# ECE 558: Digital Imaging Systems PROJECT 01

**NAME: VIKRAM PANDE**

**UNITY_ID: vspande**

## Project01 – option 1: 2D to 3D conversion-based on Single View Metrology

## Objective:

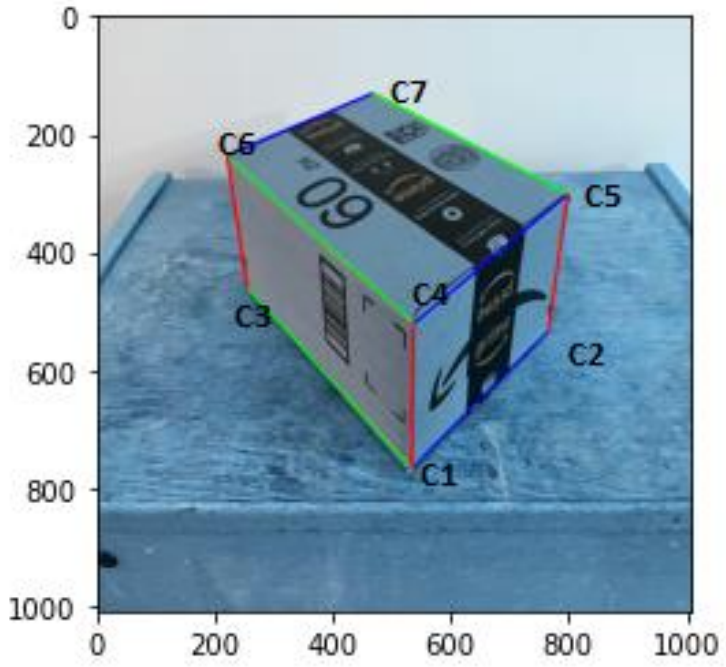
This project focuses on an implementation of the paper “single view metrology” (Criminisi, Reid and Zisserman, ICCV99). This project aims to create a 3D model with a single perspective image with some prior knowledge with Single View Metrology**.**

## General Description:

The Project is divided into 4 sub parts. The first part is the 3D perspective image acquisition. Second part is to calculate the Vanish points in the given image related to objects. Third part focuses on calculating the Projection and Homograph Matrices. Fourth part focuses on getting a texture map for each plane using warping. At last, the visualisation of 3D object is done.

1) **Image Acquisition**
   An image of box with 3D perspective was captured.
   Original Image: **IMG-2632.PNG** is the Input Image of a box.

Image with Annotation:




Co-ordinates:

```
c1 = [531,765]
c2 = [762,533]
c3 = [253,460]
c4 = [535,520]
c5 = [795,302]
c6 = [219,237]
c7 = [466,131]
```

**2) Computing Vanish Points**

The vanishing points can be found with the intersection of two lines. This can be calculated by specifying the endpoints of lines in homogenous co-ordinates.

$$e1 = (x1\_i, y1\_i, w)$$
$$e2 = (x2\_i, y2\_i, w)$$

where w = 1

Then, taking the cross products of the co-ordinates

$$(a\_i, b\_i, c\_i) = e1\ X\ e2$$

Vanish Points Result:

```
PS C:\Users\Vikram\Desktop\ECE 558 - Project01> & C:/Users/Vikram/anaconda3/python.exe "c:/Users/Vikram/Desktop/ECE 558 - Project01/main.py"
The vanish point along X axis:  [ 1.98786288e+03 -6.98169645e+02  1.00000000e+00]
The vanish point along Y axis:  [-702.3364961  -588.12097594    1.        ]
The vanish point along Z axis:  [5.05470046e+02 2.32870968e+03 1.00000000e+00]
```

**3) Compute Projection and Homograph Matrix**

**Projection Matrix:**

Projection Matrix is nothing but the multiplication of vanish points and a scaling factor, then finally concatenated into a 3X4 Matrix.

$$PM = [\, a_x V_x \quad a_y V_y \quad a_z V_z \quad w_0 \,]$$

Where $a_x, a_y, a_z - Scaling\ Factors$
$V_x, V_y, V_z - Vanish\ Points$
$w_0 - World\ Origin\ Coordinates$

Calculations for Scaling Factor:

First, we need to mark the reference points along with each axis and then we need to find the distances between the reference points and world origin co-ordinates. Once we have the distances and required points, we can use the following formula:

$$a \cdot (reference\_distance) \cdot (V - reference\_point) = reference\_point - w_0$$

Where $reference\_point - reference\ coordinate\ for\ each\ plane$
$a - Scaling\ Factor$
$V - Respective\ Vanish\ Point$
$w_0 - World\ origin\ coordinate$
$reference\_distance -$
$Euclidean\ distance\ between\ reference\ point\ and\ world\ coordinate$

**Final Projection Matrix:**

$$PM = [\, a_x V_x \quad a_y V_y \quad a_z V_z \quad w_0 \,]$$

**Homograph Matrix:**

Homograph Matrix can simply be calculated with the required columns from Projection Matrix for each different plane.

XY plane:

$$H_{xy} = [\, PM_1 \quad PM_2 \quad PM_4 \,]$$

YZ plane:

$$H_{yz} = [\, PM_2 \quad PM_3 \quad PM_4 \,]$$

XZ plane:

$$H_{xz} = [\, PM_1 \quad PM_3 \quad PM_4 \,]$$

Projection Matrix Result:

```
PS C:\Users\Vikram\Desktop\ECE 558 - Project01> & C:/Users/Vikram/anaconda3/python.exe
Projection Matrix
 [[ 1.14416723e+00 -4.95239030e-01 -2.79427206e-01  5.31000000e+02]
 [-4.01850064e-01 -4.14702159e-01 -1.28732621e+00  7.65000000e+02]
 [ 5.75576534e-04  7.05130706e-04 -5.52806656e-04  1.00000000e+00]]
```

Homograph Matrices Result:

```
Hxy
 [[ 1.14416723e+00 -4.95239030e-01  5.31000000e+02]
 [-4.01850064e-01 -4.14702159e-01  7.65000000e+02]
 [ 5.75576534e-04  7.05130706e-04  1.00000000e+00]]
Hyz
 [[-4.95239030e-01 -2.79427206e-01  5.31000000e+02]
 [-4.14702159e-01 -1.28732621e+00  7.65000000e+02]
 [ 7.05130706e-04 -5.52806656e-04  1.00000000e+00]]
Hxz
 [[ 1.14416723e+00 -2.79427206e-01  5.31000000e+02]
 [-4.01850064e-01 -1.28732621e+00  7.65000000e+02]
 [ 5.75576534e-04 -5.52806656e-04  1.00000000e+00]]
```

**4) Computing Texture Maps for XY, YZ, XZ**

Now, as we have the input image (the image acquired) and result image (homograph matrix), we can get the texture maps for each plane by warping one image into another.

Then, we need to crop the required portion of the plane from the entire image giving us the cropped XY, YZ and XZ texture maps.

Texture Maps:

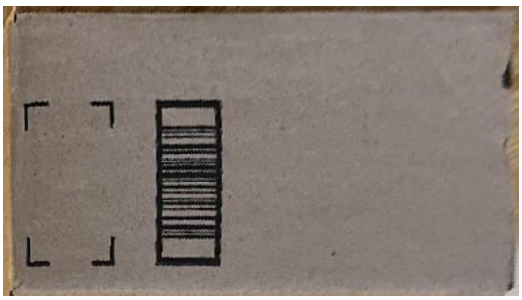
XY Plane

YZ Plane

XZ plane

Cropped Texture Maps:


XY_cropped



YZ_cropped


XZ_cropped

**5) Visualizing the 3D model**

Blender software was used to render the 3D image.

Rendered 3D Image of Box

box_output.blend

Blender Output File -

Main Python Code:

main.py python file contains all the steps from annotation to texture maps.

```
# Importing all the necessary libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import cv2
import math

# 1) ----- IMAGE ACQUSITION AND ANNOTATION -----
# Reading the image
img_ip = cv2.imread('IMG-2632.png')

#downscaling the image
w_ip, h_ip, d = img_ip.shape
down_width = int(w_ip/3)
down_height = int(h_ip/4)
down_points = (down_width, down_height)
img = cv2.resize(img_ip, down_points, interpolation= cv2.INTER_LINEAR)
img_copy = img.copy()

# --------- ANNOTATION ---------

# Plotting the co-ordinates and lines
# Total points taken are 7 so that we can plot two lines
c1 = [531,765]
c2 = [762,533]
c3 = [253,460]
c4 = [535,520]
c5 = [795,302]
c6 = [219,237]
c7 = [466,131]

# 1) X axis - Blue
cv2.line(img,c1,c2,(0,0,255),4)
cv2.line(img,c4,c5,(0,0,255),4)
cv2.line(img,c6,c7,(0,0,255),4)
# 2) Y axis - Green
cv2.line(img,c1,c3,(0,255,0),4)
cv2.line(img,c4,c6,(0,255,0),4)
cv2.line(img,c5,c7,(0,255,0),4)
# 3) Z axis - Red
```

```
cv2.line(img,c1,c4,(255,0,0),4)
cv2.line(img,c3,c6,(255,0,0),4)
cv2.line(img,c2,c5,(255,0,0),4)

# plt.imshow(img)
# plt.show()
# cv2.imwrite("annotated.png",img)


# 2) ----- COMPUTE VANISH POINTS -----
# adding 1 as third element

# 1) Blue X axis
v1_x1 = [c1[0], c1[1], 1]
v1_x2 = [c2[0], c2[1], 1]
v2_x1 = [c4[0], c4[1], 1]
v2_x2 = [c5[0], c5[1], 1]

b1_x1,b1_x2,b1_x3 = np.cross(v1_x1,v1_x2)
b2_x1,b2_x2,b2_x3 = np.cross(v2_x1,v2_x2)
Vx = np.cross([b1_x1,b1_x2,b1_x3],[b2_x1,b2_x2,b2_x3])
Vx = Vx/Vx[2]


# 2) Green Y axis
v1_y1 = [c1[0], c1[1], 1]
v1_y2 = [c3[0], c3[1], 1]
v2_y1 = [c4[0], c4[1], 1]
v2_y2 = [c6[0], c6[1], 1]

g1_y1,g1_y2,g1_y3 = np.cross(v1_y1,v1_y2)
g2_y1,g2_y2,g2_y3 = np.cross(v2_y1,v2_y2)
Vy = np.cross([g1_y1,g1_y2,g1_y3],[g2_y1,g2_y2,g2_y3])
Vy = Vy/Vy[2]


# 3) Red Z axis
v1_z1 = [c1[0], c1[1], 1]
v1_z2 = [c4[0], c4[1], 1]
v2_z1 = [c2[0], c2[1], 1]
v2_z2 = [c5[0], c5[1], 1]

r1_z1,r1_z2,r1_z3 = np.cross(v1_z1,v1_z2)
r2_z1,r2_z2,r2_z3 = np.cross(v2_z1,v2_z2)
Vz = np.cross([r1_z1,r1_z2,r1_z3],[r2_z1,r2_z2,r2_z3])
Vz = Vz/Vz[2]

# 3) ----- CONSTRUCTING PROJECTION MATRIX AND HOMOGRAPH MATRIX -----
#Taking one reference point and world coordinates
```

```
w0 = [c1[0], c1[1], 1]
ref_x = [c2[0], c2[1], 1]
ref_y = [c3[0], c3[1], 1]
ref_z = [c4[0], c4[1], 1]
ref_x = np.array([ref_x])
ref_y = np.array([ref_y])
ref_z = np.array([ref_z])

distance_x = np.sqrt(np.sum(np.square(ref_x - w0)))
distance_y = np.sqrt(np.sum(np.square(ref_y - w0)))
distance_z = np.sqrt(np.sum(np.square(ref_z - w0)))

# Converting all the required elements to array
Vx = np.array(Vx)
Vy = np.array(Vy)
Vz = np.array(Vz)
w0 = np.array(w0)
ref_x = np.array(ref_x)
ref_y = np.array(ref_y)
ref_z = np.array(ref_z)


#Getting a Scaling Factor
ax,resid,rank,s = np.linalg.lstsq( (Vx-ref_x).T , (ref_x - w0).T, rcond=None )
ax = ax[0][0]/distance_x

ay,resid,rank,s = np.linalg.lstsq( (Vy-ref_y).T , (ref_y - w0).T, rcond=None )
ay = ay[0][0]/distance_y

az,resid,rank,s = np.linalg.lstsq( (Vz-ref_z).T , (ref_z - w0).T, rcond=None )
az = az[0][0]/distance_z

# Constructing Projection Matrix
pmx = ax*Vx
pmy = ay*Vy
pmz = az*Vz
# --------- PROJECTION MATRIX ---------
proj_mat = np.empty([3,4])
proj_mat[:,0] = pmx
proj_mat[:,1] = pmy
proj_mat[:,2] = pmz
proj_mat[:,3] = w0

# print("Projection Matrix\n",proj_mat)

# Constructing Homography Matrix
Hxy = np.zeros((3,3))
Hyz = np.zeros((3,3))
Hzx = np.zeros((3,3))
```

```
Hxy[:,0] = pmx
Hxy[:,1] = pmy
Hxy[:,2] = w0

Hyz[:,0] = pmy
Hyz[:,1] = pmz
Hyz[:,2] = w0

Hzx[:,0] = pmx
Hzx[:,1] = pmz
Hzx[:,2] = w0

# Adjusting the image to make it visible
Hxy[0,2] = Hxy[0,2] + 30
Hxy[1,2] = Hxy[1,2]

Hyz[0,2] = Hyz[0,2] + 300
Hyz[1,2] = Hyz[1,2] + 600

Hzx[0,2] = Hzx[0,2] + -300
Hzx[1,2] = Hzx[1,2] + 400


# 4) ----- GETTING THE TEXTURE MAPS -----
w,h,temp = img.shape
# Getting Texture Maps
TM_xy = cv2.warpPerspective(img_copy,Hxy,(w,h),flags=cv2.WARP_INVERSE_MAP)
cv2.imshow("Txy",TM_xy)
cv2.imwrite("XY_plane.png",TM_xy)
cv2.waitKey(0)


TM_yz = cv2.warpPerspective(img_copy,Hyz,(w,h),flags=cv2.WARP_INVERSE_MAP)
cv2.imshow("Tyz",TM_yz)
cv2.imwrite("YZ_plane.png",TM_yz)
cv2.waitKey(0)


TM_zx = cv2.warpPerspective(img_copy,Hzx,(w,h),flags=cv2.WARP_INVERSE_MAP)
cv2.imshow("Txz",TM_zx)
cv2.imwrite("ZX_plane.png",TM_zx)
cv2.waitKey(0)

cv2.destroyAllWindows()
```