Digital Imaging Systems

Project 1

Project Description

This project focuses on an implementation of the paper “Single View Metrology” (Criminisi, Reid and Zisserman, ICCV99).

Implementation

1. **Image Acquisition**
   An image is acquired in 3 point perspective.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('box.png')
```

Fig 1: Original Image

2. Computing Vanishing Points

2.1 Specifying the end points of the parallel lines

Pixel co-ordintes of the object in the image are determined and passed to the program. Then the parallel lines are defined using the given corner points.

```
#Defining the corners the object in the image
def click_event(event, x, y, flags, params):
    if event == cv2.EVENT_LBUTTONDOWN:
        temp = [x,y,1]
        points.append(temp)

    return points
cv2.namedWindow("image", cv2.WINDOW_NORMAL)
cv2.resizeWindow("image", 1280, 720)
cv2.imshow('image', img)
points=[]
cv2.setMouseCallback('image', click_event)
cv2.waitKey(0)
cv2.destroyAllWindows()
print(points)

p1,p2,p3,p4,p5,p6,p7 = points
```

```
#Defining the end points of the parallel lines
x1_1 = p1
x1_2 = p2

x2_1 = p3
x2_2 = p4

x3_1 = p5
x3_2 = p6

y1_1 = p1
y1_2 = p7

y2_1 = p3
y2_2 = p5

y3_1 = p4
y3_2 = p6

z1_1 = p1
z1_2 = p3

z2_1 = p2
z2_2 = p4

z3_1 = p7
z3_2 = p5
```

```
# Annotation
img1 = cv2.imread('box.png')
#X
cv2.line(img1,(x1_1[0],x1_1[1]),(x1_2[0],x1_2[1]),(0,0,255),2)
cv2.line(img1,(x2_1[0],x2_1[1]),(x2_2[0],x2_2[1]),(0,0,255),2)
cv2.line(img1,(x3_1[0],x3_1[1]),(x3_2[0],x3_2[1]),(0,0,255),2)

#Y
cv2.line(img1,(y1_1[0],y1_1[1]),(y1_2[0],y1_2[1]),(0,255,0),2)
cv2.line(img1,(y2_1[0],y2_1[1]),(y2_2[0],y2_2[1]),(0,255,0),2)
cv2.line(img1,(y3_1[0],y3_1[1]),(y3_2[0],y3_2[1]),(0,255,0),2)


#Z
cv2.line(img1,(z1_1[0],z1_1[1]),(z1_2[0],z1_2[1]),(250,0,0),2)
cv2.line(img1,(z2_1[0],z2_1[1]),(z2_2[0],z2_2[1]),(250,0,0),2)
cv2.line(img1,(z3_1[0],z3_1[1]),(z3_2[0],z3_2[1]),(255,0,0),2)

cv2.imwrite("Annotated.png",img1)
```

Fig 2: Annotated Image with parallel lines defined
Red, green and blue lines indicate the assumed x, y and z axis respectively

2.2 Calculating the vanishing points

The vanishing points are calculated using the parallel lines defined earlier following the Bob Collins method

```
#Calculating the vanishing points
def cal_vanishing(i,j,k,l):
    a1,b1,c1 = np.cross(i,j)
    a2,b2,c2 = np.cross(k,l)
    V = np.cross([a1,b1,c1],[a2,b2,c2])
    V = V/V[2]
    return np.array(V)


Vx = cal_vanishing(x1_1,x1_2,x2_1,x2_2)
Vy = cal_vanishing(y1_1,y1_2,y2_1,y2_2)
Vz = cal_vanishing(z1_1,z1_2,z2_1,z2_2)
```

3. Computing the projection and homograph matrix

The projection matrix is calculated by using the vanishing points calculated earlier and scaling them appropriately. Later the homograph matrix is defined using the projection matrix.

3.1 Defining the reference points

```
#Define world origin and X,Y and Z reference points
wo = np.array(p1)
ref_x = np.array([p2])
ref_y = np.array([p7])
ref_z = np.array([p3])
```

3.2 Calculating the distance between the references and origin

```
# Calculate distance between world origin and reference co-ordinates
len_x = np.sqrt(np.sum(np.square(ref_x - wo)))
len_y = np.sqrt(np.sum(np.square(ref_y - wo)))
len_z = np.sqrt(np.sum(np.square(ref_z - wo)))
```

3.3 Calculating the scaling factors

```
#Calculate Scaling Factors
ax = np.linalg.lstsq( (Vx-ref_x).T , (ref_x - wo).T )[0][0][0]/len_x
ay = np.linalg.lstsq( (Vy-ref_y).T , (ref_y - wo).T )[0][0][0]/len_y
az = np.linalg.lstsq( (Vz-ref_z).T , (ref_z - wo).T )[0][0][0]/len_z
```

3.4 Creating the projection matrix

```
#Creating the projection matrix
px = ax*Vx
py = ay*Vy
pz = az*Vz

P = np.empty([3,4])
P[:,0] = px
P[:,1] = py
P[:,2] = pz
P[:,3] = wo
```

3.5 Creating the homograph matrices

```
#Homograph Matrices
Hxy = np.zeros((3,3))
Hyz = np.zeros((3,3))
Hzx = np.zeros((3,3))

Hxy[:,0] = px
Hxy[:,1] = py
Hxy[:,2] = wo

Hyz[:,0] = py
Hyz[:,1] = pz
Hyz[:,2] = wo

Hzx[:,0] = px
Hzx[:,1] = pz
Hzx[:,2] = wo
```

4. Computing the texture maps

4.1 Transforming the original image using the homograph matrices

The homographies calculated are used to alter images and generate texture maps for each plane.

```
#Computing the texture Maps
r,c,temp = img.shape
Txy = cv2.warpPerspective(img,Hxy,(r,c),flags=cv2.WARP_INVERSE_MAP)
Tyz = cv2.warpPerspective(img,Hyz,(r,c),flags=cv2.WARP_INVERSE_MAP)
Tzx = cv2.warpPerspective(img,Hzx,(r,c),flags=cv2.WARP_INVERSE_MAP)

cv2.imwrite("XY.png",Txy)
cv2.imwrite("YZ.png",Tyz)
cv2.imwrite("ZX.png",Tzx)
```

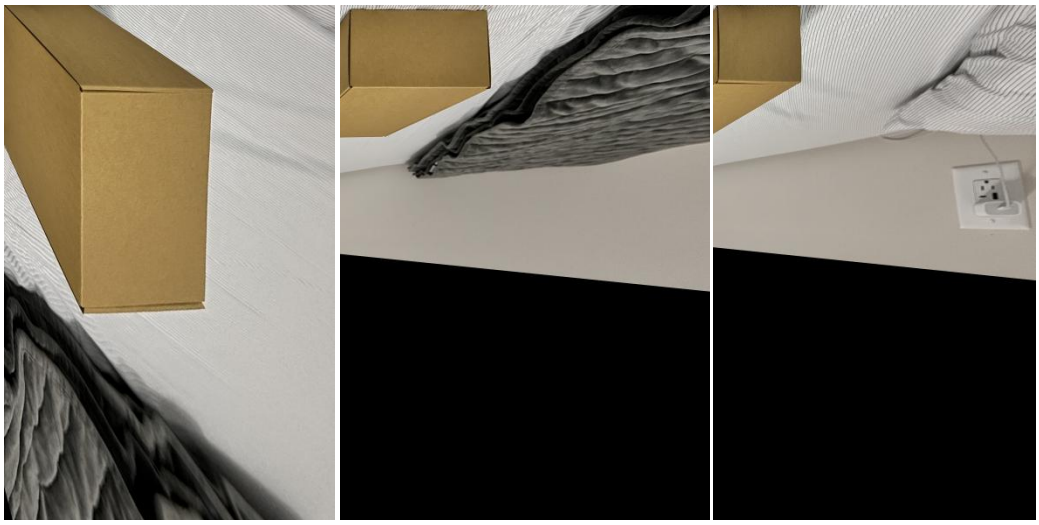

Fig 4: XY, YZ, XZ plane texture maps

4.2 Cropping the texture maps

The region of interest is cropped out of the texture maps to later construct the 3D model**.**

Fig 5: Cropped portions of the XY, YZ and ZX texture maps

5. Visualising the reconstructed 3D model

The 3D model is constructed in Blender.

Fig 6: Reconstructed 3D model of the object

References

- Kuse Manohar, 3D Reconstruction with Single View: https://kusemanohar.wordpress.com/2014/03/18/3d-reconstruction-with-single-view/
- Desmond Tsoi Yau Chat and Stanley Ng Ho Lun, Single View Metrology: https://kusemanohar.files.wordpress.com/2014/03/singleviewmetrologyfinalreport.pdf
- https://courses.cs.washington.edu/courses/cse455/02wi/projects/project3/project3_old.htm