DIS Project
Unity id: sdhotre

AIM: Implementation of the paper “single view metrology” to build 3D model using 3 point perspective image(Criminisi, Reid and Zisserman, ICCV99)

Step 1) Image acquisition

Fig 1: Three point perspective image

Step 2) Annotation, labeling and computing vanishing point

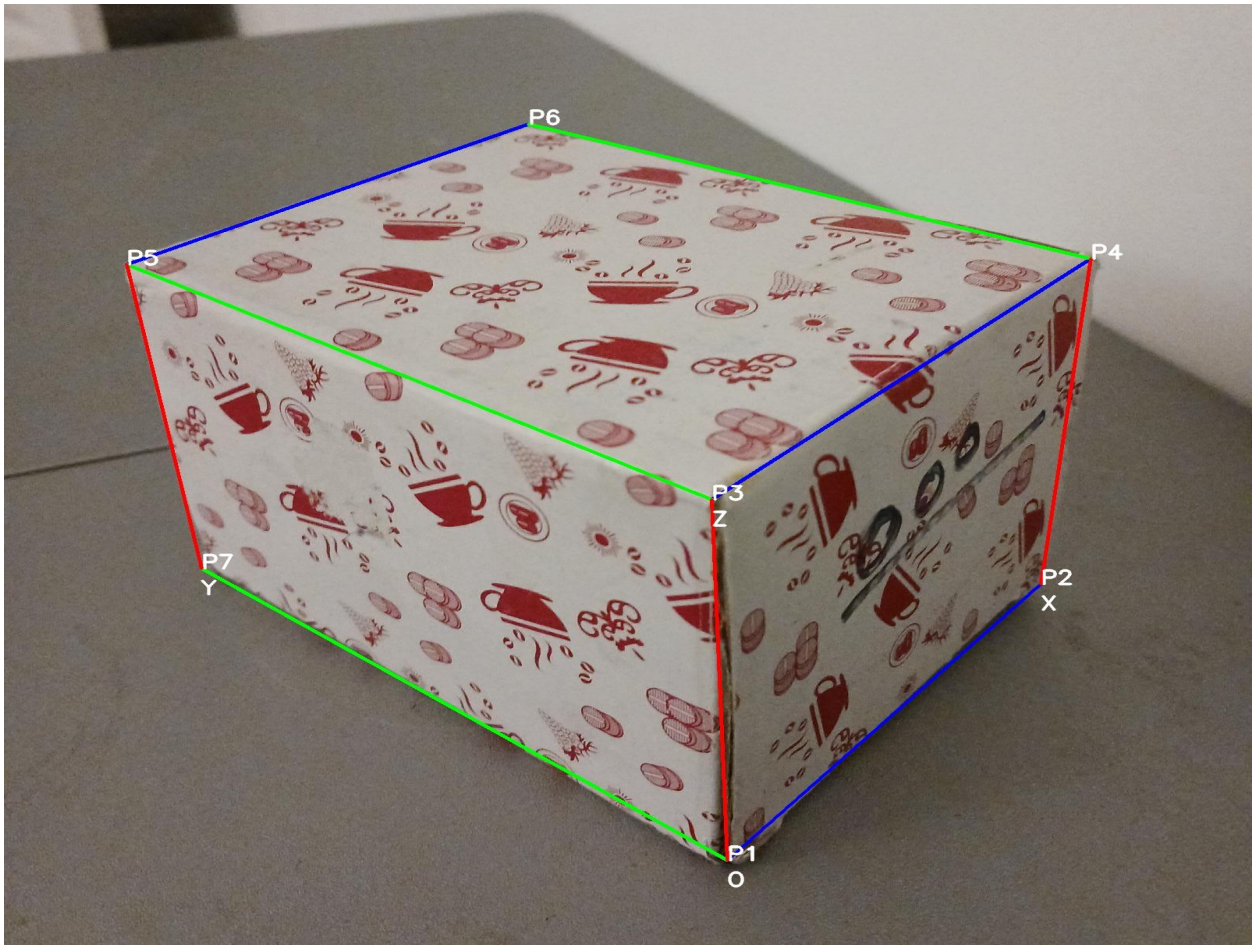


Fig 2: Annotation and labeling

Computing vanishing point(V) using Prof. Robert T Collins Method

1) Specifying endpoints(e1, e2) of the line as
   e1 = (x1_i , y1_i, w)
   e2 = (x2_i , y2_i, w)
   And making it homogenous coordinates, so w value is 1
2) Computing homogeneous coordinate vector representing the line by
   l = e1 x e2
3) If number of lines are 2 then vanishing point is intersection of this 2 lines, so
   V = l1xl2
   V= [Vx Vy Vz]
   And to make it homogenous, dividing by Vz, so
   V = [Vx Vy 1]
4) If number of lines are greater than 2 then
   a) M = sum ([a_i*a_i a_i*b_i a_i*c_i][a_i*b_i b_i*b*i b_i*c_i][a_i*c_i b_i*c_i c_i*c_i]) , For i = 1 to n

b) Do eigen decomposition and take the eigenvector associated with the smallest values of eigenvalues. This is vanishing point

Step3) computing projection matrix and homography matrix

Projection_matrix = [P1 P2 P3 P4]
P1 = Vx*ax
P2 = Vy*ay
P3 = Vz*az
P4 = Wo = world point(reference point)

Where ax, ay and az are unknown scaling factor,
Vx, Vy and Vz are vanishing points

Unknown scaling factor is given by equation
ax*reference_distance_x*[Vx-reference_x] = [reference_x - Wo]

Reference_distance_x = euclidean_distance(Wo,reference_ x)
Similarly for ay and az

Now Homography matrix are
Hxy = [P1 P2 P4]
Hxz = [P1 P3 P4]
Hyz = [P2 P3 P4]

Step 4) Texture maps and cropped images

Fig 3: XY

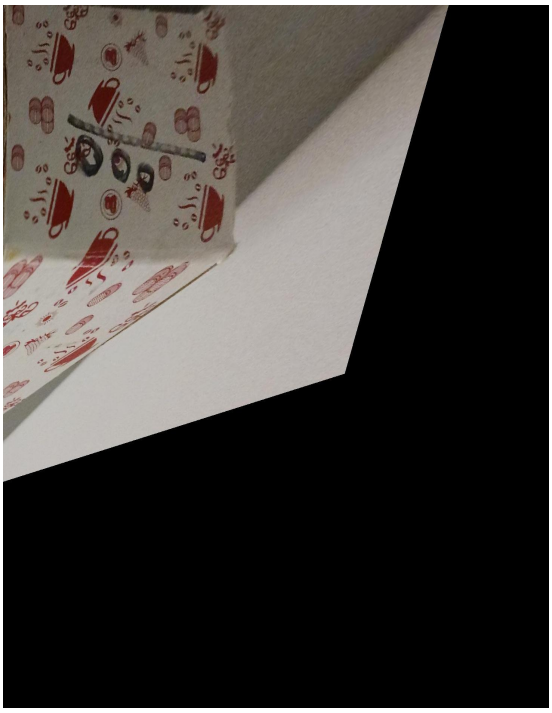


Fig 4: XZ

Fig 5: YZ

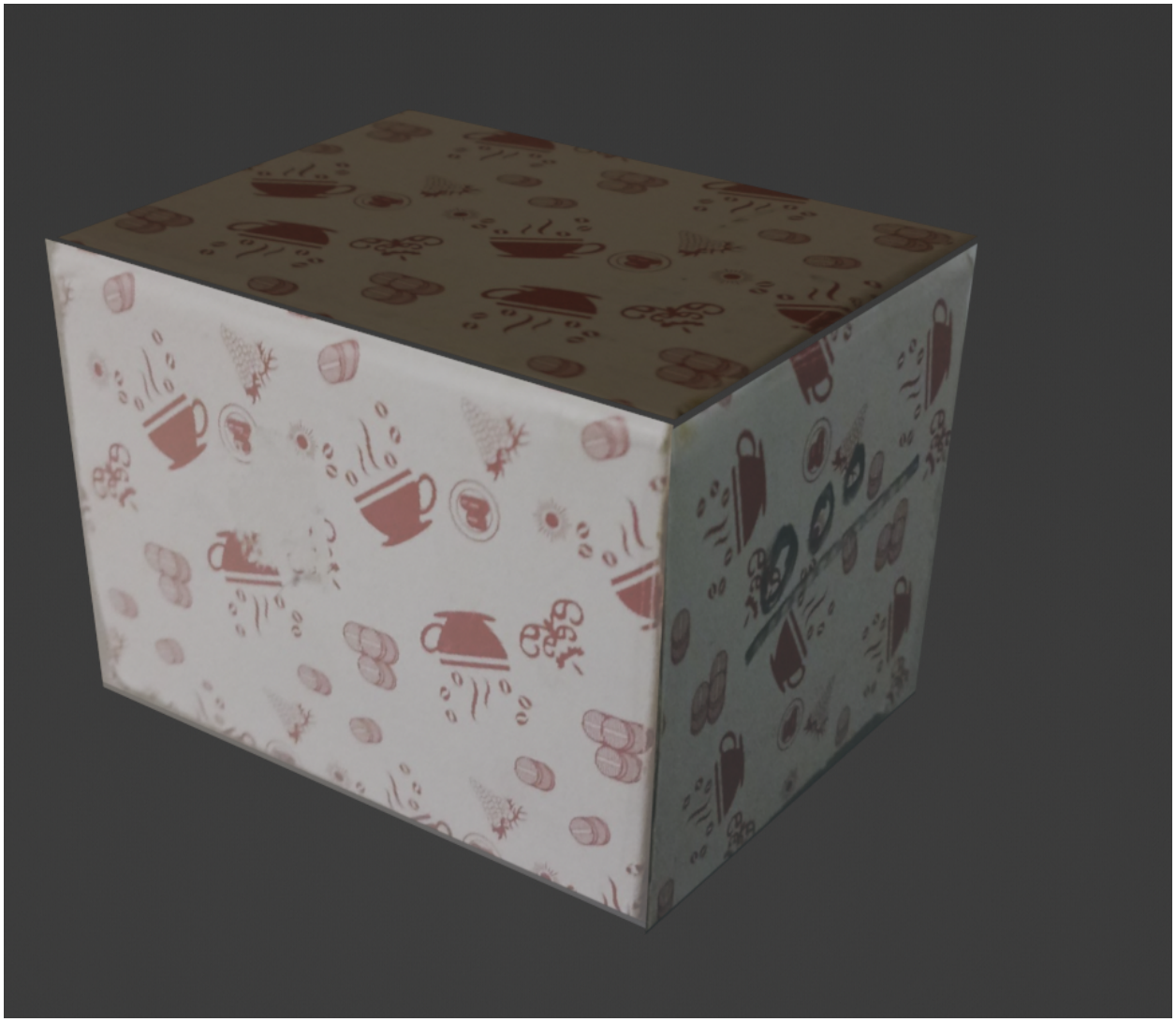
Fig 6: XY_cropped

Fig 7: XZ_cropped

Fig 8: YZ_cropped

Step 5) Reconstructed 3D model

Blender is used for 3d modeling using cropped texture images.



Code:

Annotation Code is to specify endpoints along the axes
Single Value Metrology code is the main code

Annotation Code:

```
#importing packages
import cv2
import numpy as np
import pandas as pd

img = cv2.imread('sample3.jpeg')


# Annotation
font = cv2.FONT_HERSHEY_SIMPLEX

#corner points of the box in the image
P1 = (927, 1329)
P2 = (1329, 900)
P3 = (907, 769)
P4 = (1393, 395)
P5 = (157, 404)
P6 = (672, 186)
P7 = (253, 876)

#endpoints of the line
endpoints = {
    "x" : [(P1, P2), (P3, P4), (P5, P6)],
    "y" : [(P1, P7), (P3, P5), (P4, P6)],
    "z" : [(P1, P3), (P2, P4), (P7, P5)],
}

#homogenous reference points
reference_point = {
    "x" : list(P2) + [1],
    "y" : list(P7) + [1],
    "z" : list(P3) + [1],
    "o" : list(P1) + [1]
}

# draw x lines
```

```
cv2.line(img,(P1[0],P1[1]),(P2[0],P2[1]),(255,0,0),4)       #blue
cv2.line(img,(P3[0],P3[1]),(P4[0],P4[1]),(255,0,0),4)
cv2.line(img,(P5[0],P5[1]),(P6[0],P6[1]),(255,0,0),4)

# draw y lines
cv2.line(img,(P1[0],P1[1]),(P7[0],P7[1]),(0,255,0),4)      #green
cv2.line(img,(P3[0],P3[1]),(P5[0],P5[1]),(0,255,0),4)
cv2.line(img,(P4[0],P4[1]),(P6[0],P6[1]),(0,255,0),4)


# draw z lines
cv2.line(img,(P1[0],P1[1]),(P3[0],P3[1]),(0,0,255),4)     #red
cv2.line(img,(P7[0],P7[1]),(P5[0],P5[1]),(0,0,255),4)
cv2.line(img,(P2[0],P2[1]),(P4[0],P4[1]),(0,0,255),4)

#labeling points
cv2.putText(img, 'P1', P1, font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'P2', P2, font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'P3', P3, font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'P4', P4, font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'P5', P5, font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'P6', P6, font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'P7', P7, font, 1, (255, 255, 255), 2, cv2.LINE_AA)

#labeling axis and reference point
cv2.putText(img, 'O', (P1[0], P1[1] + 40 ), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'X', (P2[0], P2[1] + 40 ), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'Y', (P7[0], P7[1] + 40 ), font, 1, (255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img, 'Z', (P3[0], P3[1] + 40 ), font, 1, (255, 255, 255), 2, cv2.LINE_AA)

data = np.zeros((7,2))
data[0,:] = P1
data[1,:] = P2
data[2,:] = P3
data[3,:] = P4
data[4,:] = P5
data[5,:] = P6
data[6,:] = P7
data = np.array(data)
df = pd.DataFrame({"X" : data[:,0], "Y" : data[:,1]})

#save csv file
df.to_csv("coordinates.csv", index=False)
```

```
cv2.imshow('Annotation', img)
cv2.imwrite('Annotation_labelling.jpeg', img)
cv2.waitKey(0)
```

Single View Metrology Code:

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

img = cv2.imread('sample3.jpeg')

# computing vanishing_points

df = pd.read_csv('coordinates.csv')     #fetching endpoints from the csv file
data = np.array(df)

P1 = data[0]
P2 = data[1]
P3 = data[2]
P4 = data[3]
P5 = data[4]
P6 = data[5]
P7 = data[6]

endpoints = {
    "x" : [(P1, P2), (P3, P4), (P5, P6)],
    "y" : [(P1, P7), (P3, P5), (P4, P6)],
    "z" : [(P1, P3), (P2, P4), (P7, P5)],
}

#homogenous reference points
reference_point = {
    "x" : list(P2) + [1],
    "y" : list(P7) + [1],
    "z" : list(P3) + [1],
    "o" : list(P1) + [1]
}

vanishing_points = {}
for key in endpoints.keys():
    lines = []
    for ep in endpoints[key]:
```

```
        e1, e2 = ep
        # Homogeneous coordinates
        e1 = list(e1) + [1]
        e2 = list(e2) + [1]
        lines.append(np.cross(e1, e2))

    if len(lines) == 2:
        vanishing = np.cross(lines[0], lines[1])
        vanishing_points[key] = vanishing / vanishing[-1]
    else:
        M = np.zeros((3, 3), dtype='float64')
        for j in range(3):
            a, b, c = lines[j]
            M += np.array([[a * a, a * b, a * c], [a * b, b * b, b * c], [a * c, b * c, c * c]])
        # Compute vanishing points
        eig_values, eig_vectors = np.linalg.eig(M)
        vanishing = eig_vectors[:, np.argmin(eig_values)]
        vanishing = vanishing / vanishing[-1]
        vanishing_points[key] = vanishing



#projection matrix, reference length and homography matrix
reference_length = {}
scaling_factor = {}
projection_matrix = np.zeros((3, 4), dtype='float64')

for key in vanishing_points.keys():
    reference_length[key] = np.sqrt(np.sum(np.square(np.array(reference_point[key]) -
np.array(reference_point["o"]))))
    A = (np.array(vanishing_points[key])-np.array(reference_point[key]))
    B = (np.array(reference_point[key] )-np.array(reference_point['o']))
    a,resid,rank,s = np.linalg.lstsq(A.reshape(-1,1) , B.reshape(-1,1))
    scaling_factor[key] = a[0][0]/reference_length[key]
    projection_matrix[:, list(vanishing_points).index(key)] = scaling_factor[key]*vanishing_points[key]

projection_matrix[:, 3] = reference_point['o']

Hxy = projection_matrix[:, [0,1,3]]
Hyz = projection_matrix[:, [1,2,3]]
Hxz = projection_matrix[:, [0,2,3]]

print(scaling_factor)
print(projection_matrix)
```

```
#texture maps
row, columns = img.shape[:2]
frame_xy = cv2.warpPerspective(img, Hxy, (row, columns), flags=cv2.WARP_INVERSE_MAP)
cv2.imwrite('xy.png', frame_xy)
frame_xz = cv2.warpPerspective(img, Hxz, (row, columns), flags=cv2.WARP_INVERSE_MAP)
cv2.imwrite('xz.png', frame_xz)
frame_yz = cv2.warpPerspective(img, Hyz, (row, columns), flags=cv2.WARP_INVERSE_MAP)
cv2.imwrite('yz.png', frame_yz)
```

Output:

Vanishing Points

```
{'x': array([ 2.87090288e+03, -7.44076140e+02,  1.00000000e+00]),
 'y': array([-1.86968390e+03, -5.62324204e+02,  1.00000000e+00]),
'z': array([9.62270504e+02, 4.17159092e+03, 1.00000000e+00])}
```

Scaling_factor

```
{'x': 0.00044365852848761694,
'y': 0.0003899989598741755,
'z': -0.0002937993042476428}
```

Projection Matrix

```
[[ 1.27370055e+00 -7.29174778e-01 -2.82714404e-01  9.27000000e+02]
 [-3.30115725e-01 -2.19305854e-01 -1.22561051e+00  1.32900000e+03]
 [ 4.43658528e-04  3.89998960e-04 -2.93799304e-04  1.00000000e+00]]
```

Homography Matrix

```
Hxy = [[ 1.27370055e+00 -7.29174778e-01  9.27000000e+02]
 [-3.30115725e-01 -2.19305854e-01  1.32900000e+03]
 [ 4.43658528e-04  3.89998960e-04  1.00000000e+00]]

Hyz = [[-7.29174778e-01 -2.82714404e-01  9.27000000e+02]
 [-2.19305854e-01 -1.22561051e+00  1.32900000e+03]
 [ 3.89998960e-04 -2.93799304e-04  1.00000000e+00]]

Hxz = [[ 1.27370055e+00 -2.82714404e-01  9.27000000e+02]
 [-3.30115725e-01 -1.22561051e+00  1.32900000e+03]
 [ 4.43658528e-04 -2.93799304e-04  1.00000000e+00]]
```

Conclusion:

I have explored how the affine 3D geometry scenes can be computed from three perspective images.

References:

A. Criminisi, I. Reid and A. Zisserman _Department of Engineering Science University of Oxford, Single View Metrology. 1999
https://kusemanohar.files.wordpress.com/2014/03/vanishingpt_bobcollins.pdf