**ECE 558 Project 01 – Option1**

**DAYYAPP**

# Implementation Of the Paper "Single View Metrology" (Criminisi, Reid and Zisserman, ICCV99)

## *Image acquisition*:

The 3-point perspective image picture which is given as the input is as given below:

Fig1: This is the original image fed as input

## *Annotation:*

The input image is then annotated to find the vertices of the box. The annotated image is as below:

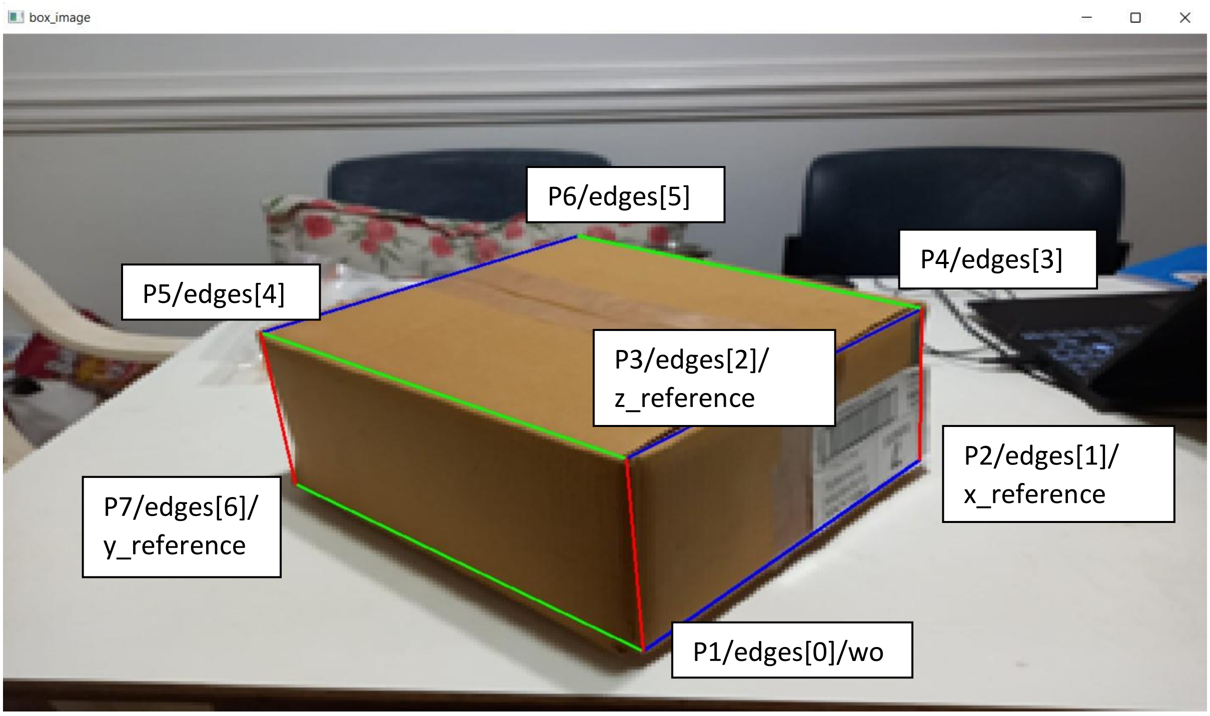



Fig2

For this image,

The vertices are as below:

P1 = [612,587,1]

P2 = [875,405,1]

P3 = [596,404,1]

P4 = [877,261,1]

P5 = [248,284,1]

P6 = [550,192,1]

P7 = [281,428,1]

## *Computing Vanishing points:*

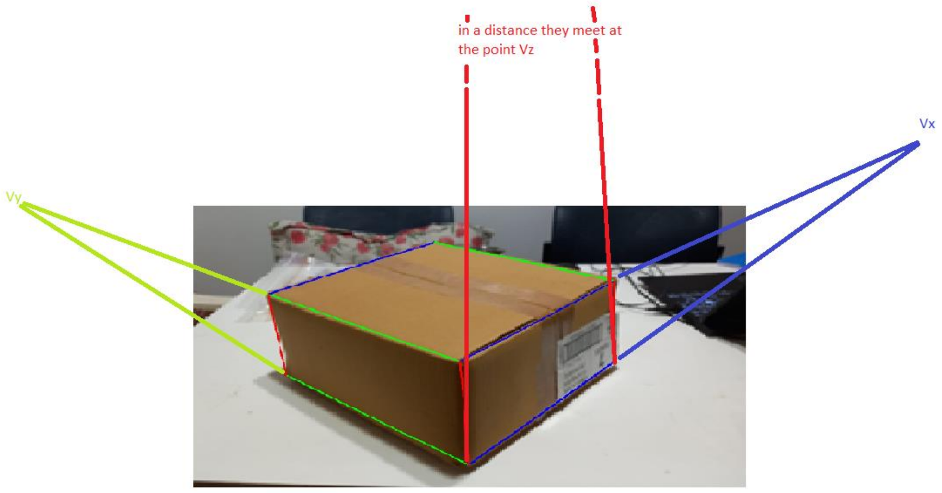



Fig3 – Vanishing points

The vanishing points can be computed for two lines by taking the vector cross product of the two lines. Now, the two-line equations are obtained by taking the cross product of the corresponding edge coordinates.

In this project, Vx is the vanishing point with respect to x-coordinate system and Vy for the y-coordinate system and Vz for the z-coordinate system. These 3 points are found from the lines depicted in the picture (Fig3).

For this image:

Vx= [ 1.65581830e+03 -1.35338136e+02  1.00000000e+00]

Vy= [-697.4980784   -42.03382014    1.        ]

Vz= [8.36767041e+02 3.15777303e+03 1.00000000e+00]

## *Projection matrix:*

The mapping of the 3D co-ordinates onto the 2D plane we use the projection matrix. For the homogeneous co-ordinate system, the projection matrix can be formed using the vanishing points.

Projection matrix = [ Vx*scaling factor(ax) Vy*scaling factor(ay) Vz*scaling factor(az) world_origin]

Here, the scaling factors can be calculated using the below formula:

$$\boldsymbol{Scaling\ factor} = (\boldsymbol{Vanishing_{point}}^{-1} * (\boldsymbol{reference_{coordinate}} - \boldsymbol{world_{origin}})) / (\boldsymbol{distance\ between\ reference\ coordinate\ and\ world\ origin})$$

Here,

- Vanishing points are Vx,Vy and Vz
- Reference coordinates are :

x_reference is P2

y_reference is P7

z_reference is P3

World Origin is wo which is P1

For this figure, the projection matrix is given by:

***Projection matrix:***

[[ 8.63049764e-01 -9.24131434e-01 -2.52374139e-01 6.12000000e+02]

[-7.05412828e-02 -5.56915864e-02 -9.52403969e-01 5.87000000e+02]

[ 5.21222507e-04 1.32492327e-03 -3.01606214e-04 1.00000000e+00]]

## *Homograph matrix:*

To map the points of one image to another image we use the homograph matrix. The homograph matrix can be calculated from the projection matrix corresponding to each planes. It is as given below:

Hxy = [Projection matrix[0] Projection matrix[1] Projection matrix[3]]

Hyz = [Projection matrix[1] Projection matrix[2] Projection matrix[3]]

Hxz = [Projection matrix[0] Projection matrix[2] Projection matrix[3]]

The homography transformation matrices are given below for XY, YZ and ZX planes:

XY plane:

[[ 8.63049764e-01 -9.24131434e-01  6.12000000e+02]
 [-7.05412828e-02 -5.56915864e-02  5.87000000e+02]
 [ 5.21222507e-04  1.32492327e-03  1.00000000e+00]]

YZ plane:

[[-9.24131434e-01 -2.52374139e-01  6.12000000e+02]
 [-5.56915864e-02 -9.52403969e-01  5.87000000e+02]
 [ 1.32492327e-03 -3.01606214e-04  1.00000000e+00]]

ZX plane:

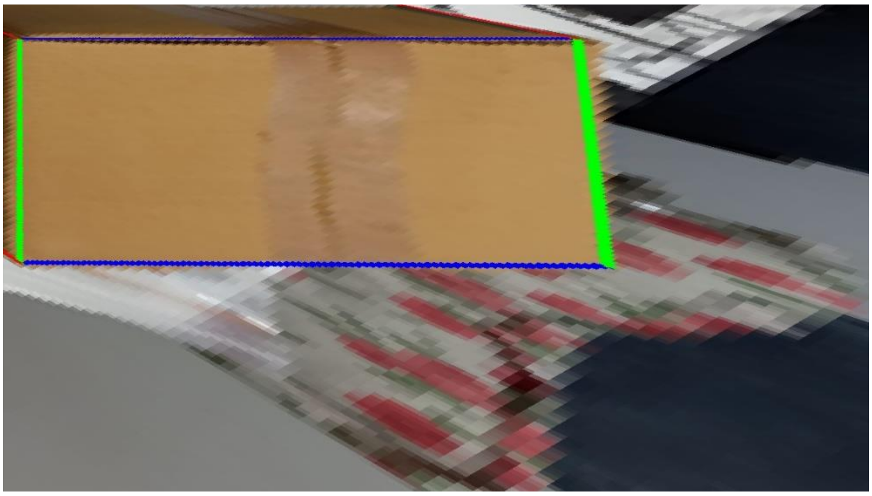
[[ 8.63049764e-01 -2.52374139e-01  6.12000000e+02]
 [-7.05412828e-02 -9.52403969e-01  5.87000000e+02]
 [ 5.21222507e-04 -3.01606214e-04  1.00000000e+00]]

## *Texture maps for XY, YZ and XZ planes*

Using the homograph matrices, the texture maps for the planes can be created by using reverse warping which warps one image onto the other.

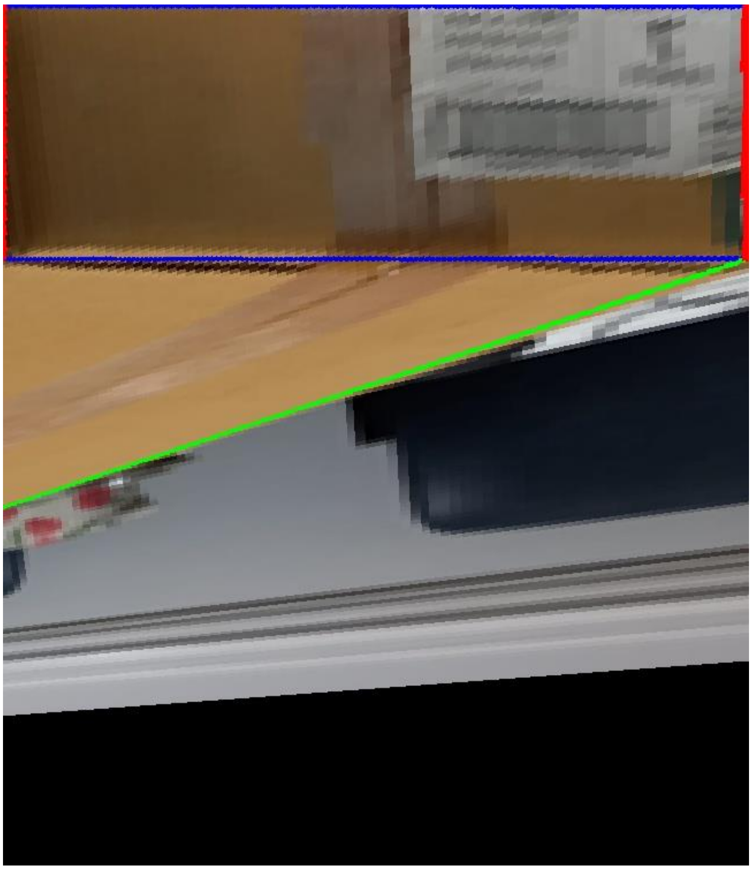
The texture maps for the corresponding planes are given below:

XY plane: Fig4(a)



YZ plane:Fig4(b)

XZ plane:Fig4(c)



Cropped images:

XY:Fig5(a)

YZ:Fig5(b)

XZ:Fig5(c)

***Visualizing the reconstructed 3D model*:**

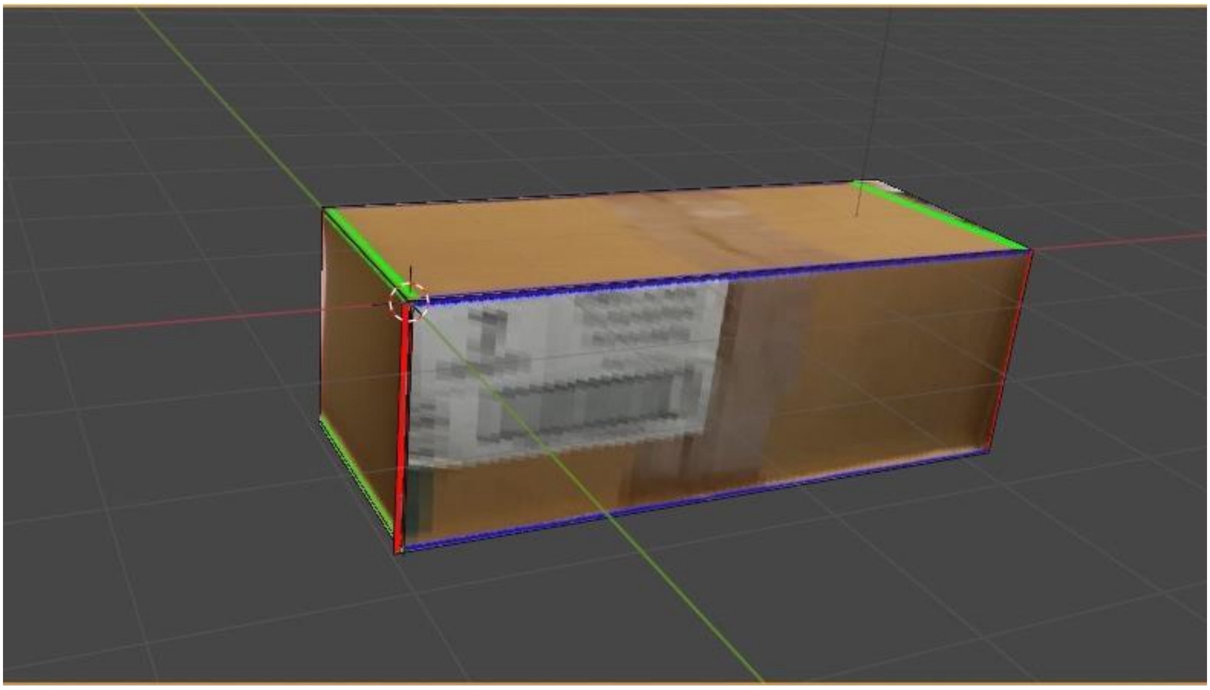


Fig6: Rendered 3D model image using the Blender tool

Using the blender tool, the 3D model with the cropped images has been constructed as above.

***Python Code:***

```
import numpy as np
import cv2

img = cv2.imread("image_box.png")
cv2.imshow('image', img)
cv2.waitKey(0)

#annotation code begins
def edge_lines(event, x, y, flag, parameters):
    global index, vertices
    if event == cv2.EVENT_LBUTTONDOWN:
        vertices = [(x,y)]
    elif event == cv2.EVENT_LBUTTONUP:
        vertices.append((x,y))
        print(f"edge line starts at: {vertices[0]} edge line ends at: {vertices[1]}")
        cv2.line(img, vertices[0], vertices[1],(0,255,0),4)  #green color lines drawn
to get vertex coordinates
        cv2.imshow("image", img)

vertices = []
index = 0
img = cv2.imread("image_box.png")
cv2.imshow('image', img)
cv2.setMouseCallback('image', edge_lines)
cv2.waitKey(0)
cv2.imwrite("annotated_box_image.png",img)
cv2.destroyAllWindows()
```

```
#annotation code ends

#marking points and drawing lines of different colors for each axes
P3 = [596,404,1]
P4 = [877,261,1]
P1 = [612,587,1]
P2 = [875,405,1]
P5 = [248,284,1]
P6 = [550,192,1]
P7 = [281,428,1]

#x axis
cv2.line(img,(P1[0],P1[1]),(P2[0],P2[1]),(250,0,0),2)
cv2.line(img,(P3[0],P3[1]),(P4[0],P4[1]),(250,0,0),2)
cv2.line(img,(P5[0],P5[1]),(P6[0],P6[1]),(250,0,0),2)

#y axis
cv2.line(img,(P1[0],P1[1]),(P7[0],P7[1]),(0,255,0),2)
cv2.line(img,(P3[0],P3[1]),(P5[0],P5[1]),(0,255,0),2)
cv2.line(img,(P4[0],P4[1]),(P6[0],P6[1]),(0,255,0),2)

#z axis
cv2.line(img,(P1[0],P1[1]),(P3[0],P3[1]),(0,0,255),2)
cv2.line(img,(P2[0],P2[1]),(P4[0],P4[1]),(0,0,255),2)
cv2.line(img,(P7[0],P7[1]),(P5[0],P5[1]),(0,0,255),2)

cv2.imshow('box_image',img)
```

```
cv2.waitKey(0)
print(np.array(P1))
#defining edges for the marked points

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

#defining reference co-ordinates and the distance between the reference and the world origin

x_reference = edges[1]
y_reference = edges[6]
z_reference = edges[2]
wo = edges[0]

length_x_wo = np.sqrt(np.sum(np.square(x_reference - wo)))
length_y_wo = np.sqrt(np.sum(np.square(y_reference - wo)))
length_z_wo = np.sqrt(np.sum(np.square(z_reference - wo)))

print(edges[0])

#vanishing point calculations:
```

```
#vanishing point for x axis
ax1,bx1,cx1 = np.cross(edges[0],edges[1])
ax2,bx2,cx2 = np.cross(edges[2],edges[3])
ax3,bx3,cx3 = np.cross(edges[4],edges[5])
Vx1_2 = np.cross([ax1,bx1,cx1],[ax2,bx2,cx2])
Vx2_3 = np.cross([ax3,bx3,cx3],[ax2,bx2,cx2])
Vx1_3 = np.cross([ax1,bx1,cx1],[ax2,bx2,cx2])

Vx1_2 = Vx1_2/Vx1_2[2]
Vx2_3 = Vx2_3/Vx2_3[2]
Vx1_3 = Vx1_3/Vx1_3[2]

#vanishing point for y axis
ay1,by1,cy1 = np.cross(edges[0],edges[6])
ay2,by2,cy2 = np.cross(edges[2],edges[4])
Vy = np.cross([ay1,by1,cy1],[ay2,by2,cy2])
Vy = Vy/Vy[2]

#vanishing point for z axis
az1,bz1,cz1 = np.cross(edges[0],edges[2])
az2,bz2,cz2 = np.cross(edges[1],edges[3])
Vz = np.cross([az1,bz1,cz1],[az2,bz2,cz2])
Vz = Vz/Vz[2]

print(Vx1_2,Vy,Vz)
```

```
#calculating projection matrix
Vx1_2 = np.array(Vx1_2)
Vy = np.array(Vy)
Vz = np.array(Vz)

ax,resid,rank,s = np.linalg.lstsq( np.array([Vx1_2]).T , (x_reference - wo).T )
ax = ax[0]/length_x_wo
ay,resid,rank,s = np.linalg.lstsq( np.array([Vy]).T , (y_reference - wo).T )
ay = ay[0]/length_y_wo
az,resid,rank,s = np.linalg.lstsq( np.array([Vz]).T , (z_reference - wo).T )
az = az[0]/length_z_wo

projection_matrix_x = ax*Vx1_2
projection_matrix_y = ay*Vy
projection_matrix_z = az*Vz

Projection_matrix = np.empty([3,4])
Projection_matrix[:,0] = projection_matrix_x
Projection_matrix[:,1] = projection_matrix_y
Projection_matrix[:,2] = projection_matrix_z
Projection_matrix[:,3] = wo

#Calculating homograph matrix

Hxy = np.zeros((3,3))
Hxy[:,0] = projection_matrix_x
Hxy[:,1] = projection_matrix_y
```

```
Hxy[:,2] = wo

Hyz = np.zeros((3,3))
Hyz[:,0] = projection_matrix_y
Hyz[:,1] = projection_matrix_z
Hyz[:,2] = wo

Hzx = np.zeros((3,3))
Hzx[:,0] = projection_matrix_x
Hzx[:,1] = projection_matrix_z
Hzx[:,2] = wo

print("The homography transformation matrices are given below for XY, YZ and ZX planes:")
print("XY plane: \n")
print(Hxy)
print("YZ plane: \n")
print(Hyz)
print("ZX plane: \n")
print(Hzx)

Hxy[0,2] = Hxy[0,2] +50
Hxy[1,2] = Hxy[1,2] -150

height,width,channel = img.shape
Txy = cv2.warpPerspective(img,Hxy,(width,height),flags=cv2.WARP_INVERSE_MAP)
```

```
cv2.imshow("Txy",Txy)
cv2.waitKey(0)


Tyz = 
cv2.warpPerspective(img,Hyz,(height,width),flags=cv2.WARP_INVERSE_MAP)
cv2.imshow("Tyz",Tyz)
cv2.waitKey(0)


Tzx = 
cv2.warpPerspective(img,Hzx,(height,width),flags=cv2.WARP_INVERSE_MAP)
cv2.imshow("Tzx",Tzx)
cv2.waitKey(0)


cv2.imwrite("box_image_XY_plane.jpg",Txy)
cv2.imwrite("box_image_YZ_plane.jpg.jpg",Tyz)
cv2.imwrite("box_image_ZX_plane.jpg.jpg",Tzx)


cv2.destroyAllWindows()
```