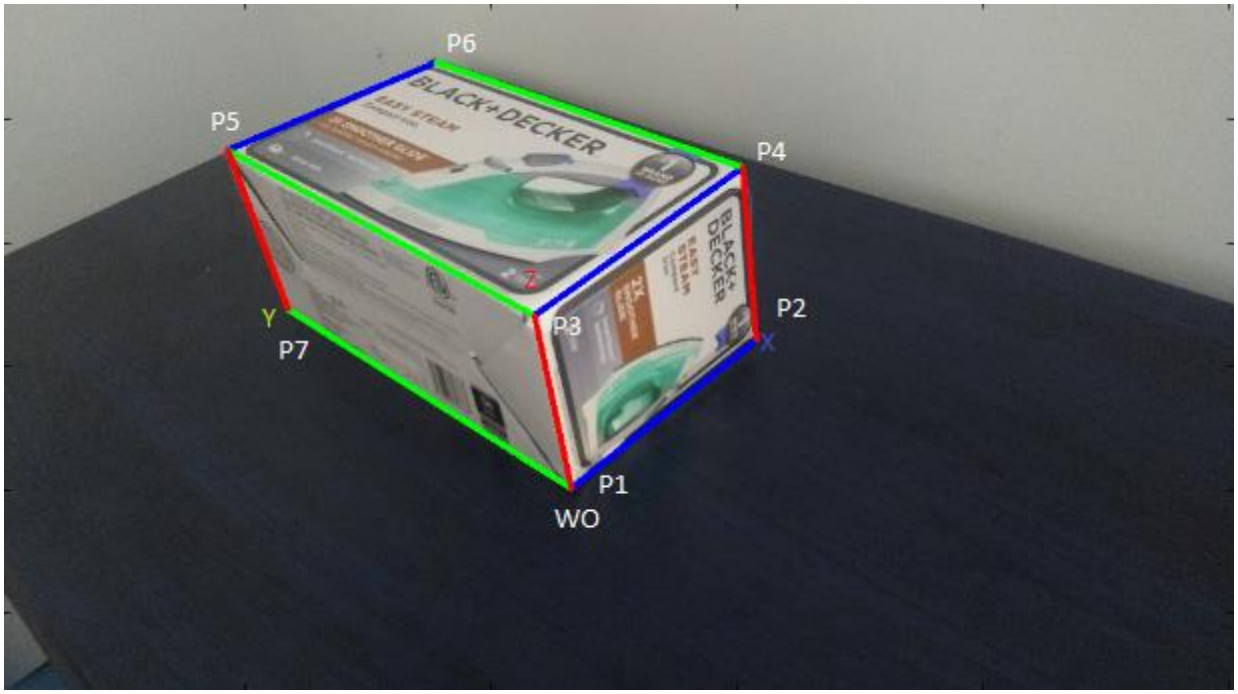
Digital Imaging Systems – Project 01

The project aims at building a 3D model of a 2D image using 3 point perspective focusing on the paper “Single View Metrology” by Criminisi, Reid and Zisserman, ICCV99.

Image Acquisition

Original image

Annotation



The image is annotated to get end points which will be used later to computing the vanishing points and ultimately the homograph matrices.

Computing Vanishing points

The vanishing points are calculated by first specifying the end points' coordinates and taking their cross-product. This gives us the equation of the line passing through those points.

e1 = [x1, y1, w]

e2 = [x2, y2, w]

Then the homogeneous coordinate vector representing the line passing through e1 and e2 is given by:-

(a, b, c) = e1 X e2

To find the point of intersection of 2 lines, we again take the cross product of the line coefficients and convert the result into homogeneous form.

Computing Projection Matrix and Homograph Matrix

Once we get the vanishing points in homogeneous coordinate system in the image plane, we proceed to calculate the projection matrix.

The projection matrix is nothing but the concatenation of the vanishing points to a scale. The last column being the world coordinates.

$$P = [aV_x \quad bV_y \quad cV_z \quad WO]$$

Where a, b, c are unknown constants and V_x, V_y, V_z are the vanishing points in column vector form. WO is the world origin in image frame.

To finds out the constants a, b and c we need the reference lengths and the reference coordinates along each axes in the image coordinate frame. The world origin W is also obtained from the image coordinate frame.

A typical solution of finding the constant a is given by the equation :-

$$a*ref\_x\_distance*[V_x - ref\_x] = [ref\_x - WO]$$

Here,

A	the unknown parameter
ref_x	the vector coordinate of a point on the x-axis(of the box) in the image frame
ref_x_distance	scalar pixel length from the World origin to ref_x in image frame
V_x	the vanishing point along the x axis in the image frame
WO	the world origin in the image frame

Similarly, the other constants b and c are also calculated from the equations:-

$$b*ref\_y\_distance*[V_y - ref\_y] = [ref\_y - WO]$$

$$c*ref\_z\_distance*[V_z - ref\_z] = [ref\_z - WO]$$

once we get the constants a, b and c we get the projection matrix from the equation:-

$P = [aV_x \quad bV_y \quad cV_z \quad WO]$

Now the homograph matrices are easily calculated by taking the columns of the projection matrix.

Hxy = [P1 P2 P4]

Hyz = [P2 P3 P4]

Hxz = [P1 P3 P4]

Computing texture maps for XY, YZ, XZ planes

In 3D world, objects in the distance appear smaller than objects close by- this is known as perspective. A perspective projection

The homograph matrices map the points of one image to the corresponding points in other image.

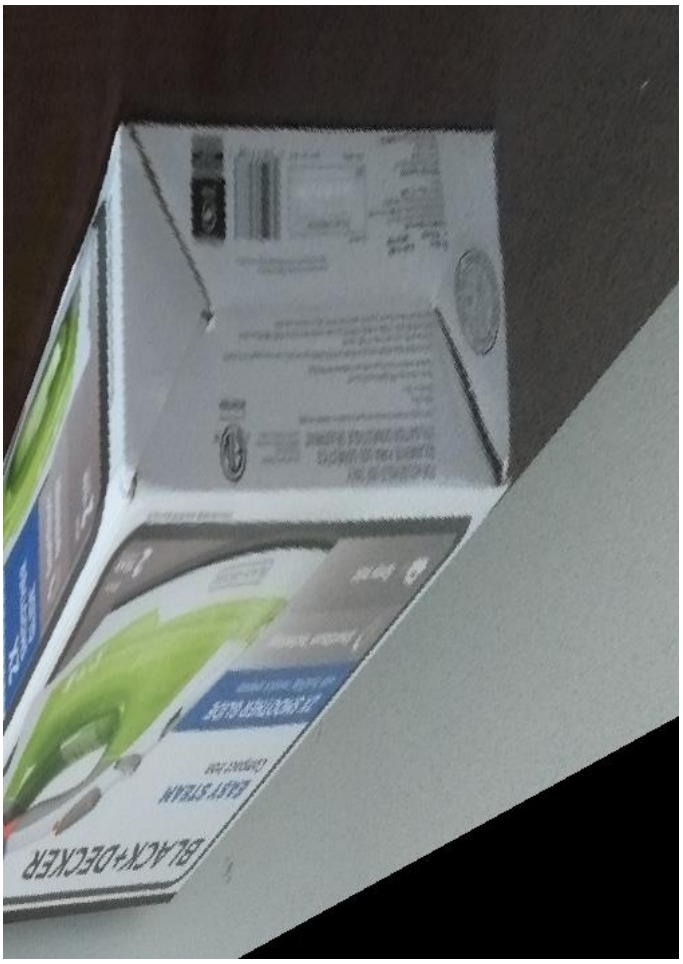
Now since we know the homograph between the 2 images (i.e. the original image and the homograph matrix), we can warp one image onto the other.

This will enhance the corresponding plane of the box from the original image. The respective plane is then cropped out to get the entire XY, YZ and ZX planes.

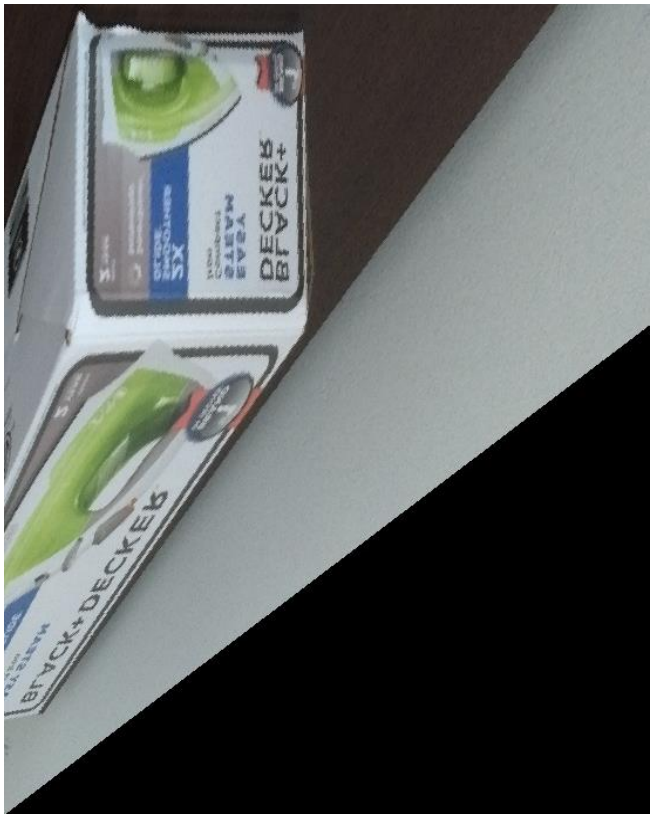
The perspective transformed images are:-


BLACK+DECKER
EASY STEAM

XY

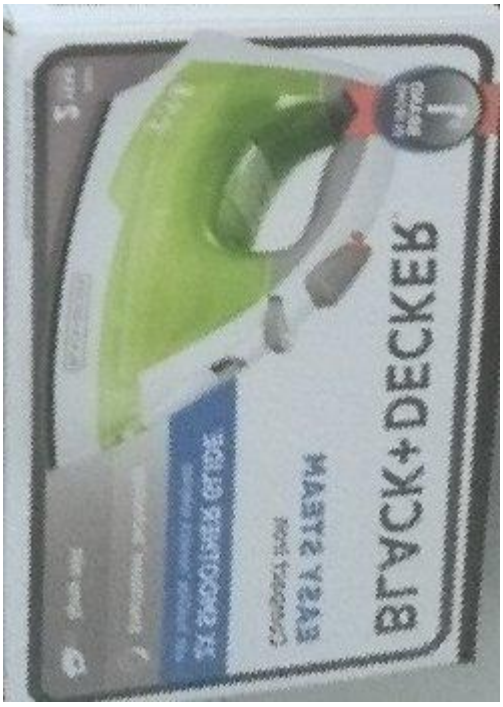

BLACK+DECKER

YZ


BLACK+DECKER
DECKER BLACK+

XZ

The cropped texture maps are: -



XY

YZ

XZ

Visualizing the reconstructed 3D model

The cropped texture maps are then used to create a wrl file for 3D modelling and visualized in view3Dscenes software.

Rendered 3D model

Results

P1 -> [465,400,1]

P2 -> [615,280,1]

P3 -> [435,260,1]

P4 -> [604,140,1]

P5 -> [186,126,1]

P6 -> [355,55,1]

P7 -> [235,255,1]

Points:-

X1 -> P1 – P2

X2 -> P3 – P4

X3 -> P5 – P6

Y1 -> P1 – P7

Y2 -> P3 – P5

Y3 -> P4 – P6

Z1 -> P1 – P3

Z2 -> P2 – P4

Z3 -> P7 – P5

WO -> P1

Ref_x -> P2

Ref_y -> P7

Ref_z -> P3

Vanishing Points:-

Vx = [2258.42105 -1034.73684 1]

Vy = [-877.138126 -446.130558 1]

Vz = [716.729323 1574.73684 1]

Projection Matrix:-

$$P = \begin{bmatrix} 1.07308505 & -0.667177557 & -0.280704070 & 465 \\ -0.49165350 & -0.339340278 & -0.616739159 & 400 \\ 0.00047514 & 0.000760629 & -0.000391645 & 1 \end{bmatrix}$$

Homograph matrices :-

Hxy =

$$\begin{bmatrix} 1.07308505 & -0.667177557 & 465 \\ -0.49165350 & -0.339340278 & 400 \\ 0.00047514 & 0.000760629 & 1 \end{bmatrix}$$

Hyz =

$$\begin{bmatrix} -0.667177557 & -0.280704070 & 565 \\ -0.339340278 & -0.616739159 & 500 \\ 0.000760629 & -0.000391645 & 100 \end{bmatrix}$$

Hxz =

$$\begin{bmatrix} 1.07308505 & -0.280704070 & 415 \\ -0.49165350 & -0.616739159 & 450 \\ 0.00047514 & -0.000391645 & 1 \end{bmatrix}$$

Codes:-

The python script annotate.py is used to specify the end points along each axes (x, y and z).

7 points are defined: P1 to P7 which are the end nodes along each of the axes.

The user is required to manually change these points for his own test image. This script creates a file named coordinates.csv containing the coordinates of these 7 points.

Then, the python script projection.py uses this csv file to load the coordinates and perform further calculations.

annotate.py

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

img = cv2.imread('image.jpg')
r,c,temp = img.shape
img = cv2.resize(img,(c/4,r/4))


P1 = [465,400,1]
P2 = [615,280,1]
P3 = [435,260,1]
P4 = [604,140,1]
P5 = [186,126,1]
P6 = [355,55,1]
P7 = [235,255,1]


X1_e1 = P1
X1_e2 = P2

X2_e1 = P3
X2_e2 = P4

X3_e1 = P5
X3_e2 = P6

Y1_e1 = P1
Y1_e2 = P7

Y2_e1 = P3
Y2_e2 = P5

Y3_e1 = P4
Y3_e2 = P6

Z1_e1 = P1
Z1_e2 = P3

Z2_e1 = P2
Z2_e2 = P4

Z3_e1 = P7
Z3_e2 = P5



###  X  ###
cv2.line(img,(X1_e1[0],X1_e1[1]),(X1_e2[0],X1_e2[1]),(0,0,255),2)        #blue
cv2.line(img,(X2_e1[0],X2_e1[1]),(X2_e2[0],X2_e2[1]),(0,0,255),2)
cv2.line(img,(X3_e1[0],X3_e1[1]),(X3_e2[0],X3_e2[1]),(0,0,255),2)

###  Y  ###
cv2.line(img,(Y1_e1[0],Y1_e1[1]),(Y1_e2[0],Y1_e2[1]),(0,255,0),2)     #green
cv2.line(img,(Y2_e1[0],Y2_e1[1]),(Y2_e2[0],Y2_e2[1]),(0,255,0),2)
cv2.line(img,(Y3_e1[0],Y3_e1[1]),(Y3_e2[0],Y3_e2[1]),(0,255,0),2)
```

```
###  Z  ###
cv2.line(img,(Z1_e1[0],Z1_e1[1]),(Z1_e2[0],Z1_e2[1]),(250,0,0),2)      #red
cv2.line(img,(Z2_e1[0],Z2_e1[1]),(Z2_e2[0],Z2_e2[1]),(250,0,0),2)
cv2.line(img,(Z3_e1[0],Z3_e1[1]),(Z3_e2[0],Z3_e2[1]),(255,0,0),2)



wo = P1
ref_x = P2
ref_y = P7
ref_z = P3

data = np.zeros((7,3))
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], "Z" : data[:,2]})
df.to_csv("coordinates.csv", index=False)

plt.imshow(img)
```

projection.py

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


img = cv2.imread('image.jpg')
r,c,temp = img.shape
img = cv2.resize(img,(c/4,r/4))
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

df = pd.read_csv('coordinates.csv');
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]


X1_e1 = P1
X1_e2 = P2

X2_e1 = P3
X2_e2 = P4

X3_e1 = P5
X3_e2 = P6

Y1_e1 = P1
Y1_e2 = P7

Y2_e1 = P3
Y2_e2 = P5

Y3_e1 = P4
Y3_e2 = P6

Z1_e1 = P1
Z1_e2 = P3

Z2_e1 = P2
Z2_e2 = P4

Z3_e1 = P7
Z3_e2 = P5

wo = P1
ref_x = P2
ref_y = P7
ref_z = P3

ref_x = np.array([ref_x])
ref_y = np.array([ref_y])
ref_z = np.array([ref_z])
```

```
ax1,bx1,cx1 = np.cross(X1_e1,X1_e2)
ax2,bx2,cx2 = np.cross(X2_e1,X2_e2)
Vx = np.cross([ax1,bx1,cx1],[ax2,bx2,cx2])
Vx = Vx/Vx[2]

ay1,by1,cy1 = np.cross(Y1_e1,Y1_e2)
ay2,by2,cy2 = np.cross(Y2_e1,Y2_e2)
Vy = np.cross([ay1,by1,cy1],[ay2,by2,cy2])
Vy = Vy/Vy[2]

az1,bz1,cz1 = np.cross(Z1_e1,Z1_e2)
az2,bz2,cz2 = np.cross(Z2_e1,Z2_e2)
Vz = np.cross([az1,bz1,cz1],[az2,bz2,cz2])
Vz = Vz/Vz[2]



length_x = np.sqrt(np.sum(np.square(ref_x - wo)))
length_y = np.sqrt(np.sum(np.square(ref_y - wo)))
length_z = np.sqrt(np.sum(np.square(ref_z - wo)))


ref_x = np.array(ref_x)
ref_y = np.array(ref_y)
ref_z = np.array(ref_z)
wo = np.array(wo)
Vx = np.array(Vx)
Vy = np.array(Vy)
Vz = np.array(Vz)


ax,resid,rank,s = np.linalg.lstsq( (Vx-ref_x).T , (ref_x - wo).T )
ax = ax[0][0]/length_x

ay,resid,rank,s = np.linalg.lstsq( (Vy-ref_y).T , (ref_y - wo).T )
ay = ay[0][0]/length_y

az,resid,rank,s = np.linalg.lstsq( (Vz-ref_z).T , (ref_z - wo).T )
az = az[0][0]/length_y

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

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


Hxy[0,2] = Hxy[0,2]
Hxy[1,2] = Hxy[1,2]

Hyz[0,2] = Hyz[0,2] + 100
Hyz[1,2] = Hyz[1,2] + 100

Hzx[0,2] = Hzx[0,2] - 50
Hzx[1,2] = Hzx[1,2] + 50



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.imshow("Txy",Txy)
cv2.imshow("Tyz",Tyz)
cv2.imshow("Tzx",Tzx)

cv2.waitKey(0)
cv2.imwrite("XY.jpg",Txy)
cv2.imwrite("YZ.jpg",Tyz)
cv2.imwrite("ZX.jpg",Tzx)
print "Saved"

cv2.destroyAllWindows()
```