**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.

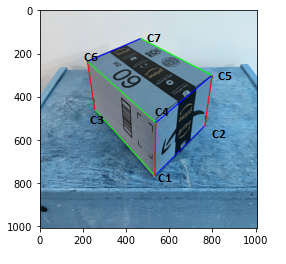
1. **Image Acquisition**

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]

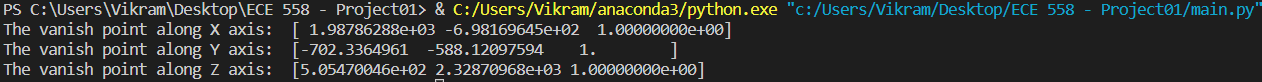
1. **Computing Vanish Points**

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.

where w = 1

Then, taking the cross products of the co-ordinates

Vanish Points Result:



1. **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.

Where

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:

Where

**Final Projection Matrix:**

**Homograph Matrix:**

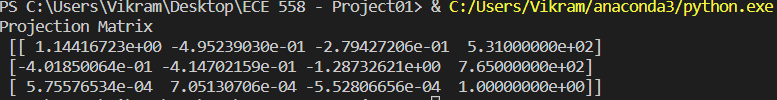
Homograph Matrix can simply be calculated with the required columns from Projection Matrix for each different plane.

XY plane:

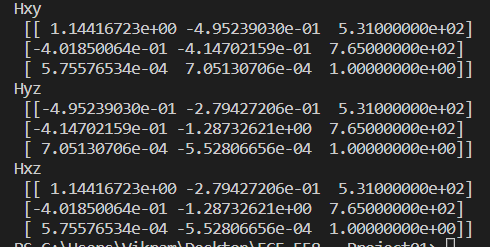
YZ plane:

XZ plane:

Projection Matrix Result:



Homograph Matrices Result:



1. **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

1. **Visualizing the 3D model**

Blender software was used to render the 3D image.



Rendered 3D Image of Box

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()