Lab-2-Report

Occlusion detection

COURSE CODE: EE5175 (ISP)

Department of Electrical and Electronics Engineering

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

CHENNAI, TAMILNADU

Under the guidance of

Prof. A.N. Rajagopalan

By

EE21S048

Ms – 2nd Semester

Date of submission: 12-02-2022

Q1: Given are two aerial images (IMG1.png, IMG2.png) of an airport parking bay

These images were captured using two cameras placed at different locations and at different instants of time but overlooking the same area. It is known that the images are related by an in-plane

It is known that the images are related by an in-plane rotation and translation.

The following point correspondences are given:

Correspondence IMG1 (x, y) IMG2 (x, y)

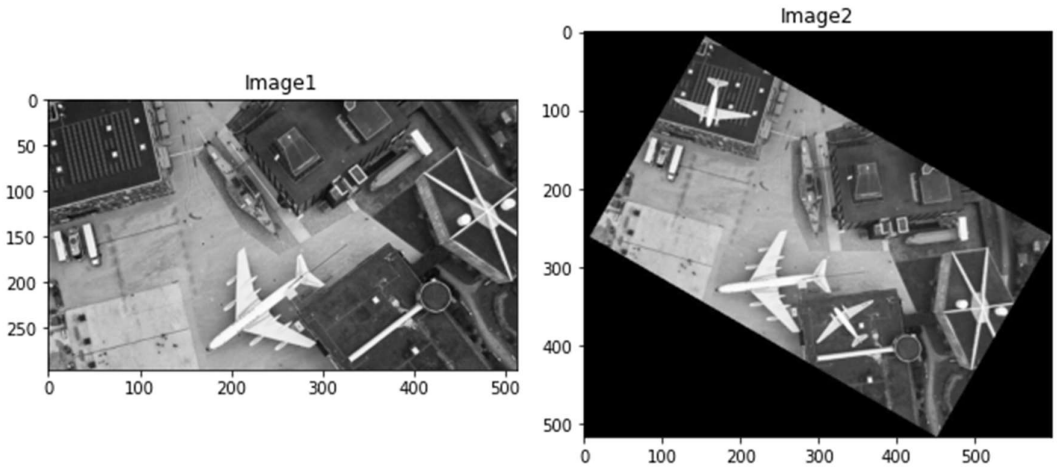
1 (29, 124) (93, 248)

2 (157, 372) (328, 399)

Determine the changes in IMG2 with respect to IMG1.

Python code:

```
from numpy import *
import sys
import math
import cv2
import matplotlib.pyplot as plt

#Reading the both images:
img1 = cv2.imread("IMG1.png",0)
img2 = cv2.imread("IMG2.png",0)

width, height= img1.shape   #finding the no' of pixels in x axis and y
ax#is of image1
print(width, height)
width2, height2=img2.shape  #finding the no' of pixels in x axis and y
a#xis of image2
print(width2,height2)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9,12), constrained_layout
=True)
ax1.imshow(img1,'gray')              #displaying gray scale image
ax1.title.set_text("Image1")         #setting title to the figure
ax2.imshow(img2,'gray')
ax2.title.set_text("Image2")
```



Calculating A,H matrices:

We have $$H = k\left[R + \frac{1}{d} * T * n^{-1}\right]$$

$T = \begin{bmatrix} Tx \\ Ty \\ 0 \end{bmatrix}$ is the translation vector for in-plane translation.

$n = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ is the normal matrix.

$R = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$ for in plane rotation.

Now:

$$\frac{1}{d} * T * n^{-1} = \begin{bmatrix} 0 & 0 & Tx/d \\ 0 & 0 & Ty/d \\ 0 & 0 & 0 \end{bmatrix}$$

$$H = k\left[R + \frac{1}{d} * T * n^{-1}\right] = k * [\begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & Tx/d \\ 0 & 0 & Ty/d \\ 0 & 0 & 0 \end{bmatrix}]$$

By solving we get

$$H = \begin{bmatrix} k * cos\theta & k * sin\theta & k * tx \\ -k * sin\theta & k * cos\theta & k * ty \\ 0 & 0 & k \end{bmatrix}$$

We have to solve for tx,ty,θ from corresponding points.

Here let

$$h_{11} = h_{22} = k * cos\theta$$

$$h_{12} = -h_{21} = k * sin\theta$$

$$h_{13} = k * tx \quad h_{23} = k * ty$$

$$h_{33} = k$$

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ -h_{12} & h_{11} & h_{23} \\ 0 & 0 & h_{33} \end{bmatrix}$$

Now we have 5 unknown h parameters in terms of θ and T. H has 4 degrees of freedom instead of the usual 3 degrees of freedom for a homography matrix corresponding to in-plane translation and in-plane rotation.

we are given a point (xi,yi) and its corresponding point (xi′,yi′). We know from the definition of H that $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ -h_{12} & h_{11} & h_{23} \\ 0 & 0 & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = \frac{h_{11} * x + \ h_{12} * y + \ h_{13}}{h_{33}}$$

$$y' = \frac{-h_{12} * x + \ h_{11} * y + \ h_{23}}{h_{33}}$$

We have

$$Ah = 0$$

$$A = \begin{bmatrix} x & y & 1 & 0 & -x' \\ -y & x & 0 & 1 & -y' \end{bmatrix} \quad h = \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{23} \\ h_{33} \end{bmatrix}$$

If we have ‘n’ such point correspondences, we built the A matrix by stacking up all A’s as following:

$$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \\ \vdots \end{bmatrix}$$

Given two point correspondings, therefore A matrix has 4 rows:

```
(x1,y1)=(29,124)
(x11,y11)=(93,248)
(x2,y2)=(157,372)
(x21,y21)=(328,399)
A[0]=[x1,y1,1,0,-x11]
A[1]=[y1,-x1,0,1,-y11]
```

```
A[2]=[x2,y2,1,0,-x21]
A[3]=[y2,-x2,0,1,-y21]
```

```
A
```

```
array([[  29.,  124.,    1.,    0.,  -93.],
       [ 124.,  -29.,    0.,    1., -248.],
       [ 157.,  372.,    1.,    0., -328.],
       [ 372., -157.,    0.,    1., -399.]])
```

SVD:

Singular value decomposition (SVD) is a matrix factorization method that generalizes the eigen decomposition of a square matrix (n x n) to any matrix (n x m).

SVD is similar to Principal Component Analysis (PCA), but more general. PCA assumes that input square matrix, SVD doesn't have this assumption. General formula of SVD is:

$\mathbf{A}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^t$, where:

- **A**-is original matrix we want to decompose
- **U**-is left singular matrix (columns are left singular vectors). **U** columns contain eigenvectors of matrix $\mathbf{A}\mathbf{A}^t$
- **Σ**-is a diagonal matrix containing singular (eigen)values
- **V**-is right singular matrix (columns are right singular vectors). **V** columns contain eigenvectors of matrix $\mathbf{A}^t\mathbf{A}$

```
# NumPy SVD gives singular values in decreasing order
u, s, v_T = linalg.svd(A)
# take the last row of v_transpose
h11, h12, h13, h23, h33 = v_T[-1]
# construct the appropriate 3x3 matrix
H = array([[h11, h12, h13], [-
h12, h11, h23], [0, 0, h33]]).reshape(3,3)
H_inv = linalg.inv(H)
```

Transforming the Image:

Here we have homography applied on img1.

$$img2 = H * img1$$
$$img1 = H^{-1} * img2$$

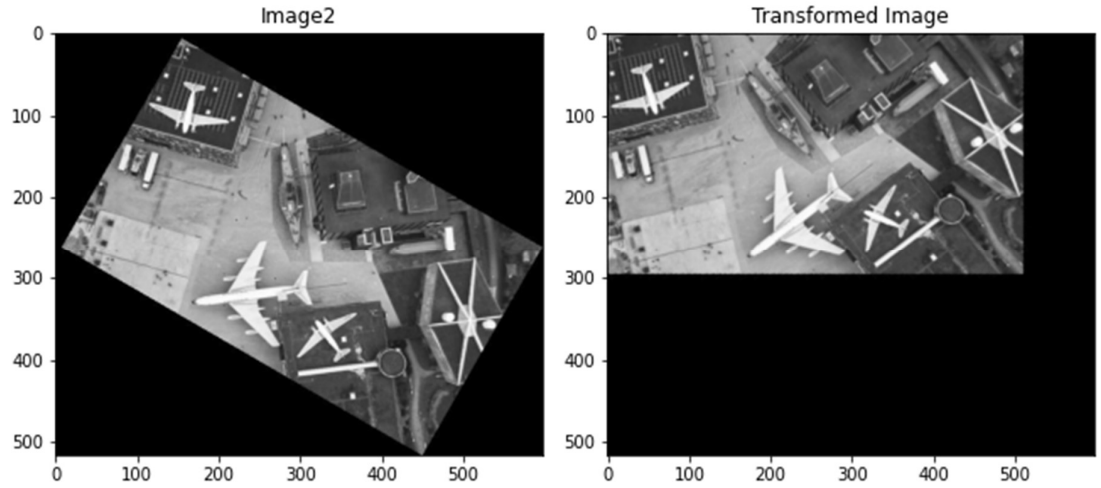
Target image or transformed image:

$$img_t = H^{-1} * img2$$
$$img2 = H * img_t$$

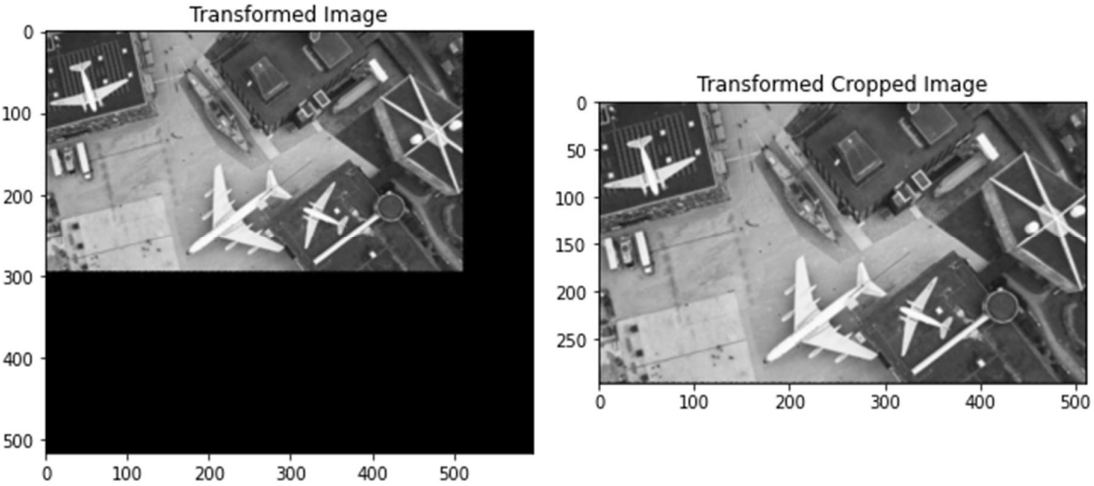
```
img_t=zeros((width2, height2)) #creating zeros 2d array for target imag
e which is of size of source image
print(img_t)
#adding zeros at four boundaries to the source the image and storing in
 other 2d array
img_padded = zeros((width2+2, height2+2))
img_padded[1:-1, 1:-1] = img2
#Assigning source intensity values from row,column 2 to -1 row,column
for xt in range(width2):        #traversing along rows of target image
  for yt in range(height2):     #traversing along columns of target img

    # convert to homogenous coordinates
    vec = array([xt, yt, 1])
    x_prime=dot(H,vec)
    if x_prime[-1]!=0:
      # return in non homogenous coordinates
      (xs,ys)=( x_prime[0]/x_prime[-1], x_prime[1]/x_prime[-1])
    else:
      (xs,ys)=(0,0)
    # Assign the value using bilinear interpolation

    x=xs+1
    y=ys+1
    xs1=math.floor(x)
    ys1=math.floor(y)
    a=x-xs1                    #distance to the landed x point from xs
    b=y-ys1                    #distance to the landed y point from ys

    #bilinear interpolation
    #assingning intensity to the target image by considering four neare
st neighbors
    if xs1>=0 and xs1<=width2 and ys1>=0 and ys1<=height2:
      img_t[xt,yt]=(1-a)*(1-b)*img_padded[xs1,ys1]+(1-
a)*b*img_padded[xs1,ys1+1]+a*(1-
b)*img_padded[xs1+1,ys1]+a*b*img_padded[xs1+1,ys1+1]
    else:
      img_t[xt,yt]=0       #assign 0 to the values where xs and ys dont
 exist.
```

Displaying the Images:

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9,12), constrained_layout
=True)
ax1.imshow(img2,'gray')
ax1.title.set_text("Image2")
ax2.imshow(img_t,'gray')
ax2.title.set_text("Transformed Image")
```



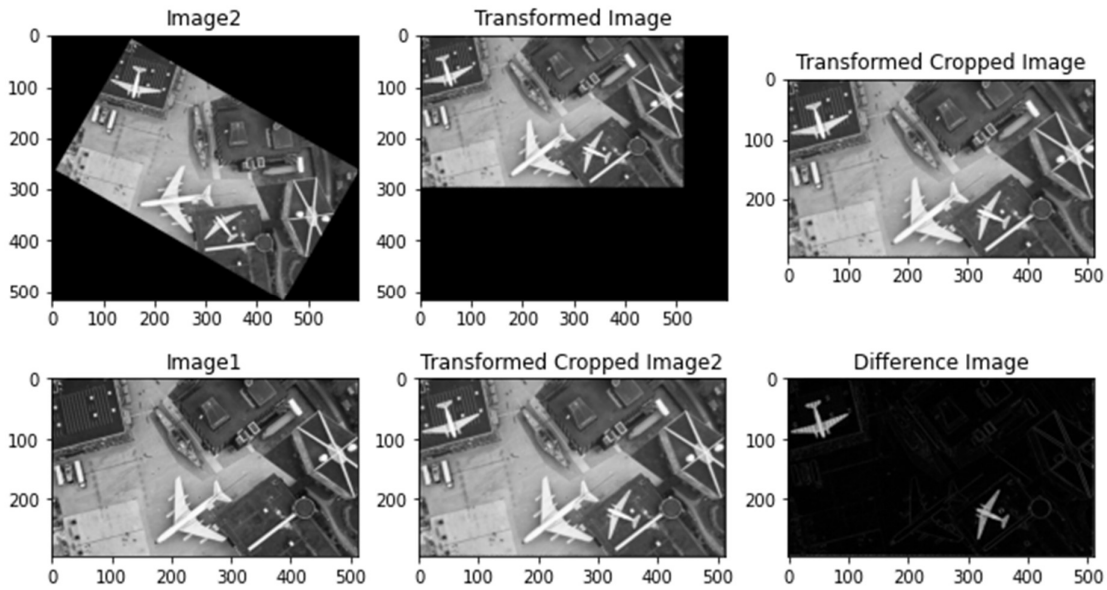

```
img_t2=img_t[:width,:height]
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9,12), constrained_layout
=True)
ax1.imshow(img_t,'gray')
ax1.title.set_text("transformed Image")
ax2.imshow(img_t2,'gray')
ax2.title.set_text("Transformed Cropped Image")
```




```
diff = abs(img1-img_t2)

fig, (ax1, ax2,ax3) = plt.subplots(1, 3, figsize=(9,12), constrained_la
yout=True)
ax1.imshow(img2,'gray')
ax1.title.set_text("Image2")
ax2.imshow(img_t,'gray')
ax2.title.set_text("Transformed Image")
ax3.imshow(img_t2,'gray')
ax3.title.set_text("Transformed Cropped Image")
```

```
fig, (ax4, ax5, ax6) = plt.subplots(1, 3, figsize=(9,12), constrained_l
ayout=True)
ax4.imshow(img1,'gray')
ax4.title.set_text("Image1")
ax5.imshow(img_t2,'gray')
ax5.title.set_text("Transformed Cropped Image2")
ax6.imshow(diff,'gray')
ax6.title.set_text("Difference Image")
```

Final Result:



Hence we can detect the two planes which are missing in Image1 by aligning Image2 onto it and finding difference between images.

Observations:

- If we try to align image1 onto image2 we loose information after transforming because size of image2 is greater than size of image1. So, it is important to select image to be transformed so that there is no loss of information.
- The difference of two images is not sufficient to record the changes between them. First alignment is needed (i.e. transformation of one image using homography) and difference would help to detect changes.

References:

[1] https://towardsdatascience.com/simple-svd-algorithms-13291ad2eef2

[2] https://numpy.org/doc/stable/reference/generated/numpy.linalg.svd.html