COMPUTER VISION

A Lab Report

submitted by

SOUMYA SARA JOHN

in partial fulfillment of the requirements
for the award of credits in CV : AVD863

MASTER OF TECHNOLOGY

DEPT. OF AVIONICS
INDIAN INSTITUTE OF SPACE SCIENCE AND TECHNOLOGY
Thiruvananthapuram - 695547

May 2018

Lab 1
Image Transformations

Question1:Perform the following image transformations:

(a) Translation
(b) Rotation
(c) Scaling

Theory

Image can be translated,rotated and scled by simple matrix multiplications.
For translation by tx in x-direction, ty in y-direction , new coordinates [x',y'] is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

For rotation along Z-axis by an angle θ, new coordinates [x',y'] is :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

For scaling by s1 and s2 , the new coordinates [x',y'] is : x' = x*s1
y' = y*s2

Program Code:

```
from PIL import Image as im
import numpy as np
from scipy.misc import toimage,imread

i1=im.open('cam.tif')
i1.save('cam.png')
i2=im.open('cam.png')
size=i2.size

tx=25
ty=4
t=np.array([[1,0,tx],[0,1,ty],[0,0,1]])

c=np.cos(np.pi/8)
s=np.sin(np.pi/8)
r=np.array([[c,-s,0],[s,c,0],[0,0,1]])
```

```
sx=2
sy=2

new=np.zeros((size[0],size[1]))
rotated=np.zeros((size[0],size[1]))
scale=np.zeros((sx*size[0],sy*size[1]))


for i in range(0,size[1]-1):
    for j in range(0,size[0]-1):
        hc=np.array([i,j,1])
        newhc=np.dot(t,hc)
        a=i2.getpixel((i,j))
        newr=np.dot(r,hc)
        s1=sx*i
        s2=sy*j

        if(newhc[0]>0 and newhc[0]<255 and newhc[1]>0 and newhc[1]<255):
            new[newhc[1],newhc[0]]=a
        if(newr[0]>0 and newr[0]<255 and newr[1]>0 and newr[1]<255):
            rotated[newr[1],newr[0]]=a
        if(s1>0 and s1<255*sx and s2>0 and s2<255*sy):
            scale[s2,s1]=a

toimage(new).show()
toimage(new).save('Trans.png')
toimage(rotated).show()
toimage(rotated).save('Rotated.png')
toimage(scale).show()
toimage(scale).save('Scaled.png')
```

Results

(a) Origianl Image

(b) Translated Image

(c) Rotated Image

Figure 2: Scaled Image

Comments

By using homogeneous coordinate system,the different transformations-translation,rotation and scaling have become simple matrix operations.

Question2: Affine Transformation

Theory

Steps:

1. Given two images img1, img2, manually select three pairs of corresponding points (A1,A1'),(B1,B1') and (C1,C1') from the images.

2. Compute the affine transformation matrix

$$\begin{bmatrix} A1 & A2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & A1 & A2 & 1 \\ B1 & B2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & B1 & B2 & 1 \\ C1 & C2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & C1 & C2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} A1' \\ A2' \\ B1' \\ B2' \\ C1' \\ C2' \end{bmatrix}$$

   where Affine matrix is given by:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}$$

3. Apply the transformation on any one of the images and verify.

Program Code:

```
from PIL import Image
import numpy as np
```

```
from skimage.transform import AffineTransform
from numpy.linalg import inv
from scipy.misc import toimage

p1=np.array([646,263])
p2=np.array([411,251])
p3=np.array([526,195])

P1=np.array([418,321])
P2=np.array([188,304])
P3=np.array([307,252])

T=np.array([[p1[0],p1[1],1,0,0,0],
            [0,0,0,p1[0],p1[1],1],
            [p2[0],p2[1],1,0,0,0],
            [0,0,0,p2[0],p2[1],1],
            [p3[0],p3[1],1,0,0,0],
            [0,0,0,p3[0],p3[1],1]])
p=np.array([P1[0],P1[1],P2[0],P2[1],P3[0],P3[1]])

Tinv=inv(T)
res=np.dot(Tinv,p)

rmat=np.reshape(res,(2,3))

im=Image.open('Image1.jpg')
size=im.size

newmat=np.zeros((size[0],size[1],3))

for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        a=im.getpixel((i,j))
        hc=np.array([i,j,1])
        newhc=np.dot(rmat,hc)
        if(newhc[0]>0 and newhc[1]<size[1]-1 and newhc[1]>0 and newhc[0]<size[0]-1):
            newmat[newhc[0],newhc[1]]=a

toimage(np.transpose(newmat)).show()
toimage(newmat).save('affine.png')
```

Result

Affine Transformation matrix

```
array([[  9.84044017e-01,  -1.04195323e-01,  -1.90289065e+02],
       [  2.25584594e-02,   9.74896836e-01,   5.00293673e+01]])
```

(a) Image 1

(b) Image 2

(c) Affine transformation of Image 1

Discussion

Affine transformations are a linear mapping method that preserves points,straight lines and planes.It means that parallel lines will remain parallel even after transformation. The affine transformation technique is typically used to correct for geometric distortions or deformations that occur with non-ideal camera angles.

Question 3:Perspective Transformation(Homography)

Theory

Steps:

1. Given two images img1, img2, manually select three pairs of corresponding points (a1,A1),(a2,A2),(a3,A3),(a4,A4),(b and (b4,B4) from the images.

2. Compute the perspective transformation matrix

$$\begin{bmatrix} a1 & b1 & 1 & 0 & 0 & 0 & -a1A1 & -b1A1 \\ a2 & b2 & 1 & 0 & 0 & 0 & -a2A2 & -b2A2 \\ a3 & b3 & 1 & 0 & 0 & 0 & -a3A3 & -b3A3 \\ a4 & b4 & 1 & 0 & 0 & 0 & -a4A4 & -b4A4 \\ 0 & 0 & 0 & a1 & b1 & 1 & -a1B1 & -b1B1 \\ 0 & 0 & 0 & a2 & b2 & 1 & -a2B2 & -b2B2 \\ 0 & 0 & 0 & a3 & b3 & 1 & -a3B3 & -b3B3 \\ 0 & 0 & 0 & a4 & b4 & 1 & -a4B4 & -b4B4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} A1 \\ A2 \\ A3 \\ A4 \\ B1 \\ B2 \\ B3 \\ B4 \end{bmatrix}$$

where homography matrix is given by:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix}$$

3. Apply the transformation on any one of the images and verify.

Program Code:

```
from PIL import Image
import numpy as np
from skimage.transform import AffineTransform
from numpy.linalg import inv
from scipy.misc import toimage

p1=np.array([646,263])
p2=np.array([411,251])
p3=np.array([526,195])
p4=np.array([650,195])

P1=np.array([418,321])
P2=np.array([188,304])
P3=np.array([307,252])
P4=np.array([421,258])

T=np.array([[p1[0],p1[1],1,0,0,0,-(p1[0]*P1[0]),-(p1[1]*P1[0])],
            [p2[0],p2[1],1,0,0,0,-(p2[0]*P2[0]),-(p2[1]*P2[0])],
            [p3[0],p3[1],1,0,0,0,-(p3[0]*P3[0]),-(p3[1]*P3[0])],
            [p4[0],p4[1],1,0,0,0,-(p4[0]*P4[0]),-(p4[1]*P4[0])],
            [0,0,0,p1[0],p1[1],1,-(p1[0]*P1[1]),-(p1[1]*P1[1])],
            [0,0,0,p2[0],p2[1],1,-(p2[0]*P2[1]),-(p2[1]*P2[1])],
            [0,0,0,p3[0],p3[1],1,-(p3[0]*P3[1]),-(p3[1]*P3[1])],
            [0,0,0,p4[0],p4[1],1,-(p4[0]*P4[1]),-(p4[1]*P4[1])]])
p=np.array([P1[0],P2[0],P3[0],P4[0],P1[1],P2[1],P3[1],P4[1]])

D=np.dot(np.transpose(T),T)
Dinv=inv(D)
D1=np.dot(np.transpose(T),p)
res=np.dot(Dinv,D1)

hom_mat=np.array([[res[0],res[1],res[2]],
                  [res[3],res[4],res[5]],
                  [res[6],res[7],1]])

im=Image.open('Image1.jpg')
size=im.size

newmat=np.zeros((size[0],size[1],3))
```

```
for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        a=im.getpixel((i,j))
        hc=np.array([i,j,1])
        newhc=np.dot(hom_mat,hc)
        x=newhc[0]/newhc[2]
        y=newhc[1]/newhc[2]
        #print(x)
        #print(y)
        if(x>0 and x<size[0]-1 and y>0 and y<size[1]-1):
            newmat[x,y]=a
toimage(np.transpose(newmat)).save('homographym.png')
```

Result

Perspective Transformation matrix

```
array([[  9.66654331e-01,  -4.39598475e-02,  -1.96214948e+02],
       [  2.67743191e-02,   9.58481429e-01,   5.22592130e+01],
       [ -7.14619329e-06,   1.84932904e-05,   1.00000000e+00]])
```

(a) Image 1

(b) Image 2

(c) Perspective transformation on Image 1

Discussion

A homography is an invertible mapping of points and lines on the projective plane ¶^{2} to itself such that three points lie on the same line if and only if their mapped points are also collinear. H is a homogeneous matrix and only has 8 degrees of freedom even though it contains 9 elements. This means that there are 8 unknowns that need to be solved for.

Lab 2
Image Enhancement

---

1. Given an image along with camera coordinates and corresponding world coordinates; Estimate the projection matrix

(a) Consider all the points given and obtain projection matrix
(b) Consider non planar points and obtain projection matrix
(c) Randomly choose 4 non planar points and 2 planar points on two planes and obtain projection matrix
(d) Calculate the Reprojection errors for all points using the projection matrices obtained above.Which method is good and why?
(e) Mark the given points and reprojected points for all the methods

Discussion

Camera model in general is a mapping from world to image coordinates. Extrinsic Parameters define the location and orientation of the camera with respect to the world frame.Intrinsic Parameters allow a mapping between camera coordinates and pixel coordinates in the image frame.

Program Code:

Program Code:

(a)

```
from PIL import Image
import numpy as np
from skimage.transform import AffineTransform
from numpy.linalg import inv
from scipy.misc import toimage

p1=np.array([[254, 532],
             [280, 432],
             [294, 480],
             [182, 584],
             [345, 575],
             [458, 652],
             [467, 355],
             [296, 363],
             [420, 508],
             [350, 415],
             [354, 465],
             [330, 472],
             [326, 425],
             [310, 524],
```

```
            [ 333, 520],
            [384, 512],
            [460, 502],
            [508, 568],
            [498, 425],
            [176, 455],
            [138, 495],
            [138, 454],
            [178, 495],
            [96,  495],
            [96,  454],
            [180, 535],
            [218, 534],
            [138, 537],
            [215, 455],
            [214, 417]])

P1=np.array([[0,    0,   0],
            [108,  24,  72],
        [108,  12,  36],
        [180, 126, -36],
        [288,  72, -36],
        [324,  36, -72],
        [324,  15,  72],
        [270,  68,  90],
        [252,  0,   0],
        [180,  0,   72],
        [180,  0,   36],
        [144,  0,   36],
        [144,  0,   72],
        [108,  0,   0],
        [144,  0,   0],
        [216,  0,   0],
        [288,  0,   0],
        [324,  0, -36],
        [324,  0,  36],
        [0,   72,  72],
        [0,  108,  36],
        [0,  108,  72],
        [0,   72,  36],
        [0,  144,  36],
        [0,  144,  72],
        [0,   72,   0],
        [0,   36,   0],
        [0,  108,   0],
        [0,   36,  72],
        [0,   36, 108]])
t=np.zeros((30,11))
m=np.zeros((30,11))
n=np.zeros((60,11))
for i in range(0,30):
    t[i]=np.array([P1[i][0],P1[i][1],P1[i][2],1,0,0,0,0,-(p1[i][0]*P1[i][0]),-(p1[i][0]*P1[i][1]),-(
    m[i]=np.array([0,0,0,0,P1[i][0],P1[i][1],P1[i][2],1,-(p1[i][1]*P1[i][0]),-(p1[i][1]*P1[i][1]),-(

n=np.concatenate((t,m))

q1=np.dot(np.transpose(n),n)
q1inv=inv(q1)
```

```
r=np.array([p1[:,0],p1[:,1]])
r=r.reshape((1,60))
q2=np.dot(np.transpose(n),np.transpose(r))
q=np.dot(q1inv,q2)
q=np.append(q,1)
q=q.reshape((3,4))
```

(b)

```
from PIL import Image
import numpy as np
from skimage.transform import AffineTransform
from numpy.linalg import inv
from scipy.misc import toimage

p1=np.array([[254, 532],
             [280, 432],
             [294, 480],
             [182, 584],
             [345, 575],
             [458, 652]])
P1=np.array([[0,    0,   0],
             [108,  24,  72],
         [108,  12,  36],
         [180, 126, -36],
         [288,  72, -36],
         [324,  36, -72]])
t=np.zeros((6,11))
m=np.zeros((6,11))
n=np.zeros((12,11))
for i in range(0,6):
    t[i]=np.array([P1[i][0],P1[i][1],P1[i][2],1,0,0,0,0,-(p1[i][0]*P1[i][0]),-(p1[i][0]*P1[i][1]),-(
    m[i]=np.array([0,0,0,0,P1[i][0],P1[i][1],P1[i][2],1,-(p1[i][1]*P1[i][0]),-(p1[i][1]*P1[i][1]),-(

n=np.concatenate((t,m))

q1=np.dot(np.transpose(n),n)
q1inv=inv(q1)

r=np.array([p1[:,0],p1[:,1]])
r=r.reshape((1,12))
q2=np.dot(np.transpose(n),np.transpose(r))
q=np.dot(q1inv,q2)
q=np.append(q,1)
q=q.reshape((3,4))
```

(c)

```
from PIL import Image
import numpy as np
from skimage.transform import AffineTransform
from numpy.linalg import inv
from scipy.misc import toimage

p1=np.array([[254, 532],
             [280, 432],
             [294, 480],
```

```
            [182, 584],
            [350, 415],
            [354, 465]])
P1=np.array([[0,    0,   0],
            [108,  24,  72],
        [108,  12,  36],
        [180, 126, -36],
        [180,  0,   72],
        [180,  0,   36]])
t=np.zeros((6,11))
m=np.zeros((6,11))
n=np.zeros((12,11))
for i in range(0,6):
    t[i]=np.array([P1[i][0],P1[i][1],P1[i][2],1,0,0,0,0,-(p1[i][0]*P1[i][0]),-(p1[i][0]*P1[i][1]),-(
    m[i]=np.array([0,0,0,0,P1[i][0],P1[i][1],P1[i][2],1,-(p1[i][1]*P1[i][0]),-(p1[i][1]*P1[i][1]),-(

n=np.concatenate((t,m))

q1=np.dot(np.transpose(n),n)
q1inv=inv(q1)

r=np.array([p1[:,0],p1[:,1]])
r=r.reshape((1,12))
q2=np.dot(np.transpose(n),np.transpose(r))
q=np.dot(q1inv,q2)
q=np.append(q,1)
q=q.reshape((3,4))
```

(d)

```
%-------------------------------------------------------------------------------------------------------
```

(e)

```
import numpy as np
from numpy.linalg import inv
from scipy.misc import toimage

p1=np.array([[254, 532],
            [280, 432],
            [294, 480],
            [182, 584],
            [345, 575],
            [458, 652],
            [467, 355],
            [296, 363],
            [420, 508],
            [350, 415],
            [354, 465],
            [330, 472],
            [326, 425],
            [310, 524],
            [333, 520],
            [384, 512],
            [460, 502],
            [508, 568],
            [498, 425],
            [176, 455],
```

```
            [138, 495],
            [138, 454],
            [178, 495],
            [96,  495],
            [96,  454],
            [180, 535],
            [218, 534],
            [138, 537],
            [215, 455],
            [214, 417]])

P1=np.array([[0,    0,   0],
            [108,  24,  72],
        [108,  12,  36],
        [180, 126, -36],
        [288,  72, -36],
        [324,  36, -72],
        [324,  15,  72],
        [270,  68,  90],
        [252,  0,   0],
        [180,  0,   72],
        [180,  0,   36],
        [144,  0,   36],
        [144,  0,   72],
        [108,  0,   0],
        [144,  0,   0],
        [216,  0,   0],
        [288,  0,   0],
        [324,  0, -36],
        [324,  0,  36],
        [0,   72,  72],
        [0,  108,  36],
        [0,  108,  72],
        [0,   72,  36],
        [0,  144,  36],
        [0,  144,  72],
        [0,   72,   0],
        [0,   36,   0],
        [0,  108,   0],
        [0,   36,  72],
        [0,   36, 108]])

Ma=np.array([[5.716989415127926577e-02, -1.141700813022049843e+00, -2.458815790703283710e-03, 2.5786
[-7.995415726013028745e-01 ,-1.264394917754501080e-01 ,-9.516921125177546514e-01 ,5.3114888256105223
[-1.396640523543418028e-03 ,-3.205276006551550927e-04 ,2.712173972925363552e-04 ,1.00000000000000000

Mb=np.array([[-6.327924752554281440e-02,-1.005058551069396344e+00,-5.679743737550779770e-02,2.542233
[-1.004152222872903621e+00,-1.128064129397898796e-01 ,-1.158986295145950862e+00,5.325118203893864575
[-1.753126683513794859e-03 ,-3.552331485630055852e-04,-2.902280078140861974e-04 ,1.00000000000000000

Mc=np.array([[-1.374854680558996733e-01,-1.465568090385204414e+00,-6.488177504616032820e-02
,2.542172602457708308e+02],
[-1.124988759052143905e+00 ,-1.666379017580766231e+00 ,-8.754098657973372610e-01 ,5.3212036665719142
[-1.992845070028170795e-03 ,-3.034304671473364579e-03 ,4.850810686463091770e-06 ,1.00000000000000000

z=np.zeros((30,4))
for i in range(0,30):
    z[i]=np.append(P1[i],1)
```

```
newpa=np.dot(Ma,np.transpose(z))
newpa=np.transpose(newpa)
newpb=np.dot(Mb,np.transpose(z))
newpb=np.transpose(newpb)
newpc=np.dot(Mc,np.transpose(z))
newpc=np.transpose(newpc)


p1newa=np.zeros((30,2))
p1newb=np.zeros((30,2))
p1newc=np.zeros((30,2))
for i in range(0,30):
    for j in range(0,2):
        p1newa[i][j]=newpa[i][j]/newpa[i][2]
        p1newb[i][j]=newpb[i][j]/newpb[i][2]
        p1newc[i][j]=newpc[i][j]/newpc[i][2]

print('Reprojected coordinates:')
print('(a)')
print(p1newa)
print('(b)')
print(p1newb)
print('(c)')
print(p1newc)
```

Results

(a)

```
array([[  5.71698942e-02,  -1.14170081e+00,  -2.45881579e-03,
          2.57862549e+02],
       [ -7.99541573e-01,  -1.26439492e-01,  -9.51692113e-01,
          5.31148883e+02],
       [ -1.39664052e-03,  -3.20527601e-04,   2.71217397e-04,
          1.00000000e+00]])
```

(b)

```
array([[ -6.32792475e-02,  -1.00505855e+00,  -5.67974374e-02,
          2.54223327e+02],
       [ -1.00415222e+00,  -1.12806413e-01,  -1.15898630e+00,
          5.32511820e+02],
       [ -1.75312668e-03,  -3.55233149e-04,  -2.90228008e-04,
          1.00000000e+00]])
```

(c)

```
array([[ -1.37485468e-01,  -1.46556809e+00,  -6.48817750e-02,
          2.54217260e+02],
       [ -1.12498876e+00,  -1.66637902e+00,  -8.75409866e-01,
          5.32120367e+02],
       [ -1.99284507e-03,  -3.03430467e-03,   4.85081069e-06,
          1.00000000e+00]])
```

(d)

```
M1-->
(p1-p1new)
array([[-3.862549  ,  0.851117  ],
       [ 5.36626743, -1.49937375],
       [ 1.3398078 ,  1.65873299],
       ...,
       [-1.38391173,  0.95015488],
       [ 0.13222031,  0.55517556],
       [ 1.28053149,  0.57820862]])

error sum=-0.465145353677

M2-->
(p1-p1new)
array([[ -0.22332741,  -0.51182039],
       [ -0.55176655,  -0.52890377],
       [  0.91202493,   1.34484743],
       ...,
       [-13.48890293,  -4.08765302],
       [ -6.40995638,  -5.51617613],
       [ -7.69098878,  -4.89998956]])
error sum=-223.281436527

M3-->
(p1-p1new)
array([[ -0.21726025,  -0.12036666],
       [ -0.11242151,   0.16012282],
       [  0.83259719,   0.24963588],
       ...,
       [ -4.69910212,  13.19514834],
       [ -5.83061999,  -4.08949748],
       [ -4.16671683,  -6.64091857]])
error sum=-401.841765073
```

(e)

```
Reprojected coordinates:
(a)
[[ 257.86254908  531.14888256]
 [ 274.63373269  433.49937352]
 [ 292.66019238  478.34126677]
 ...,
 [ 139.38391147  536.04984467]
 [ 214.86777966  454.44482403]
 [ 212.71946847  416.42179099]]
(b)
[[ 254.22332741  532.51182039]
 [ 280.55176655  432.52890377]
 [ 293.08797507  478.65515257]
 ...,
 [ 151.48890293  541.08765302]
 [ 221.40995638  460.51617613]
 [ 221.69098878  421.89998956]]
(c)
[[ 254.21726025  532.12036666]
 [ 280.11242151  431.83987718]
 [ 293.16740281  479.75036412]
...,
```

```
 [ 142.69910212  523.80485166]
 [ 220.83061999  459.08949748]
 [ 218.16671683  423.64091857]]
```

2. Estimate the intrinsic and extrinsic parameters for the above projection matrices.

Program Code:

```
import numpy as np
M=np.array([[-1.37485468055899673e-01,-1.46556809038520441e+00,-6.48817750461603282e-02
,2.54217260245770830e+02],
[-1.12498875905214390e+00 ,-1.66637901758076623e+00 ,-8.75409865797337261e-01 ,5.3212036665719142
[-1.99284507002817079e-03 ,-3.03430467147336457e-03 ,4.85081068646309177e-06 ,1.00000000000000000


gamma_abs=np.sqrt((np.square(M[2][0]))+(np.square(M[2][2]))+(np.square(M[2][2])))
M=M/gamma_abs

Tz=M[2][3]
r31=M[2][0]
r32=M[2][1]
r33=M[2][2]

q1=np.array([M[0][0],M[0][1],M[0][2]])
q2=np.array([M[1][0],M[1][1],M[1][2]])
q3=np.array([M[2][0],M[2][1],M[2][2]])

ox=np.dot(np.transpose(q1),q3)
oy=np.dot(np.transpose(q2),q3)

int1=np.dot(np.transpose(q1),q1)
int2=np.dot(np.transpose(q2),q2)

fx=np.sqrt(int1-np.square(ox))
fy=np.sqrt(int2-np.square(oy))

r11=(ox*M[2][0] - M[0][0])/fx
r12=(ox*M[2][1] - M[0][1])/fx
r13=(ox*M[2][2] - M[0][2])/fx

r21=(ox*M[2][0] - M[1][0])/fy
r22=(ox*M[2][1] - M[1][1])/fy
r23=(ox*M[2][2] - M[1][2])/fy

Tx=(ox*M[2][3] - M[0][3])/fx
Ty=(oy*M[2][3] - M[1][3])/fy

print('Camera Parameters:')
print('Intrinsic Parameters:')
I=np.array([[-fx,0,ox],[0,-fy,oy],[0,0,1]])
print(I)
print('Extrinsic Parameters:')
R=np.array([[r11,r12,r13],[r21,r22,r23],[r31,r32,r33]])
print('Rotation Matrix:')
print(R)
T=np.array([[1,0,0,Tx],[0,1,0,Ty],[0,0,1,Tz],[0,0,0,1]])
```

```
print('Translation matrix:')
print(T)
print('Gamma=')
print(gamma_abs)
```

Result

```
Camera Parameters:
Intrinsic Parameters:
[[             nan   0.00000000e+00   1.18863858e+03]
 [  0.00000000e+00              nan   1.83659144e+03]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00]]
Extrinsic Parameters:
Rotation Matrix:
[[        nan         nan         nan]
 [        nan         nan         nan]
 [-0.99999408 -1.52259036  0.0024341 ]]
Translation matrix:
[[   1.            0.            0.                  nan]
 [   0.            1.            0.                  nan]
 [   0.            0.            1.          501.79218153]
 [   0.            0.            0.            1.        ]]
Gamma=
0.00199285687742
```

3. Find the location of the camera from projection matrices obtained from question 1

4. How do you find the orientation of the camera from projection matrix? Find the surface normal for all the three matrices and state your observations with inferences.

The camera marix is given by,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}$$

The surface normal for the projection matrix is the normal to the plane given by

$$a_{41}x + a_{42}y + a_{43}z + a_{44} = 0 \tag{1}$$

Lab 3
Image Stitching and Corner Detection

1.Image Stitching

Write a program to find the underlying transformation between the images provided. Assuming the transformation is affine, try to align those two images. Decide the point correspondences manually. Perform stitching operation using built-in function and compare your results.

Discussion

P1,P2,P3 are the points in first image. p1,p2,p3 are the corresponding points in the second iimage. Affine transformation matrix was calculated to align Image 2. Both Image1 and Image 2 are of size 800 x 600. 258 pixels more are required in width to create a stitched image. Affine transformation was done on Image 2 and the resultant image was saved as 'affine1.png' . 258 x 600 pixel values from Image 1 and all the pixel values from affine1 are used to form the stitched image.

Program Code:

```
from PIL import Image
import numpy as np
from skimage.transform import AffineTransform
from numpy.linalg import inv
from scipy.misc import toimage
P1=np.array([646,263])
P2=np.array([411,251])
P3=np.array([526,195])

p1=np.array([418,321])
p2=np.array([188,304])
p3=np.array([307,252])

T1=np.array([[p1[0],p1[1],1,0,0,0],
             [0,0,0,p1[0],p1[1],1],
             [p2[0],p2[1],1,0,0,0],
             [0,0,0,p2[0],p2[1],1],
             [p3[0],p3[1],1,0,0,0],
             [0,0,0,p3[0],p3[1],1]])
p1=np.array([P1[0],P1[1],P2[0],P2[1],P3[0],P3[1]])

Tinv1=inv(T1)
res1=np.dot(Tinv1,p1)

rmat1=np.reshape(res1,(2,3))
im2=Image.open('Image2.jpg')
size=im2.size
```

```
newmat1=np.zeros((size[0]+258,size[1],3))

for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        a=im2.getpixel((i,j))
        hc=np.array([i,j,1])
        newhc=np.dot(rmat1,hc)
        if(newhc[0]>0 and newhc[1]<size[1]-1 and newhc[1]>0 and newhc[0]<size[0]+258-1):
            newmat1[np.int(newhc[0]),np.int(newhc[1])]=a
toimage(np.transpose(newmat1)).show()
toimage(np.transpose(newmat1)).save('affine1.png')

q1=np.zeros((600,1058,3))
im2=Image.open('Image1.jpg')
#im2.show()
size=im2.size

im4=Image.open('affine1.png')


for i in range(0,600):
    for j in range(0,1058):
        b=im4.getpixel((j,i))
        q1[i][j]=b
for i in range(0,600):
    for j in range(0,258):
        a=im2.getpixel((j,i))
        q1[i][j]=a

toimage(q1).show()
toimage(q1).save('Stitched1.png')
```

Results

(a) Image 1

(b) Image 2

Figure 2: Affine transformation on Image 2

Figure 3: Stitched image

Comments

Similarly, we can create a transformation matrix to align Image 1 with Image 2 and use this to stitch Image 2 and transformed Image1. To get properly stitched image, we should take start and end pixels of the two images appropriately. The edge present when stitched can be smoothened by averaging using a small window nearby the edge pixles.

2. Harris Corner detection

Use checkerboard.png image from resources folder.

Discussion

In computer vision, usually we need to find matching points between different frames of an environment. If we know how two images relate to each other, we can use both images to extract information of

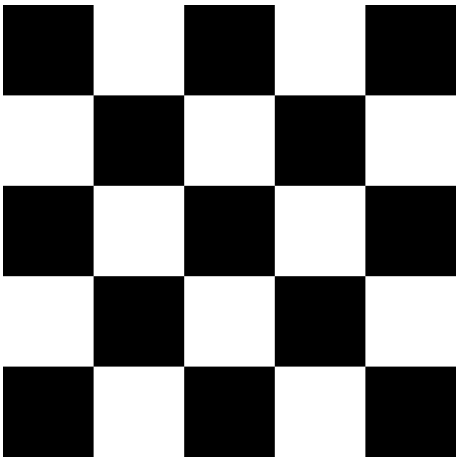
them. These matching points can be edges, corners or blobs. Corner is the intersection of two edges and represents a point in which the directions of these two edges change. Hence, the gradient of the image (in both directions) have a high variation, which can be used to detect it.

Program Code

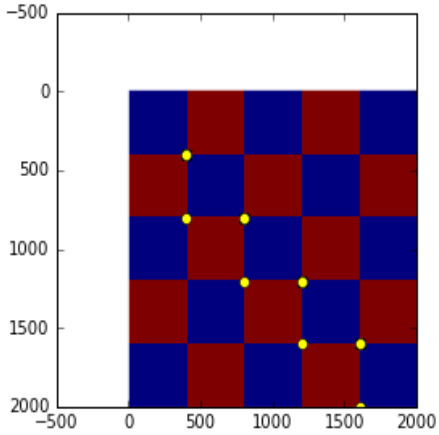
```
from PIL import Image
import numpy as np
from numpy.linalg import inv
from scipy.misc import toimage

im=Image.open('checkerboard.png')
size=im.size
s1=size[0]+3
s2=size[1]+3
a=im.convert('L')
Ix=np.zeros((s1,s2))
Iy=np.zeros((s1,s2))
Ixy=np.zeros((s1,s2))
for i in range(0,size[0]-2):
    for j in range(0,size[1]-2):
        a1=a.getpixel((i,j))
        a2=a.getpixel((i+1,j))
        a3=a.getpixel((i,j+1))
        a4=a.getpixel((i+1,j+1))
        Ix[i+3][j+3]=np.subtract(a2,a1)
        Iy[i+3][j+3]=np.subtract(a3,a1)
        #Ixy[i+3][j+3]=np.subtract(a4,a1)

a1=a.getpixel((i+1,j+1))
Ix[i+4][j+4]=-1*a1
Iy[i+4][j+4]=-1*a1
#Ixy[i+4][j+4]=-1*a1

toimage(Ix).save('Ix.png')
toimage(Iy).save('Iy.png')


Ixsq=np.dot(Ix,Ix)
Iysq=np.dot(Iy,Iy)
Ixy=np.dot(Ix,Iy)
toimage(Ixy).save('Ixy.png')
toimage(Ixsq).save('Ixsquare.png')
toimage(Iysq).save('Iysquare.png')

from scipy.signal import gaussian as gauss
import scipy.signal as ss
w=gauss(121,1)
w=w.reshape((11,11))
print('Gaussian window of std=1')
print(w)

Ax=ss.convolve2d(Ixsq,w)
Ay=ss.convolve2d(Iysq,w)
Axy=ss.convolve2d(Ixy,w)
corner=np.zeros((4000000,1))
p=q=0
k=0.05
```

```
for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        M=np.ndarray((2,2))
        M[0][0]=Ax[i+3][j+3]
        M[0][1]=M[1][0]=Axy[i+3][j+3]
        M[1][1]=Ay[i+3][j+3]
        det=(M[0][0]*M[1][1])-(M[0][1]*M[1][0])
        trace=M[0][0]+M[1][1]
        R=det - k*trace
        if(R>25000):
            print('Corner')
            print(R)
            corner[p]=i
            corner[p+1]=j
            p=p+1

import matplotlib.pyplot as plt
fig,ax=plt.subplots()
imm=ax.imshow(a)
for i in range(0,p):
    ax.plot([corner[i]],[corner[i+1]],'o',color='yellow')
plt.show()
```

Results



(a) Checkerboard image



(b) Corners detected using Harris Detection

Comments

Harris Corner Detection can be used for image alignment, stitching, object detection,etc.

Lab 4
Feature Descriptors

---

1. Implement Histogram of Oriented Gradients.

Theory

A feature descriptor helps to describe an image of size m x n using a single vector called the feature vector of some length,say N. In the case of HOG, the input image is of size 64 x 128 and the feature vector is of length 3780 x 1.
Steps:

1. Calculate the gradient images gx and gy using Sobel operator and the magnitude and direction of the gradient using the formulae:
$$g = \sqrt{g_x^2 + g_y^2}$$
$$\theta = arctan\frac{g_y}{g_x}$$
2. Calculate the histogram of gradients in 8 x 8 cells:
   - Divide the image into 8 x 8 cells
   - For each 8 x 8 cell, create a histogram with 9 bins for amgles from 0 to 180
   - A bin is selected based on the direction and the vote is selected based on the magnitude
3. A 16 x 16 block will contain 4 histograms ; concatenate them together to get a vector of length 36 x 1. Normalixe this vector using L2-norm.
4. Concatenate all the 16 x 16 block histogram vectors. The 16 x 16 blocks must be selected with 50% overlap. The final vector will be of length 3780 x 1.

Program Code:

```
from scipy.misc import imread,toimage
from PIL import Image
from scipy import ndimage
import numpy as np
import math

im=Image.open('a1.png')
size=im.size
im.show()
im = imread('a1.png');
toimage(im).show()
dx=ndimage.sobel(im,0)
dy=ndimage.sobel(im,1)
#toimage(dx).show()
```

```
#toimage(dy).show()

m = np.sqrt(np.square(dx) + np.square(dy))
orn = np.arctan2(dy,dx)
mag=np.zeros((size[1],size[0]))
orien=np.zeros((size[1],size[0]))

for i in range(0,size[1]-1):
    for j in range(0,size[0]-1):
        mag[i][j]= np.max(m[i][j])
        orien[i][j] = np.max(orn[i][j])

#-----------Step 2 -------------------------------------------------------
b=np.zeros((8,8))
o=np.zeros((8,8))
hist=np.zeros((128,9))
p=0
n=0
z=0
for l in range(0,63):
    k=0
    for k in range(0,127):

        for i in range(0,7):
            for j in range(0,7):
                b[i][j]=mag[i+k][j+l]
                o[i][j]=orien[i+k][j+l]
                o[i][j]=o[i][j] * 180 / np.pi
                if((o[i][j]>0 and o[i][j]<10) or (o[i][j]>170 and o[i][j]<180)):
                    hist[z][0] = hist[z][0] + b[i][j]
                elif(o[i][j]>10 and o[i][j]<30):
                    hist[z][1] = hist[z][1] + b[i][j]
                elif(o[i][j]>30 and o[i][j]<50):
                    hist[z][2] = hist[z][2] + b[i][j]
                elif(o[i][j]>50 and o[i][j]<70):
                    hist[z][3] = hist[z][3] + b[i][j]
                elif(o[i][j]>70 and o[i][j]<90):
                    hist[z][4] = hist[z][4] + b[i][j]
                elif(o[i][j]>90 and o[i][j]<110):
                    hist[z][5] = hist[z][5] + b[i][j]
                elif(o[i][j]>110 and o[i][j]<130):
                    hist[z][6] = hist[z][6] + b[i][j]
                elif(o[i][j]>130 and o[i][j]<150):
                    hist[z][7] = hist[z][7] + b[i][j]
                elif(o[i][j]>150 and o[i][j]<170):
                    hist[z][8] = hist[z][8] + b[i][j]
                elif(o[i][j] == 10):
                    hist[z][0] = hist[z][0] + (b[i][j]*(20-o[i][j])/20)
                    hist[z][1] = hist[z][1] + (b[i][j]*(o[i][j]-0)/20)
                elif(o[i][j] == 30):
                    hist[z][1] = hist[z][1] + (b[i][j]*(40-o[i][j])/20)
                    hist[z][2] = hist[z][2] + (b[i][j]*(o[i][j]-20)/20)
                elif(o[i][j] == 50):
                    hist[z][2] = hist[z][2] + (b[i][j]*(60-o[i][j])/20)
                    hist[z][3] = hist[z][3] + (b[i][j]*(o[i][j]-40)/20)
                elif(o[i][j] == 70):
                    hist[z][3] = hist[z][3] + (b[i][j]*(80-o[i][j])/20)
                    hist[z][4] = hist[z][4] + (b[i][j]*(o[i][j]-60)/20)
```

```
                elif(o[i][j] == 90):
                    hist[z][4] = hist[z][4] + (b[i][j]*(100-o[i][j])/20)
                    hist[z][5] = hist[z][5] + (b[i][j]*(o[i][j]-80)/20)
                elif(o[i][j] == 110):
                    hist[z][5] = hist[z][5] + (b[i][j]*(120-o[i][j])/20)
                    hist[z][6] = hist[z][6] + (b[i][j]*(o[i][j]-100)/20)
                elif(o[i][j] == 130):
                    hist[z][6] = hist[z][6] + (b[i][j]*(140-o[i][j])/20)
                    hist[z][7] = hist[z][7] + (b[i][j]*(o[i][j]-120)/20)
                elif(o[i][j] == 150):
                    hist[z][7] = hist[z][7] + (b[i][j]*(160-o[i][j])/20)
                    hist[z][8] = hist[z][8] + (b[i][j]*(o[i][j]-140)/20)
                elif(o[i][j] == 170):
                    hist[z][8] = hist[z][8] + (b[i][j]*(0-o[i][j])/20)
                    hist[z][0] = hist[z][0] + (b[i][j]*(o[i][j]-160)/20)
    k=k+8
    z=z+1
        #p=p+1
l=l+8
    #n=n+1
#--------------------Step 3-----------------------------------------
j=0
a=np.zeros((105,36))
while(j<105):
    h1=np.ndarray.tolist(hist[j])
    h2=np.ndarray.tolist(hist[j+1])
    h3=np.ndarray.tolist(hist[j+8])
    h4=np.ndarray.tolist(hist[j+9])
    x=0
    x=np.concatenate((h1,h2),0)
    x=np.append(x,h3)
    x=np.append(x,h4)
    x_norm =np.linalg.norm(x)
    if(x_norm>0):
        a[j]=x/x_norm
    else:
         a[j]=x
    j=j+1
feature_giant = np.reshape(a,(105*36 , 1))  #HOG feature descriptor for the image
#-------------Step 4-------Visualization--------------------
im=Image.open('a1.png').convert('L')
import matplotlib.pyplot as plt
x,y=np.meshgrid(np.linspace(0,133,8),np.linspace(0,69,8))
fig, ax = plt.subplots()
ax.imshow(im)
u=[]
v=[]
for i in range(0,105):
    t1 = np.argmax(a[i])
    t2 = np.amax(a[i])
    u.append(t2*np.sin(t1*20))
    v.append(t2*np.cos(t1*20))
ax.quiver(y,x,v,u,color='red',scale=1)
plt.show()
```

Results

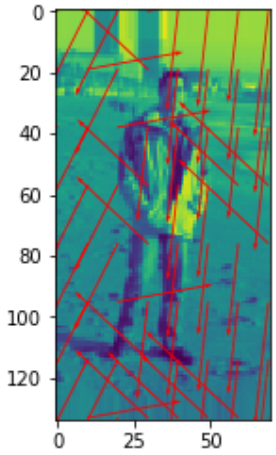


Figure 1: Hog visualization on image

Lab 5
Optical Flow Estimation

---

1. Lucas Kanade method of Optical Flow Estimation

Write a program to find the Optical flow between the given images using Lucas Kanade method.

Theory

Optical flow describes a sparse or dense vector field, where a displacement vector is assigned to certain pixel position, that points to where that pixel can be found in another image.The Lucas Kanade method is a widely used differential method for optical flow estimation. This method solves the basic optical flow equations for all the pixels in that neighbourhood, by the least squares criterion. It is a purely local method because it cannot provide flow information in the interior of uniform regions of the image. It assumes that the flow is essentially constant in a local neighbourhood of the pixel under consideration.
Steps:

1. Compute the gradients I_x and I_y of the first image.
2. Compute I_t as image1 - image2
3. Compute I_x^2 ,I_y^2,I_xI_y, I_xI_t,I_tI_y
4. For each point in the above computed matrices, find the sum of pixels in a neighbourhood
5. Compute optical flow vector [u,v] for each pixel :

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x^2 & \sum I_xI_y \\ \sum I_yI_x & \sum I_y^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum I_xI_t \\ -\sum I_yI_t \end{bmatrix} \tag{1}$$

6. Plot the optical flow vectors on the first image

Program Code:

```
from PIL import Image,ImageDraw
im=Image.open('OF1.png').convert('L')

#im.show()
im.save('OF11.png')
size=im.size
im=Image.open('OF2.png').convert('L')
#im.show()
im.save('OF22.png')
size=im.size

from scipy.ndimage import sobel
from scipy.misc import toimage,imread
im1=imread('OF11.png')
```

```
im2=imread('OF22.png')
ix=sobel(im1,0)
iy=sobel(im1,1)
it=im1-im2
toimage(ix).save('Ix.png')
toimage(iy).save('Iy.png')
toimage(it).save('It.png')
import numpy as np
ix_sq=np.square(ix)
iy_sq=np.square(iy)
ix_iy=np.multiply(ix,iy)
ix_it=np.multiply(ix,it)
iy_it=np.multiply(iy,it)
u=np.zeros((size[1],size[0]))
v=np.zeros((size[1],size[0]))
toimage(ix_sq).save('Ixsquare.png')
toimage(iy_sq).save('Iysquare.png')
toimage(ix_iy).save('IxIy.png')
toimage(ix_it).save('IxIt.png')
toimage(iy_it).save('IyIt.png')
im=Image.open('OF11.png')
for i in range(2,size[1]-3):
    for j in range(2,size[0]-3):
        int1=0
        int2=0
        int3=0
        int4=0
        int5=0
        for m in range(-2,2):
            for n in range(-2,2):

                int1=int1+ix_sq[i+m][j+n]
                int2=int2+iy_sq[i+m][j+n]
                int3=int3+ix_iy[i+m][j+n]
                int4=int4+ix_it[i+m][j+n]
                int5=int5+iy_it[i+m][j+n]

        mat1=np.array([[int1,int3],[int3,int2]])

        int4=(-1)*int4
        int5=(-1)*int5
        mat2=np.array([int4,int5])
        if(np.linalg.det(mat1)!=0):
            mat1=np.linalg.inv(mat1)
        else:
            mat1=np.identity(2)

        u1=np.dot(mat1,mat2)
        u[i][j]=u1[0]
        v[i][j]=u1[1]

import matplotlib.pyplot as plt
x,y=np.meshgrid(np.linspace(0,719,10),np.linspace(0,1279,20))
fig, ax = plt.subplots()
ax.imshow(im)
#plt.hold
ax.quiver(y,x,u/np.max(u),v/np.max(v),color='red',scale=10)
plt.show()
```

Results

(a) Image 1

(b) Image 2

Figure 1: Given images

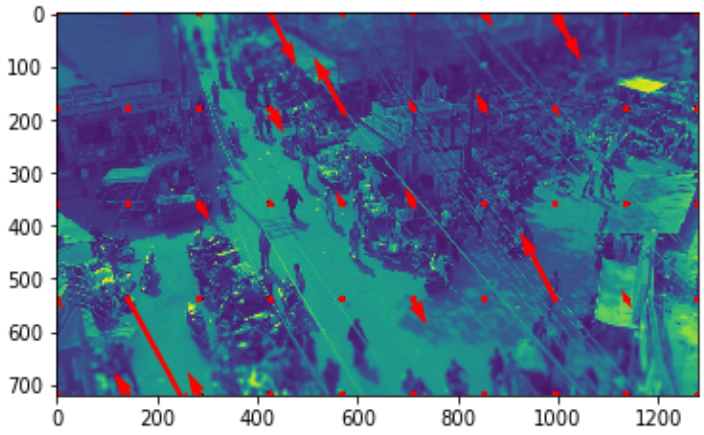


Figure 2: Optical flow in the given images

2. Horn and Schunk method of Optical Flow estimation

Write a program to find the Optical flow between the given images using Horn and Schunk method.

Theory

The Horn-Schunck algorithm assumes smoothness in the flow over the whole image. Thus, it tries to minimize distortions in flow and prefers solutions which show more smoothness. Advantages of the HornSchunck algorithm include that it yields a high density of flow vectors, i.e. the flow information missing in inner parts of homogeneous objects is filled in from the motion boundaries. On the negative side, it is more sensitive to noise than local methods.

Steps:

1. Initialize u and v as zero matrices of size equal to the input image 1

2. Initialize the convolution matrices:

$$convmat1 = \begin{bmatrix} -0.25 & 0.25 \\ -0.25 & 0.25 \end{bmatrix}$$

$$convmat2 = \begin{bmatrix} -0.25 & -0.25 \\ 0.25 & 0.25 \end{bmatrix}$$

$$convmat3 = \begin{bmatrix} 0.25 & 0.25 \\ 0.25 & 0.25 \end{bmatrix}$$

$$convmat4 = \begin{bmatrix} -0.25 & -0.25 \\ -0.25 & -0.25 \end{bmatrix}$$

3. Compute I_x,I_y and I_t as:
   $I_x = conv(image1, convmat1) + conv(image2, convmat1)$
   $I_y = conv(image1, convmat2) + conv(image2, convmat2)$
   $I_t = conv(image1, convmat3) + conv(image2, convmat4)$

4. Initialize kernel as

$$\begin{bmatrix} 1/12 & 1/6 & 1/12 \\ 1/6 & 0 & 1/6 \\ 1/12 & 1/6 & 1/12 \end{bmatrix}$$

5. for iter = 1:100

   - $u_{avg} = conv(u, kernel)$
   - $v_{avg} = conv(v, kernel)$
   - $u = u_{avg} - \frac{I_x(uI_x+vI_y+I_t)}{\alpha^2+I_x^2+I_y^2}$
   - $v = v_{avg} - \frac{I_y(uI_x+vI_y+I_t)}{\alpha^2+I_x^2+I_y^2}$

6. Plot the optical flow vectors on the first image

Program Code:

```
from scipy.ndimage import sobel
from scipy.misc import toimage,imread
im1=imread('OF11.png')
im2=imread('OF22.png')
from PIL import Image
im=Image.open('OF1.png').convert('L')
#im.show()
im.save('OF11.png')
size=im.size
conv_mat1=np.array([[-0.25,0.25],[-0.25,0.25]])
conv_mat2=np.array([[-0.25,-0.25],[0.25,0.25]])
conv_mat3=np.array([[0.25,0.25],[0.25,0.25]])
conv_mat4=np.array([[-0.25,-0.25],[-0.25,-0.25]])

from scipy.signal import convolve2d as cc
conv10=cc(im1,conv_mat1,mode='same')
conv20=cc(im1,conv_mat2,mode='same')
conv30=cc(im1,conv_mat3,mode='same')

conv11=cc(im2,conv_mat1,mode='same')
conv21=cc(im2,conv_mat2,mode='same')
conv41=cc(im2,conv_mat4,mode='same')
```

```
Ix=conv10+conv11
Iy=conv20+conv21
It=conv30+conv41

kernel=np.array([[1/12 ,1/6,1/12],[1/6,0,1/6],[1/12,1/6,1/12]])

v=np.zeros((size[1],size[0]))
u=np.zeros((size[1],size[0]))

for i in range(0,100):
    uavg=cc(u,kernel,mode='same')
    vavg=cc(v,kernel,mode='same')

    z1=np.multiply(Ix,u) + np.multiply(Iy,v) + It
    z2=1 + np.square(Ix) +np.square(Iy)
    z3=np.multiply(Ix,z1)
    z4=np.multiply(Iy,z1)
    u=uavg-(z3/z2)
    v=vavg-(z4/z2)
import matplotlib.pyplot as plt
x,y=np.meshgrid(np.linspace(0,719,5),np.linspace(0,1279,10))
fig, ax = plt.subplots()
ax.imshow(im)
#plt.hold
ax.quiver(y,x,u/np.max(u),v/np.max(v),color='red',scale=10)
plt.show()
```

Results

(a) Image 1

(b) Image 2

Figure 3: Given images

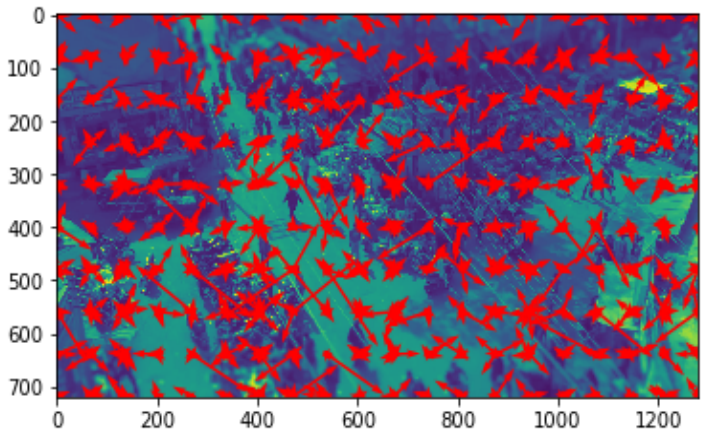


Figure 4: Optical flow in the given images

Lab 6
KLT Tracker

---

1. KLT Tracker

Write a program to track the car in the video Track.mp4 , using the KLT method. You can show the track of any one Harris corner in few consecutive frames of the video.

Theory

KLT tracker makes use of spatial intensity information to direct the search for the position that yields the best match.
Steps:

1. Compute Harris corners of the current frame
2. Select any one Harris corner of your choice
3. Compute the transformation of that point compared to the next frame using KLT method
4. Using the computed transformation,determine the location of that point in the new frame
5. Repeat the above steps for any number of frames of your choice

Program Code:

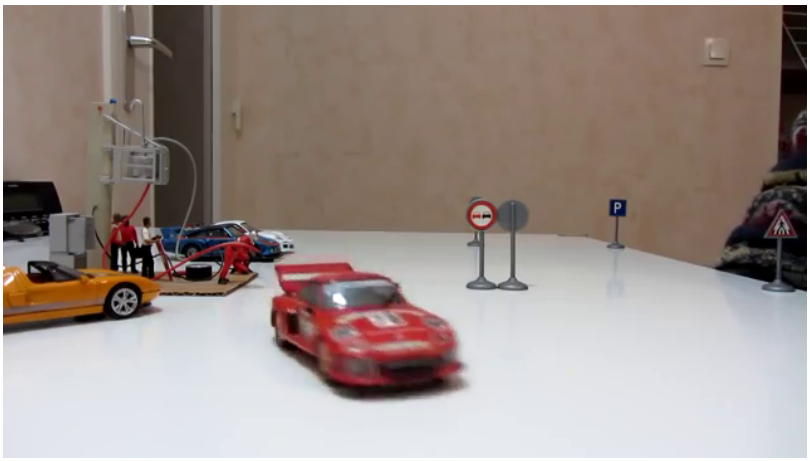
```
from PIL import Image
import numpy as np
from numpy.linalg import inv
from scipy.misc import toimage,imread

im=Image.open('D:/To desktop/Mtech/sem2/cvlab/lab6/frames/251.png')
size=im.size
s1=size[0]+3
s2=size[1]+3
a=im.convert('L')
Ix=np.zeros((s1,s2))
Iy=np.zeros((s1,s2))
Ixy=np.zeros((s1,s2))
for i in range(0,size[0]-2):
    for j in range(0,size[1]-2):
        a1=a.getpixel((i,j))
        a2=a.getpixel((i+1,j))
        a3=a.getpixel((i,j+1))
        a4=a.getpixel((i+1,j+1))
        Ix[i+3][j+3]=np.subtract(a2,a1)
        Iy[i+3][j+3]=np.subtract(a3,a1)
        #Ixy[i+3][j+3]=np.subtract(a4,a1)
```

```
a1=a.getpixel((i+1,j+1))
Ix[i+4][j+4]=-1*a1
Iy[i+4][j+4]=-1*a1
#Ixy[i+4][j+4]=-1*a1

toimage(Ix).save('Ix.png')
toimage(Iy).save('Iy.png')


Ixsq=np.multiply(Ix,Ix)
Iysq=np.multiply(Iy,Iy)
Ixy=np.multiply(Ix,Iy)
toimage(Ixy).save('Ixy.png')
toimage(Ixsq).save('Ixsquare.png')
toimage(Iysq).save('Iysquare.png')

from scipy.signal import gaussian as gauss
import scipy.signal as ss
w=gauss(121,1)
w=w.reshape((11,11))
print('Gaussian window of std=1')
print(w)

Ax=ss.convolve2d(Ixsq,w)
Ay=ss.convolve2d(Iysq,w)
Axy=ss.convolve2d(Ixy,w)
corner=np.zeros((4000000,1))
p=q=0
k=0.05

for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        M=np.ndarray((2,2))
        M[0][0]=Ax[i+3][j+3]
        M[0][1]=M[1][0]=Axy[i+3][j+3]
        M[1][1]=Ay[i+3][j+3]
        det=(M[0][0]*M[1][1])-(M[0][1]*M[1][0])
        trace=M[0][0]+M[1][1]
        R=det - k*trace
        if(R>50000000):
            print('Corner')
            print(R)
            corner[p]=i
            corner[p+1]=j
            p=p+1

import matplotlib.pyplot as plt
fig,ax=plt.subplots()
imm=ax.imshow(a)
for i in range(0,p):
    if(210<corner[i]<250 and 210<corner[i+1]<250):
        print(corner[i])
        print(corner[i+1])
        point = np.array([corner[i],corner[i+1],1])
        ax.plot([corner[i]],[corner[i+1]],'o',color='red')
plt.show()
#----Affine tfmn ------------------------------------------------------------
```

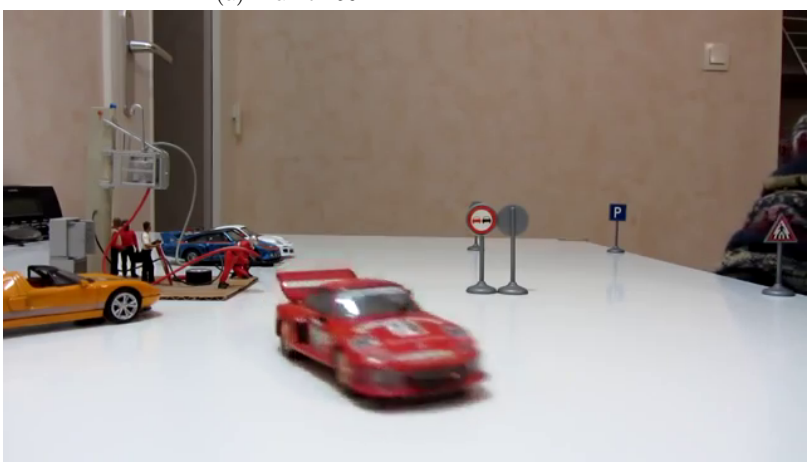
```
p1=np.array([230,230])
p2=np.array([230,231])
p3=np.array([300,300])

P1=np.array([232,232])
P2=np.array([232,233])
P3=np.array([302,302])

T=np.array([[p1[0],p1[1],1,0,0,0],
            [0,0,0,p1[0],p1[1],1],
            [p2[0],p2[1],1,0,0,0],
            [0,0,0,p2[0],p2[1],1],
            [p3[0],p3[1],1,0,0,0],
            [0,0,0,p3[0],p3[1],1]])
p=np.array([P1[0],P1[1],P2[0],P2[1],P3[0],P3[1]])

Tinv=inv(T)
res=np.dot(Tinv,p)

rmat=np.reshape(res,(2,3))

im=Image.open('D:/To desktop/Mtech/sem2/cvlab/lab6/frames/250.png')
size=im.size

newmat=np.zeros((size[0],size[1],3))

for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        a=im.getpixel((i,j))
        hc=np.array([i,j,1])
        newhc=np.dot(rmat,hc)
        if(newhc[0]>0 and newhc[1]<size[1]-1 and newhc[1]>0 and newhc[0]<size[0]-1):
            newmat[np.int(newhc[0])][np.int(newhc[1])]=a

#toimage(np.transpose(newmat)).show()
toimage(np.transpose(newmat)).save('affine.png')
#-------------------------------------------------------------------
T=imread('D:/To desktop/Mtech/sem2/cvlab/lab6/frames/251.png')
I=imread('D:/To desktop/Mtech/sem2/cvlab/lab6/affine.png')
subt = T-I
size=T.shape
Igradx = np.zeros(size)
Igrady = np.zeros(size)
for i in range(0,size[0]-1):
    for j in range(0,size[1]-2):
        Igradx[i][j] = I[i][j+1]-I[i][j]
for j in range(0,size[1]-1):
    for i in range(0,size[0]-2):
        Igrady[i][j] = I[i+1][j]-I[i][j]
H=np.zeros((6,6))
for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        h=np.zeros((6,6))
        Ix=Igradx[i][j]
        Iy=Igrady[i][j]
        w=np.array([Ix,Iy,i*Ix,j*Ix,i*Iy,j*Iy])
        h=np.dot(w,np.transpose(w))
        H=H+h
```

```
sum1=np.zeros((1,6))
dif=np.zeros((3,1))
for i in range(0,size[0]-1):
    for j in range(0,size[1]-1):
        w=np.zeros((6,3))
        prod1=np.zeros((6,1))
        Ix=Igradx[i][j]
        Iy=Igrady[i][j]
        w=np.array([Ix,Iy,i*Ix,j*Ix,i*Iy,j*Iy])
        dif=T[i][j]-I[i][j]
        prod1=np.dot(w,dif)
        sum1=sum1+prod1
del_p = np.dot(H,np.transpose(sum1))
p=res
new_p = p + np.transpose(del_p)
new_p = np.reshape(new_p,(2,3))
c = np.dot(new_p,point)
im=Image.open('D:/To desktop/Mtech/sem2/cvlab/lab6/frames/251.png').convert('L')
fig, ax = plt.subplots()
ax.imshow(im)
ax.plot([c[0][0]],[c[1][0]],'o',color='red')
plt.show()
```

Results



(a) Frame 250



(b) Frame 251

Figure 1: Frames

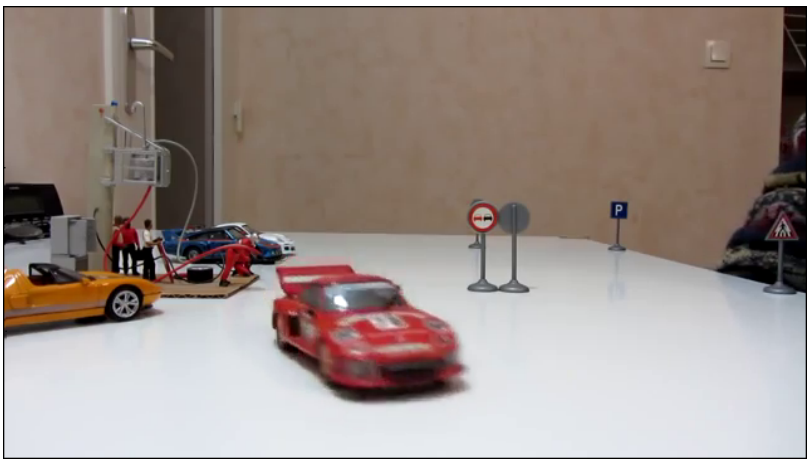
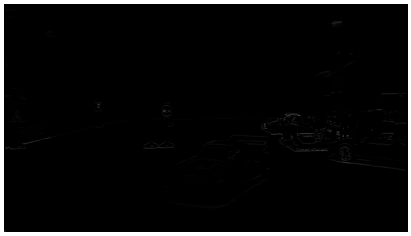


Figure 2: Affine transformation of frame 250

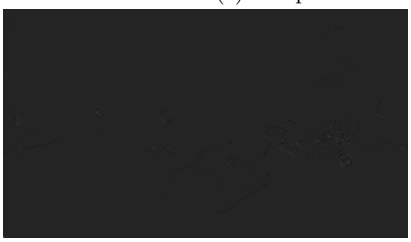
(a) Ix

(b) Iy

(c) Ix squared

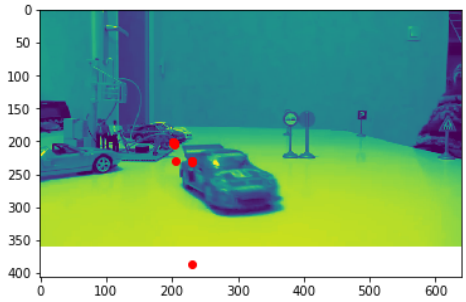


(d) Iy squared

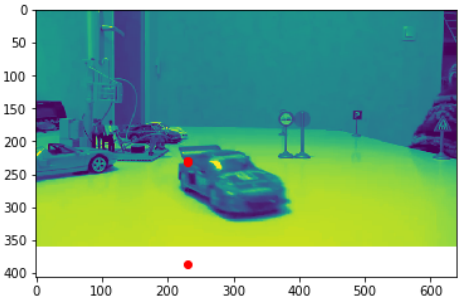


(e) IxIy

Figure 3: Gradients for Hessian matrix



(a) Harris point in frame 250



(b) Harris point in frame 251

Figure 4: KLT tracking