ECE558 Project 01 – Option 01

SINGLE VIEW METROLOGY

Ashwini Muralidharan

Objective : Given a 2-D image, 3D affine measurements are computed from a single perspective view of a scene using minimal geometric information of the 2-D image, as applied by Criminisi, Reid and Zisserman in “Single View Metrology”.

The project is divided into 4 subtasks :

1. Image acquisition :
   A photo of a box following the 3-point perspective image guide is clicked.

FIG. : Original image

Several parallel lines along each axis in the image are manually marked and annotated as shown below.

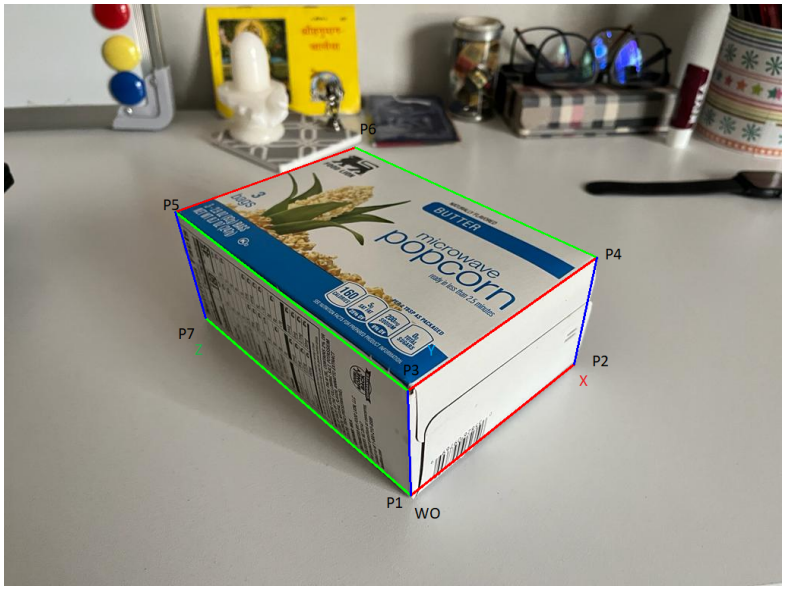

FIG. : Annotated image

Lines that are parallel in the real world may not necessarily be parallel in the image plane. They intersect at some point in the 2D image, thus giving rise to a Vanishing Point.

2. Computing Vanishing Points :
   Vanishing points are computed at the intersection point of the parallel lines. Two lines from each plane are taken and cross product between them yields the point of intersection i.e., the vanishing point.

   Vanishing point along x-axis, $V_x = [ax1, bx1, cx1] \times [ax2, bx2, cx2]$
   Vanishing point along y-axis, $V_y = [ay1, by1, cy1] \times [ay2, by2, cy2]$
   Vanishing point along z-axis, $V_z = [az1, bz1, cz1] \times [az2, bz2, cz2]$
   (where, ax_i, ay_i, az_i are the coefficients of the line passing through 2 points from the image)

   The result is converted to the homogeneous coordinate system by dividing the first two positions by the third.

3. Computing Projection Matrix and Homograph Matrix :
   The concatenation of the Vanishing Points column-wise gives the Projection Matrix, where the columns are up to a scaling constant. The scaling constants are evaluated wrt reference points using the formulae below.

   $$a_x = ([V_X - wo]^{-1} * (ref_x - wo)) / ref_x distance$$

$a_y = ([V_y - wo]^{-1} * (ref_y - wo)) / ref_y distance$

$a_z = ([V_z - wo]^{-1} * (ref_z - wo)) / ref_z distance$

where a_x, a_y, a_z are the scaling constants,

wo is this world origin,

ref_x, ref_y, ref_z are reference points,

$ref_x distance$, $ref_y distance$, $ref_z distance$ are the distances of the reference points from world origin.

Projection matrix,

$P = [aV_x \; bV_y \; cV_z \; wo]$

The Homography Matrix (H) corresponding to each of the normal planes (ie. XY, YZ, XZ) is calculated. For Hxy, take the 1st, 2nd and 4th columns of the projection matrix (P), for Hyz the 2nd, 3rd, 4th columns and for Hzx, the 3rd, 1st and 4th columns are taken. Using reverse warpings on these homography matrices, the image is transformed and texture maps are created.

(a) Hxy

(a) Hyz

(c) Hzx

FIG. : Perspective transformed images along each plane

4. Visualizing the reconstructed 3D model :

The faces of the box from the transformed images are cropped.

FIG.: Cropped faces from the transformed images

The images were then imported into Blender, a 3D visualization software to render the model. Illustrated below is the rendered 3D model to reconstruct the box back in 3D.

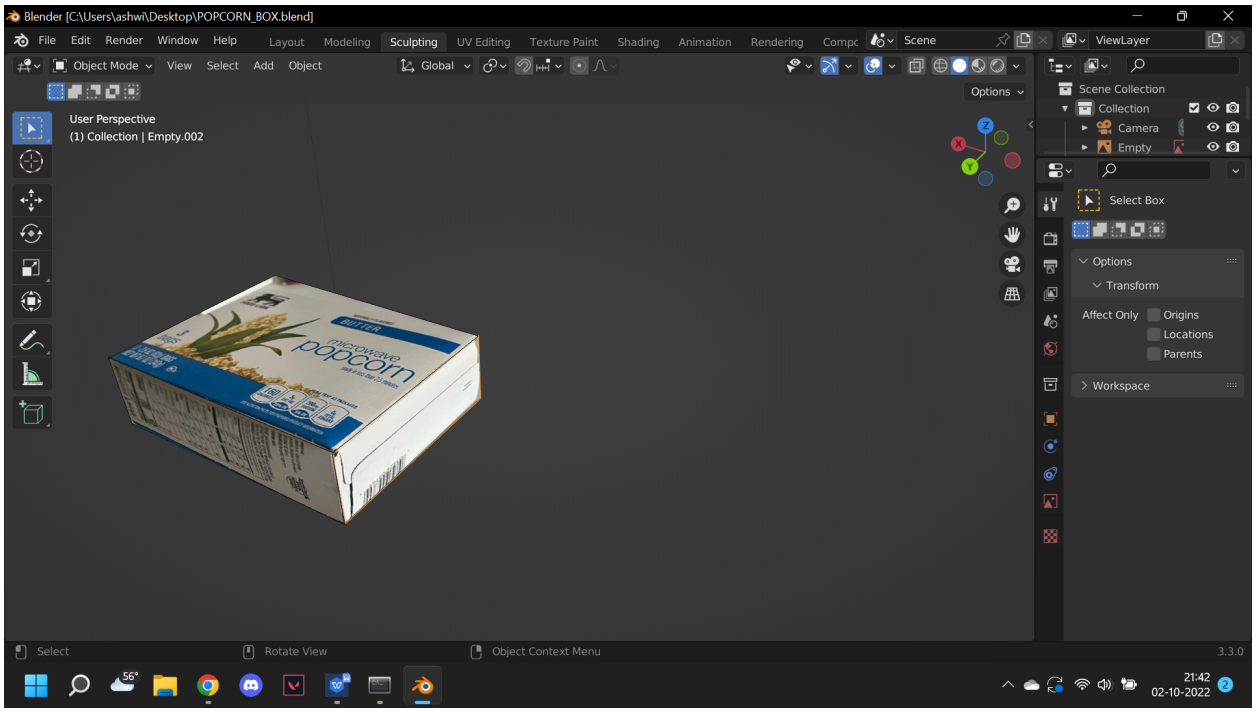

FIG.: Snapshot of reconstructed image from Blender

PYTHON CODE :

- Annotation interface :

```
import cv2
import pandas as pd
import numpy as np

def draw_lines(event, x, y, flag, parameters):
    global index, line_coord
    if event == cv2.EVENT_LBUTTONDOWN:
        line_coord = [(x,y)]
    elif event == cv2.EVENT_LBUTTONUP:
        line_coord.append((x,y))
        print(f"Line starting: {line_coord[0]} line ending: {line_coord[1]}")

        cv2.line(img, line_coord[0], line_coord[1],color[index],4)
        cv2.imshow("image", img)
    elif event == cv2.EVENT_RBUTTONDOWN:
        index  = (index + 1) % 3

color = [(0,0,255),(0,255,0),(255,0,0)]
line_coord = []
index = 0

img = cv2.imread("popcorn_box.jpg")
cv2.imshow('image', img)

cv2.setMouseCallback('image', draw_lines)

cv2.waitKey(0)
cv2.imwrite("annotated.png",img)
cv2.destroyAllWindows()

P1 = [535,648, 1]
P2 = [750, 475, 1]
P3 = [531, 510, 1]
P4 = [780, 334, 1]
P5 = [225, 274, 1]
P6 = [462, 189, 1]
P7 = [265, 414, 1]

x1_p1 = P1
x1_p2 = P2
```

```
y1_p1 = P1
y1_p2 = P7

z1_p1 = P1
z1_p2 = P3

x2_p1 = P3
x2_p2 = P4

y2_p1 = P3
y2_p2 = P5

z2_p1 = P2
z2_p2 = P4

x3_p1 = P5
x3_p2 = P6

y3_p1 = P4
y3_p2 = P6

z3_p1 = P7
z3_p2 = P5

cv2.line(image,(x1_p1[0],x1_p1[1]),(x1_p2[0],x1_p2[1]),(0,0,255),2)
cv2.line(image,(x2_p1[0],x2_p1[1]),(x2_p2[0],x2_p2[1]),(0,0,255),2)
cv2.line(image,(x3_p1[0],x3_p1[1]),(x3_p2[0],x3_p2[1]),(0,0,255),2)

cv2.line(image,(y1_p1[0],y1_p1[1]),(y1_p2[0],y1_p2[1]),(0,255,0),2)
cv2.line(image,(y2_p1[0],y2_p1[1]),(y2_p2[0],y2_p2[1]),(0,255,0),2)
cv2.line(image,(y3_p1[0],y3_p1[1]),(y3_p2[0],y3_p2[1]),(0,255,0),2)

cv2.line(image,(z1_p1[0],z1_p1[1]),(z1_p2[0],z1_p2[1]),(250,0,0),2)
cv2.line(image,(z2_p1[0],z2_p1[1]),(z2_p2[0],z2_p2[1]),(250,0,0),2)
cv2.line(image,(z3_p1[0],z3_p1[1]),(z3_p2[0],z3_p2[1]),(255,0,0),2)

cv2.imshow('image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()

cv2.imwrite('annotated_image.png', image)
```


```
coordinates = np.zeros((7,3))
coordinates[0,:] = P1
coordinates[1,:] = P2
coordinates[2,:] = P3
coordinates[3,:] = P4
coordinates[4,:] = P5
coordinates[5,:] = P6
coordinates[6,:] = P7

coordinates = np.array(coordinates)

df = pd.DataFrame({"X" : coordinates[:,0], "Y" : coordinates[:,1], "Z" : coordinates[:,2]})
df.to_csv("coordinates.csv", index=False)
```

- Single View Metrology

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

image = cv2.imread('popcorn_box.jpg')

df = pd.read_csv('coordinates.csv');
coordinates = np.array(df)

P1 = coordinates[0]
P2 = coordinates[1]
```

```
P3 = coordinates[2]
P4 = coordinates[3]
P5 = coordinates[4]
P6 = coordinates[5]
P7 = coordinates[6]

x1_p1 = P1
x1_p2 = P2

y1_p1 = P1
y1_p2 = P7

z1_p1 = P1
z1_p2 = P3

x2_p1 = P3
x2_p2 = P4

y2_p1 = P3
y2_p2 = P5

z2_p1 = P2
z2_p2 = P4

x3_p1 = P5
x3_p2 = P6

y3_p1 = P4
y3_p2 = P6

z3_p1 = P7
z3_p2 = P5

# Compute Vanishing Points

ax1,bx1,cx1 = np.cross(x1_p1,x1_p2) #obtain co-efficients of line equation of line X1
ax2,bx2,cx2 = np.cross(x2_p1,x2_p2) #obtain co-efficients of line equation of line X2
Vx = np.cross([ax1,bx1,cx1],[ax2,bx2,cx2]) #taking cross product of line X1 and X2 gives the point at which the 2 lines meet i.e., the vanishing point

ay1,by1,cy1 = np.cross(y1_p1,y1_p2) #line Y1
ay2,by2,cy2 = np.cross(y2_p1,y2_p2) #line Y2
Vy = np.cross([ay1,by1,cy1],[ay2,by2,cy2]) #intersection of Y1 and Y2 gives vanishing point
```

```
az1,bz1,cz1 = np.cross(z1_p1,z1_p2) #line Z1
az2,bz2,cz2 = np.cross(z2_p1,z2_p2) #line Z2
Vz = np.cross([az1,bz1,cz1],[az2,bz2,cz2]) #intersection of Z1 and Z2 gives vanishing point

print("Hence the Vanishing points for this image are :")
print("Along x-axis : ", Vx)
print("Along y-axis : ", Vy)
print("Along z-axis : ", Vz)
```

```
Hence the Vanishing points for this image are :
Along x-axis :  [10340985 -2672842     5237]
Along y-axis :  [6928740 4551624   -7884]
Along z-axis :  [2617140 6515442    4704]
```

```
# To obtain VP in homogeneous coordinate system i.e., of the form (x, y, 1), divide by the third co-efficient
Vx = Vx/Vx[2]
Vy = Vy/Vy[2]
Vz = Vz/Vz[2]

print("Hence the Vanishing points for this image in homogeneous form are :")
print("Along x-axis : ", Vx)
print("Along y-axis : ", Vy)
print("Along z-axis : ", Vz)
```

```
Hence the Vanishing points for this image in homogeneous form are :
Along x-axis :  [ 1.97460092e+03 -5.10376551e+02  1.00000000e+00]
Along y-axis :  [-878.83561644 -577.32420091    1.        ]
Along z-axis :  [5.56364796e+02 1.38508546e+03 1.00000000e+00]
```

```
#computing projection matrix

wo = P1 #world origin
ref_x = P2
ref_y = P7
ref_z = P3

# reference axis distance from World Origin
ref_x_dis = np.sqrt(np.sum(np.square(np.array([ref_x]) - wo)))
ref_y_dis = np.sqrt(np.sum(np.square(np.array([ref_y]) - wo)))
ref_z_dis = np.sqrt(np.sum(np.square(np.array([ref_z]) - wo)))

#compute scaling constants evaluated wrt reference points
```

```
ax,_,_,_ = np.linalg.lstsq( (np.array([Vx]) - wo).T , (np.array([ref_x]) - wo).T )
ax = ax[0][0]/ref_x_dis

ay,_,_,_ = np.linalg.lstsq( (np.array([Vy]) - wo).T , (np.array([ref_y]) - wo).T )
ay = ay[0][0]/ref_y_dis

az,_,_,_ = np.linalg.lstsq( (np.array([Vz]) - wo).T , (np.array([ref_z]) - wo).T )
az = az[0][0]/ref_z_dis

px = ax*Vx
py = ay*Vy
pz = az*Vz

P = np.zeros([3,4])
P[:,0] = px
P[:,1] = py
P[:,2] = pz
P[:,3] = wo
""" variable description of P : px, py, pz, wo are arranged as columns in P """

#computing Homograph matrix

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

row,col,_ = image.shape

warp_xy = cv2.warpPerspective(image,Hxy,(row,col),flags=cv2.WARP_INVERSE_MAP)
warp_yz = cv2.warpPerspective(image,Hyz,(row,col),flags=cv2.WARP_INVERSE_MAP)
warp_zx = cv2.warpPerspective(image,Hzx,(row,col),flags=cv2.WARP_INVERSE_MAP)
```

```
cv2.imshow("Txy",warp_xy)
cv2.imshow("Tyz",warp_yz)
cv2.imshow("Tzx",warp_zx)

cv2.waitKey(0)
cv2.destroyAllWindows()


cv2.imwrite("Hxy.png", warp_xy)
cv2.imwrite("Hyz.png", warp_yz)
cv2.imwrite("Hzx.png", warp_zx)
```