8).

(a) Compute the intersection without using homogeneous coordinates
The equation for l1:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \quad \rightarrow \quad y - 4 = \frac{7-4}{1-5}(x-5) \quad \rightarrow \quad y = -\frac{3}{4}x + \frac{31}{4}$$

The equation for l2:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \quad \rightarrow \quad y - 3 = \frac{8-3}{-3-6}(x-6) \quad \rightarrow \quad y = -\frac{5}{9}x + \frac{57}{9}$$

Finding the intersection:

$$-\frac{3}{4}x + \frac{31}{4} = -\frac{5}{9}x + \frac{57}{9} \quad \rightarrow \quad \frac{7}{36}x = \frac{51}{36} \quad \rightarrow \quad x = \frac{51}{7}$$

$$y = -\frac{3}{4}x + \frac{31}{4} = -\frac{3}{4} * \frac{51}{7} + \frac{31}{4} = -\frac{153}{28} + \frac{217}{28} = \frac{16}{7}$$

(b) Compute the intersection using homogeneous coordinates Finding the equation of the lines:

$$l1 = x1 \times x2 = \begin{vmatrix} i & j & k \\ 5 & 4 & 1 \\ 1 & 7 & 1 \end{vmatrix} = \begin{bmatrix} -3 \\ -4 \\ 31 \end{bmatrix}$$

$$l2 = x1 \times x2 = \begin{vmatrix} i & j & k \\ 6 & 3 & 1 \\ -3 & 8 & 1 \end{vmatrix} = \begin{bmatrix} -5 \\ -9 \\ 57 \end{bmatrix}$$

Finding the intersection:

$$x = l1 \times l2 = \begin{bmatrix} -3 \\ -4 \\ 31 \end{bmatrix} \times \begin{bmatrix} -5 \\ -9 \\ 57 \end{bmatrix} = \begin{vmatrix} i & j & k \\ -3 & -4 & 31 \\ -5 & -9 & 57 \end{vmatrix} = \begin{bmatrix} 51 \\ 16 \\ 7 \end{bmatrix}$$

$$(x, y) = (\frac{x_1}{x_3}, \frac{x_2}{x_3}) = (\frac{51}{7}, \frac{16}{7})$$

Problem 2: Homogeneous Coordinate (Lines)

The four lines (l1, l2, l3, and l4) are intersected at six points. Among the six points, please find four points that form a quadrangle with your code, not manually.

$$l_1 : y = 0.59756x + 921.94$$
$$l_2 : y = -1.935x + 6050.3$$
$$l_3 : y = 0.51575x - 495.56$$
$$l_4 : y = -1.3858x + 1685.5$$

This script first finds the six intersection points of the four lines. Then, it generates all possible combinations of four points within the six intersection points. Next, combinations with three colllinear points are filtered out. Three combinations remain after filtering.

For each four point combination, three differerent quadrilaterals can be formed: two with the edges intersecting and one without edges intersecting. So there 3x3=9 different quadrilaterals remaining. For each of these quadrilaterals, the four edges are then checked. First, the edges are checked to see if they intersect one of the two intersection points not included in the quadrilateral to filter out the case where the quadrilaterals are complex (have edges intersecting). Next, the edges are checked for colllinearity with the four initial lines l1, l2, l3, l4. If any quadrilateral has all four edges which don't intersect any other points and are colllinear with one of the four lines, then that quadrilateral is the final solution.

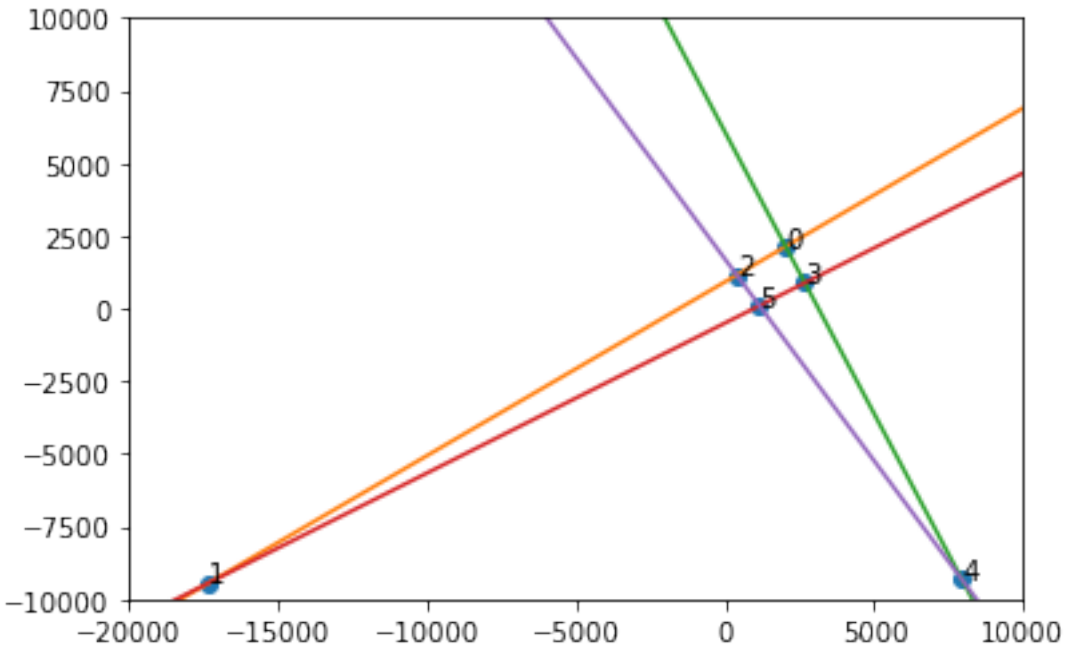
[30]:
```
# lines in homogenous coords
l1 = np.array([0.59756, -1, 921.94])
l2 = np.array([-1.935, -1, 6050.3])
l3 = np.array([0.51575, -1, -495.56])
l4 = np.array([-1.3858, -1, 1685.5])
lines = [l1, l2, l3, l4]

# find intersection points
pts = []
for i in range(0,4):
    for j in range(i+1, 4):
        pts.append(np.cross(lines[i], lines[j]))
```

```
# find all 14 combinations of 4 points within the 6 intersection points
comb = list(combinations(range(len(pts)), 4))

# determine which combinations contain three collinear points
isCollin = []
for com in comb:
    p0 = pts[com[0]]
    p1 = pts[com[1]]
    p2 = pts[com[2]]
    p3 = pts[com[3]]
    pts2 = [p0, p1, p2, p3]

    #check combination of points 0,1,2 0,1,3 0,2,3 1,2,3 for colinearity
    ln_comb = list(combinations(range(len(com)), 3))
    for ln_com in ln_comb:
        line1 = np.cross(pts2[ln_com[0]], pts2[ln_com[1]])
        line2 = np.cross(pts2[ln_com[0]], pts2[ln_com[2]])

        # collinear if the cross product is zero
        if np.all(abs(np.cross(line1, line2)) < 1e-3):
            idx = comb.index(com)
            isCollin.append(idx)
            break

# filter out combinations with 3 collinear points
comb = [comb[i] for i in range(len(comb)) if i not in isCollin]

quads = []
for com in comb:
    # for any four points, there are two ways to connect the points
    # either without edges intersecting  or with edges intersecting
    perm1 = [com[0], com[1], com[2], com[3], com[0]]
    perm2 = [com[0], com[1], com[3], com[2], com[0]]
    perm3 = [com[0], com[2], com[1], com[3], com[0]]
    perms = [perm1, perm2, perm3]

    for perm in perms:
        # pts and lines vectors for this quad permutation
        quad_pts = [pts[i] for i in perm]
        quad_edges = [np.cross(quad_pts[i], quad_pts[i+1]) for i in␣
↪range(len(perm)-1)]

        # check if edges are valid
        j=0
        valid_edges = []
        for edge in quad_edges:
```

```
            # check if point in the quadrilateral lies on edge
            other_pts = [pts[i] for i in range(len(pts)) if i not in perm]
            edge_pts = (pts[perm[j]], pts[perm[j+1]])
            y_edge_pts = [pt[1]/pt[2] for pt in edge_pts]

            isIntersect = False
            for pt in other_pts:
                #using cartesian coords, set y of point and line to be equal to␣
↪determine if point lies on line
                x_pt = pt[0]/pt[2]
                y_pt = pt[1]/pt[2]

                edge_eqn = edge/(-1*edge[1])
                y_edge = edge_eqn[0]*x_pt+edge_eqn[2] #y=ax+b where x is x␣
↪coord of other pt

                # if points lies on edge
                if(abs((y_pt - y_edge)) < 1e-3) and (y_pt<max(y_edge_pts)) and␣
↪(y_pt>min(y_edge_pts)):
                    isIntersect = True

                if (isIntersect): break

            # determine if edge lies on one the four lines l1, l2, l3, l4
            isLine = False
            for line in lines:
                if np.all(abs(np.cross(edge, line)) < 1e-3):
                    isLine = True

                if (isLine): break

            j = j+1
            valid_edges.append(not isIntersect and isLine)

        if all(valid_edges):
            quad = perm

# print(quad)
quad_coords = [pts[i] for i in quad[:-1]] #homogenous
quad_coords = [(pt[0]/pt[2], pt[1]/pt[2]) for pt in quad_coords] #cartesian
print("The coordinate points are:")
for coord in quad_coords:
    print(coord)


# plotting
```

```
x1 = np.arange(-20000, 10000)
y1 = 0.59756*x1 + 921.94
y2 = -1.935*x1 + 6050.3
y3 = 0.51575*x1 - 495.56
y4 = -1.3858*x1 + 1685.5
x = [pt[0]/pt[2] for pt in pts]
y = [pt[1]/pt[2] for pt in pts]

plt.plot(x,y, 'o')
plt.plot(x1, y1)
plt.plot(x1, y2)
plt.plot(x1, y3)
plt.plot(x1, y4)
for i in range(6):
    plt.annotate(str(i), (x[i], y[i]))
plt.xlim([-20000, 10000]), plt.ylim([-10000, 10000])
plt.show()
```

```
The coordinate points are:
(2024.970780554064, 2131.9815396278864)
(384.9830590513069, 1151.990476766699)
(1146.990612921038, 96.00040861402547)
(2670.9619504233397, 881.9886259308374)
```



Problem 3: Homography (10 points)

Image 1 and Image 2 capture a flat rectangular table from different viewpoints. The pixel locations of the four corners of the table on Image 1 are the locations computed in Problem 2. The perspective transformation from the points in image 1 to the ones in image 2 is:

$$H = \begin{bmatrix} 0.44973 & -0.3531 & 1147.5 \\ 0.2448 & 0.41441 & 96.5 \\ 0 & 0 & 1 \end{bmatrix}$$

Please find a line equation for each side of the table in Image 2. The line equation should be the form of y = ax + b.

From Problem 2, the pixel locations of the four corners in Image 1 are (2024.970780554064, 2131.9815396278864), (384.9830590513069, 1151.990476766699), (1146.990612921038, 96.00040861402547), (2670.9619504233397, 881.9886259308374)

$$p_0 = \begin{bmatrix} x_0' \\ y_0' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.44973 & -0.3531 & 1147.5 \\ 0.2448 & 0.41441 & 96.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2024.971 \\ 2131.982 \\ 1 \end{bmatrix} = \begin{bmatrix} 1305.387 \\ 1475.727 \\ 1 \end{bmatrix}$$

$$p_1 = \begin{bmatrix} x_1' \\ y_1' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.44973 & -0.3531 & 1147.5 \\ 0.2448 & 0.41441 & 96.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 384.983 \\ 1151.990 \\ 1 \end{bmatrix} = \begin{bmatrix} 913.871 \\ 668.140 \\ 1 \end{bmatrix}$$

$$p_2 = \begin{bmatrix} x_2' \\ y_2' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.44973 & -0.3531 & 1147.5 \\ 0.2448 & 0.41441 & 96.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1146.991 \\ 96.000 \\ 1 \end{bmatrix} = \begin{bmatrix} 1629.438 \\ 417.067 \\ 1 \end{bmatrix}$$

$$p_3 = \begin{bmatrix} x_3' \\ y_3' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.44973 & -0.3531 & 1147.5 \\ 0.2448 & 0.41441 & 96.5 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2670.962 \\ 881.989 \\ 1 \end{bmatrix} = \begin{bmatrix} 2037.282 \\ 1115.856 \\ 1 \end{bmatrix}$$

Next, the equations of the lines can be found.

$$l_0 = p_0 \times p_1 = \begin{vmatrix} i & j & k \\ 1305.387 & 1475.727 & 1 \\ 913.871 & 668.140 & 1 \end{vmatrix} = \begin{bmatrix} 807.59 \\ -391.52 \\ -476118.91 \end{bmatrix}$$

$$l_1 = p_1 \times p_2 = \begin{vmatrix} i & j & k \\ 913.871 & 668.140 & 1 \\ 1629.438 & 417.067 & 1 \end{vmatrix} = \begin{bmatrix} 251.07 \\ 715.57 \\ -707548.19 \end{bmatrix}$$

$$l_2 = p_2 \times p_3 = \begin{vmatrix} i & j & k \\ 1629.438 & 417.067 & 1 \\ 2037.282 & 1115.856 & 1 \end{vmatrix} = \begin{bmatrix} -698.79 \\ 407.84 \\ 968536.64 \end{bmatrix}$$

$$l_3 = p_3 \times p_0 = \begin{vmatrix} i & j & k \\ 2037.282 & 1115.856 & 1 \\ 1305.387 & 1475.727 & 1 \end{vmatrix} = \begin{bmatrix} -359.87 \\ -731.89 \\ 1549847.11 \end{bmatrix}$$

The line vectors can be converted to the form y=ax+b by dividing the line vector by a factor such that the second element is equal to -1. The first element would then be equal to a and the third element equal to b.

$$l_0 : \begin{bmatrix} 807.59 \\ -391.52 \\ -476118.91 \end{bmatrix} * \frac{1}{391.52} = \begin{bmatrix} 2.06 \\ -1 \\ -1216.9 \end{bmatrix} \rightarrow y = 2.06x - 1216.9$$

$$l_1 : \begin{bmatrix} 251.07 \\ 715.57 \\ -707548.19 \end{bmatrix} * \frac{1}{-715.57} = \begin{bmatrix} -0.351 \\ -1 \\ 988.79 \end{bmatrix} \rightarrow y = -0.351x + 988.79$$

$$l_2 : \begin{bmatrix} -698.79 \\ 407.84 \\ 968536.64 \end{bmatrix} * \frac{1}{-407.84} = \begin{bmatrix} 1.71 \\ -1 \\ -2374.8 \end{bmatrix} \rightarrow y = 1.71x - 2374.8$$

$$l_3 : \begin{bmatrix} -359.87 \\ -731.89 \\ 1549847.11 \end{bmatrix} * \frac{1}{731.89} = \begin{bmatrix} -0.492 \\ -1 \\ 2117.6 \end{bmatrix} \rightarrow y = -0.492x + 2117.6$$

Problem 4: Linear Algebra (20 points)

(a) Find a transpose of inverse H (include your derivation)

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

$$H = IH$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} H$$

Subtracting row 1 l1 times from row 3 and subtracting row 2 l2 times from row 3:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & l_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -l_1 & -l_2 & 1 \end{bmatrix} H$$

Dividing row 3 by l3:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{-l_1}{l_3} & \frac{-l_2}{l_3} & \frac{1}{l_3} \end{bmatrix} H$$

Therefore:

$$H^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{-l_1}{l_3} & \frac{-l_2}{l_3} & \frac{1}{l_3} \end{bmatrix} \rightarrow H^{-1^T} = \begin{bmatrix} 1 & 0 & \frac{-l_1}{l_3} \\ 0 & 1 & \frac{-l_2}{l_3} \\ 0 & 0 & \frac{1}{l_3} \end{bmatrix}$$

(b) What is the definition of a row and null space? Please find the rank and nullity of A.

$$A = \begin{bmatrix} 1 & 3 & 2 & 1 & 4 \\ 3 & 6 & 8 & 4 & -7 \\ -1 & 2 & 3 & 7 & 9 \\ 7 & 21 & 14 & 7 & 28 \end{bmatrix}$$

For some vector A, the row space is the subspace that is spanned by the row vectors of A and the null space is the set of all vectors X such that $Ax = 0$.

Using elementary row operations, A can be reduced to:

$$\begin{bmatrix} 1 & 0 & 0 & \frac{-66}{25} & \frac{-151}{25} \\ 0 & 1 & 0 & \frac{11}{25} & \frac{121}{25} \\ 0 & 0 & 1 & \frac{29}{25} & \frac{-56}{25} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Therefore:

$$rank(A) = 3$$

$$nullity(A) = n - rank(A) = 5 - 3 = 2$$

(c) You need to find a non-singular (non-trivial) vector of x that satisfy Ax = 0. Does x exist? What is the null space of A and the nullity of A? Please explain your answer. Please answer these questions for each A below.

(i)

The reduced row echelon form of $A = \begin{bmatrix} 1 & 3 & 2 & 1 \\ 3 & 6 & 8 & 4 \\ 7 & 21 & 14 & 7 \\ 1 & 21 & 14 & -3 \\ -1 & 2 & 3 & 7 \end{bmatrix}$ is $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$

Therefore $rank(A) = 4$ and $nullity(A) = 0$. Since $nullity(A) = 0$, no non-trivial solutions exist for $Ax = 0$

(ii)

The reduced row echelon form of $A = \begin{bmatrix} 1 & 3 & 2 & 1 \\ 3 & 6 & 8 & 4 \\ 1 & 21 & 14 & -3 \\ -1 & 2 & 3 & 7 \end{bmatrix}$ is $\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Therefore $rank(A) = 4$ and $nullity(A) = 0$. Since $nullity(A) = 0$, no non-trivial solutions exist for $Ax = 0$

(iii)

The reduced row echelon form of $A = \begin{bmatrix} 1 & 3 & 2 & 1 \\ 3 & 6 & 8 & 4 \\ 1 & 21 & 14 & -3 \end{bmatrix}$ is $\begin{bmatrix} 1 & 0 & 0 & \frac{5}{3} \\ 0 & 1 & 0 & \frac{-5}{18} \\ 0 & 0 & 1 & \frac{1}{12} \end{bmatrix}$

Therefore $rank(A) = 3$ and $nullity(A) = 1$. Since $nullity(A) > 0$, non-trivial solutions exist for $Ax = 0$.
In equation form, the reduced row echelon matrix is:

$$\begin{cases} x_1 = -\frac{5}{3}x_4 \\ x_2 = \frac{5}{18}x_4 \\ x_3 = -\frac{1}{12}x_4 \\ x_4 = x_4 \end{cases}$$

Then we can solve for the null space:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = c \begin{bmatrix} -\frac{5}{3} \\ \frac{5}{18} \\ -\frac{1}{12} \\ 1 \end{bmatrix} = c \begin{bmatrix} -60 \\ 10 \\ -3 \\ 36 \end{bmatrix}, c \in \mathbb{R}$$

Problem 5: Image Overlay (10 points)

You are going to hang a good painting or meme. Please find an image putting on the whiteboard and award frame, and a digital clock (see clock_wall.png) is placed on the designated block (marked as 'clock'). Your images and the clock image are projected to the marked regions in (a) and (b). Write your own code to automatically project your image. You should find homography for each image.

You can make your tool using the code provided or from the scratch. Note that you should not use fitgeotrans in MATLAB and findhomography, getPerspectiveTransform in Python. You need to write your own code to find homography (perspective transformation), meaning that you need to write your own ComputeH. The source code of ComputeH.p is obscured.

You need to manually provide the coordinates of four corners of each region on the image where your image overlay. Or, you can record a video to show your code is working if you are making an interactive code for picking the corner like the one in the demo.

[42]:
```
def ComputeH(new_pts, size):
    old_pts = np.array([(0,0), (size[0], 0), (size[0], size[1]), (0, size[1])])

    x = old_pts[:,0]
    y = old_pts[:,1]
    xp = new_pts[:,0]
    yp = new_pts[:,1]

    A = np.zeros((new_pts.shape[0]*2, 9))
    A[::2,0:2] = old_pts
```

```
    A[1::2,3:5] = old_pts
    A[::2,2] = 1
    A[1::2,5] = 1
    A[0::2,6] = -x*xp
    A[0::2,7] = -xp*y
    A[0::2,8] = -xp
    A[1::2,6] = -x*yp
    A[1::2,7] = -yp*y
    A[1::2,8] = -yp

    H = null_space(A)
    return H.reshape(3,3)

# code modified from https://pub.towardsai.net/
 ↪what-is-perspective-warping-opencv-and-python-750e7a13d386
def combineImages(warp_img, base_img, corners):
    mask = np.zeros(base_img.shape, dtype=np.uint8)
    roi_corners = np.int32(corners)
    cv2.fillConvexPoly(mask, roi_corners, (255, 255, 255))
    mask = cv2.bitwise_not(mask)
    masked_img = cv2.bitwise_and(base_img, mask)
    output = cv2.bitwise_or(warp, masked_img)
    return output
```

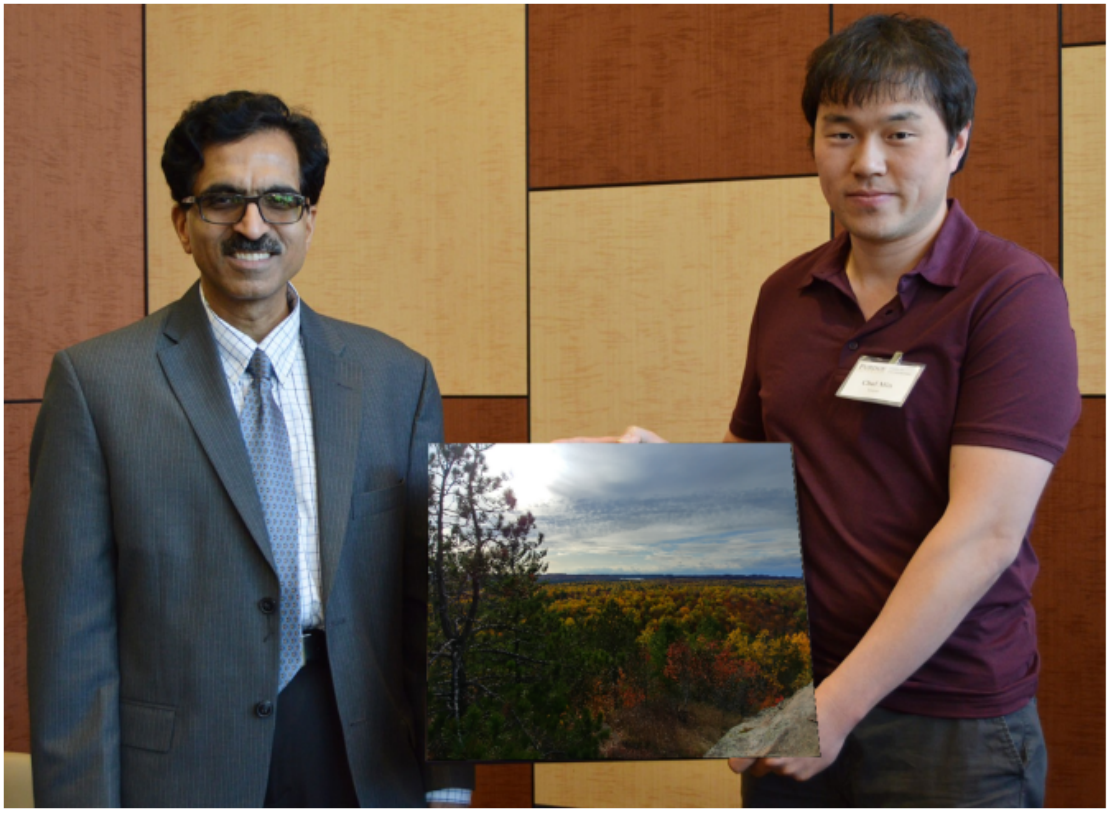
(a)

[50]:
```
base_img = cv2.imread('prob5_1.jpg')
board_img = cv2.imread('prob5_board.jpg')
clock_img = cv2.imread('clock_wall.png')

# board pic
corner = np.array([[100, 350], [650, 420], [650, 800], [100, 900]])
H = ComputeH(corner, (board_img.shape[1], board_img.shape[0]))
warp = cv2.warpPerspective(board_img, H, (base_img.shape[1], base_img.shape[0]))
output = combineImages(warp, base_img, corner)

# clock pic
corner = np.array([[1200, 462], [1318, 455], [1318, 495], [1200, 500]])
H = ComputeH(corner, (clock_img.shape[1], clock_img.shape[0]))
warp = cv2.warpPerspective(clock_img, H, (output.shape[1], output.shape[0]))
output = combineImages(warp, output, corner)


# plotting
plt.figure(figsize=(12,9))
plt.axis('off')
plt.imshow(output[:,:,::-1])
```

```
plt.show()
```

(b)

[51]:
```
base_img = cv2.imread('prob5_2.jpg')
award_img = cv2.imread('prob5_award.jpg')

corner = np.array([[570, 590], [1055, 590], [1095, 1010], [568, 1015]])
H = ComputeH(corner, (award_img.shape[1], award_img.shape[0]))
warp = cv2.warpPerspective(award_img, H, (base_img.shape[1], base_img.shape[0]))
output = combineImages(warp, base_img, corner)


# plotting
plt.figure(figsize=(12,9))
plt.axis('off')
plt.imshow(output[:,:,::-1])
plt.show()
```



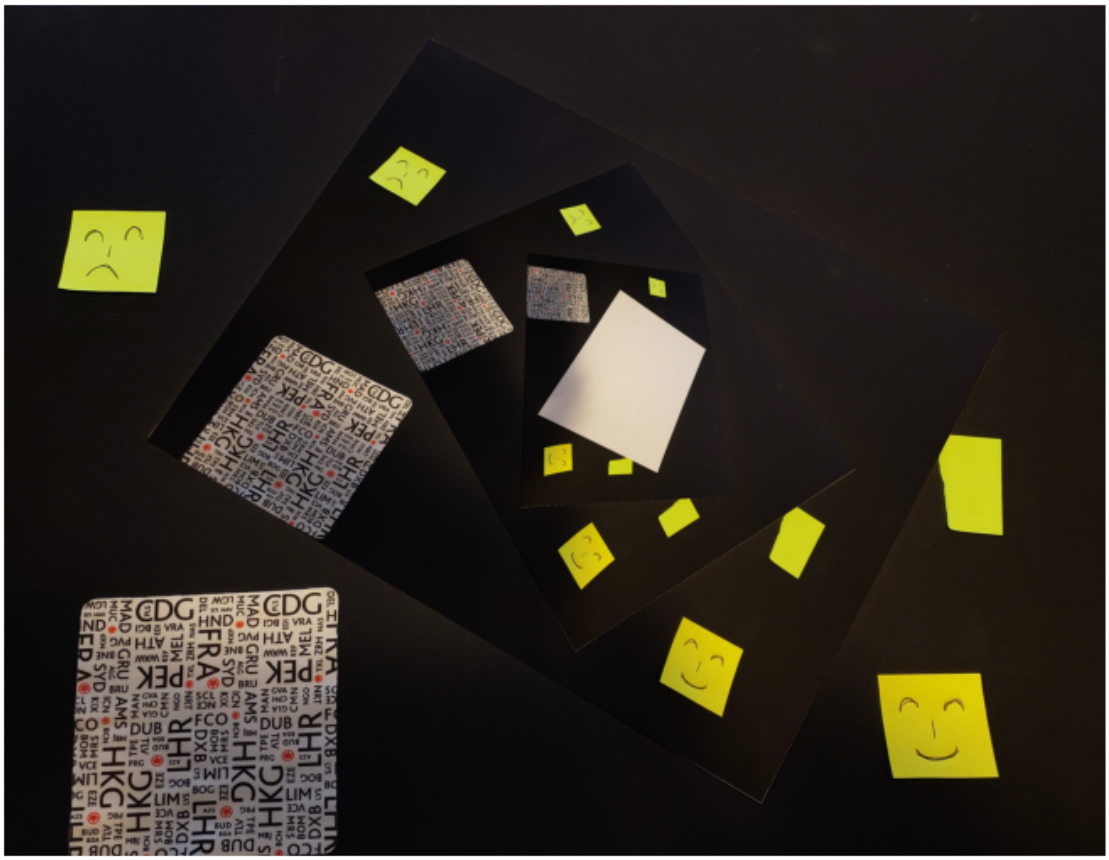
Problem 6: Picture in Picture in Picture in Picture (20 points)

You are going to overlay the given picture in the paper on the picture, and repeat this process three times more. Thus, this task is to call "picture in picture in picture in picture". Once you select the four corners of the paper, the resulting image in prob6_result.jpg should be automatically generated without any manual process. Your code must generate the final photo at once. The only input that you can provide is the four corners of the paper in the image. You should not use any manually inputted values of the other values. A sample code is not provided in this problem because you will reuse the code in Problem 5.

First, you need to show that your code is working for the given image of prob6.jpg. You should include the resulting images in the report. Next, you should demonstrate the functionality of your code using your own image like the one in the demo videos. Please think about how or what to take images to show the functions of your code. You can capture the video (like demo video) and include it in your submission or the link to the video.

[52]:
```
img = cv2.imread('prob6.jpg')

corner = np.array([[1135, 90], [2660, 870], [2017, 2125], [380, 1160]])
output = img.copy()
```

```
H_0 = ComputeH(corner, (img.shape[1], img.shape[0])) #original homography matrix

for i in range(3):
    # apply homography to image
    H = ComputeH(corner, (img.shape[1], img.shape[0]))
    warp = cv2.warpPerspective(img, H, (output.shape[1], output.shape[0]))
    output = combineImages(warp, output, corner)

    # compute new corner points
    new_pts = [np.dot(H_0, np.transpose(np.append(pt, 1))) for pt in corner]
    new_pts = [[pt[0]/pt[2], pt[1]/pt[2]] for pt in new_pts]
    corner = np.array(new_pts)


# plotting
plt.figure(figsize=(12,12*(img.shape[0]/img.shape[1])))
plt.axis('off')
plt.imshow(output[:,:,::-1])
plt.show()
```



Using my own image:

[53]:
```
img = cv2.imread('prob6_2.jpg')

corner = np.array([[540, 600], [3460, 580], [3650, 2250], [450, 2300]])
output = img.copy()

H_0 = ComputeH(corner, (img.shape[1], img.shape[0])) #original homography matrix

for i in range(3):
    # apply homography to image
    H = ComputeH(corner, (img.shape[1], img.shape[0]))
    warp = cv2.warpPerspective(img, H, (output.shape[1], output.shape[0]))
    output = combineImages(warp, output, corner)

    # compute new corner points
    new_pts = [np.dot(H_0, np.transpose(np.append(pt, 1))) for pt in corner]
    new_pts = [[pt[0]/pt[2], pt[1]/pt[2]] for pt in new_pts]
    corner = np.array(new_pts)


# plotting
plt.figure(figsize=(12,12*(img.shape[0]/img.shape[1])))
plt.axis('off')
plt.imshow(output[:,:,::-1])
plt.show()
```

Problem 7: Build your 3D Planar Measurement Software (20 points)

You are going to make a 3D planar measurement tool using homography.

A sample code is not provided in this problem because you will reuse the code in Problem 5. drawpolyline is useful for selecting the line for measurement.

You need to demonstrate the functionality of your tool using your own image as well as physical measurements. Any size of a calibration paper can be used. Please provide at least two measurements on the plane that the calibration paper is placed. You can capture the video (like demo video) and include it in your submission or the link to the video.

[56]:
```
def click_event(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        if len(pts) < 4:
            cv2.circle(img_copy, (x, y), 4, (0, 0, 255), -1)
            pts.append([x, y])
            isClosed = True if len(pts)>=4 else False
            poly_pts = np.array(pts).reshape((-1,1,2))
            cv2.polylines(img_copy, [poly_pts], isClosed, (0, 0, 0), 3)
            cv2.imshow('Select Corners of Paper', img_copy)

def click_event2(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        if len(pts1) < 2:
            cv2.circle(img_copy, (x, y), 4, (0, 0, 255), -1)
            pts1.append([x, y])
            poly_pts = np.array(pts1).reshape((-1,1,2))
            cv2.polylines(img_copy, [poly_pts], False, (0, 0, 0), 2)
            cv2.imshow('Measure Distances', img_copy)

        elif len(pts2) < 2:
            cv2.circle(img_copy, (x, y), 4, (0, 0, 255), -1)
            pts2.append([x, y])
            poly_pts = np.array(pts2).reshape((-1,1,2))
            cv2.polylines(img_copy, [poly_pts], False, (0, 0, 0), 2)
            cv2.imshow('Measure Distances', img_copy)



img = cv2.imread('prob7.jpg')
```

```
paper_size = [8.5, 11]

# Select 4 corners of paper (select short side first)
pts = []
img_copy = img.copy()
cv2.imshow('Select Corners of Paper', img_copy)
cv2.setMouseCallback('Select Corners of Paper', click_event)
cv2.waitKey(0)
cv2.destroyAllWindows()

# compute homography
H = ComputeH(np.array(pts), paper_size)
H_inv = np.linalg.inv(H)

# select edge to be measured
pts1 = []
pts2 = []
img_copy = img.copy()
cv2.imshow('Measure Distances', img_copy)
cv2.setMouseCallback('Measure Distances', click_event2)
cv2.waitKey(0)
cv2.destroyAllWindows()

# convert selected points to homogenous coordinates
pts1 = [[pt[0], pt[1], 1] for pt in pts1]
pts2 = [[pt[0], pt[1], 1] for pt in pts2]

# apply homography (x = inv(H)x')
real_pts1 = [np.dot(H_inv, pt) for pt in pts1]
real_pts2 = [np.dot(H_inv, pt) for pt in pts2]

# convert to cartesian coords
real_pts1 = [[pt[0]/pt[2], pt[1]/pt[2]] for pt in real_pts1]
real_pts2 = [[pt[0]/pt[2], pt[1]/pt[2]] for pt in real_pts2]
real_pts1 = np.array(real_pts1)
real_pts2 = np.array(real_pts2)

# calculate distances
dist1 = np.linalg.norm(real_pts1[1,:]-real_pts1[0,:])
dist2 = np.linalg.norm(real_pts2[1,:]-real_pts2[0,:])
print(dist1)
print(dist2)
```

```
68.0718494916691
24.035681753600134
```

The yoga mat measures 68x24 inches in reality.