ECE 558 – Digital Imaging Systems

Project 1

Submitted by :

Surya Dutta (NCSU ID: 200481187)

Option 1

Project Description – This project focuses on an implementation of the paper "single view metrology" (Criminisi, Reid and Zisserman, ICCV99). It describes how aspects of the affine 3D geometry of a scene can be computed from a single perspective image with some prior knowledge.

Part 1 – Image Acquisition

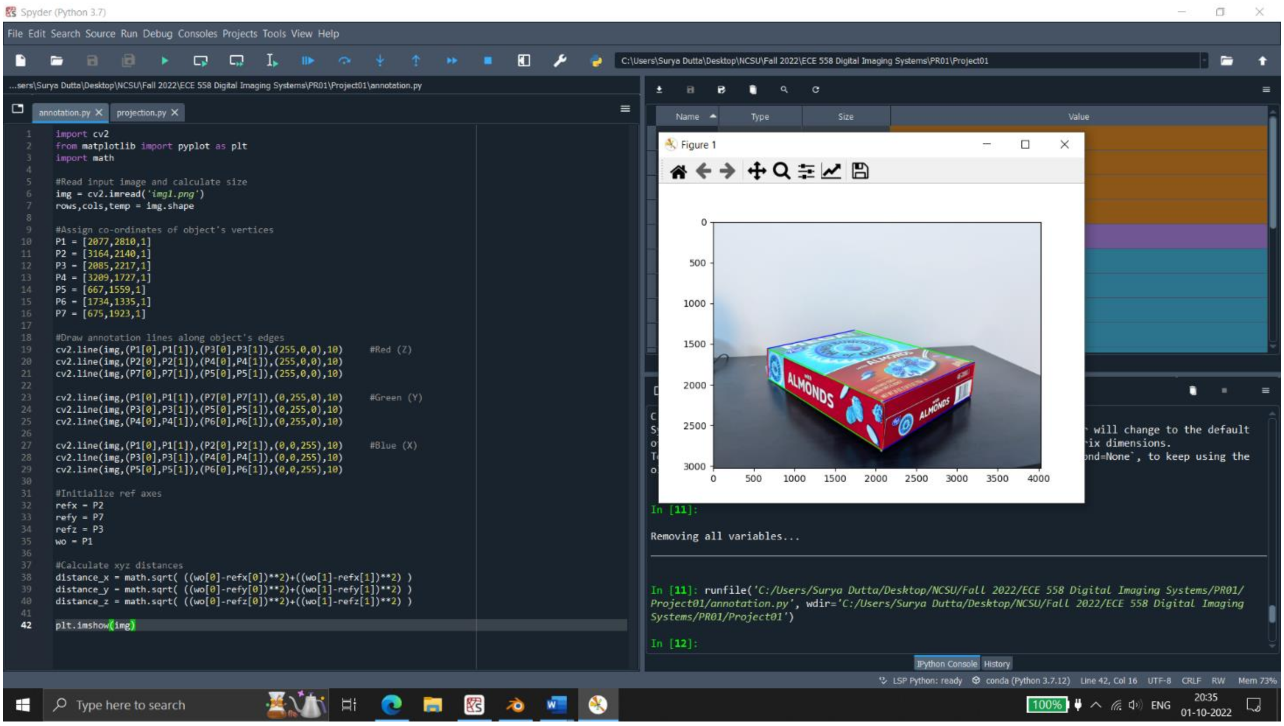
Original input image –

Part 2 – Annotation

Annotated Image -

Annotated Image

Screenshot of window showing the annotated image with the code in the background –



Code used for drawing projected lines along the edges (annotation) –

```
import cv2
from matplotlib import pyplot as plt
import math

#Read input image and calculate size
img = cv2.imread('img1.png')
rows,cols,temp = img.shape

#Assign co-ordinates of object's vertices
P1 = [2077,2810,1]
P2 = [3164,2140,1]
P3 = [2085,2217,1]
P4 = [3209,1727,1]
P5 = [667,1559,1]
P6 = [1734,1335,1]
P7 = [675,1923,1]

#Draw annotation lines along object's edges
cv2.line(img,(P1[0],P1[1]),(P3[0],P3[1]),(255,0,0),10)     #Red (Z)
cv2.line(img,(P2[0],P2[1]),(P4[0],P4[1]),(255,0,0),10)
cv2.line(img,(P7[0],P7[1]),(P5[0],P5[1]),(255,0,0),10)

cv2.line(img,(P1[0],P1[1]),(P7[0],P7[1]),(0,255,0),10)     #Green (Y)
cv2.line(img,(P3[0],P3[1]),(P5[0],P5[1]),(0,255,0),10)
cv2.line(img,(P4[0],P4[1]),(P6[0],P6[1]),(0,255,0),10)

cv2.line(img,(P1[0],P1[1]),(P2[0],P2[1]),(0,0,255),10)     #Blue (X)
cv2.line(img,(P3[0],P3[1]),(P4[0],P4[1]),(0,0,255),10)
cv2.line(img,(P5[0],P5[1]),(P6[0],P6[1]),(0,0,255),10)

#Initialize ref axes
refx = P2
refy = P7
```

```
refz = P3

wo = P1


#Calculate xyz distances

distance_x = math.sqrt( ((wo[0]-refx[0])**2)+((wo[1]-refx[1])**2) )

distance_y = math.sqrt( ((wo[0]-refy[0])**2)+((wo[1]-refy[1])**2) )

distance_z = math.sqrt( ((wo[0]-refz[0])**2)+((wo[1]-refz[1])**2) )


plt.imshow(img)
```

Part 3 – Computing Projection and Homograph Matrices

Screenshot of the code used -

```
#Compute vanishing pts
ax1,bx1,cx1 = np.cross(P1,P2)
ax2,bx2,cx2 = np.cross(P3,P4)
ay1,by1,cy1 = np.cross(P1,P7)
ay2,by2,cy2 = np.cross(P3,P5)
az1,bz1,cz1 = np.cross(P1,P3)
az2,bz2,cz2 = np.cross(P2,P4)

vx = np.cross([ax1,bx1,cx1],[ax2,bx2,cx2])
vy = np.cross([ay1,by1,cy1],[ay2,by2,cy2])
vz = np.cross([az1,bz1,cz1],[az2,bz2,cz2])
vx = np.array(vx/vx[2])
vy = np.array(vy/vy[2])
vz = np.array(vz/vz[2])

#Compute the projection matrices
ax,resid,rank,s = np.linalg.lstsq( (vx-refx).T , (refx - wo).T )
ay,resid,rank,s = np.linalg.lstsq( (vy-refy).T , (refy - wo).T )
az,resid,rank,s = np.linalg.lstsq( (vz-refz).T , (refz - wo).T )

px = (ax[0][0]/lengthx)*vx
py = (ay[0][0]/lengthy)*vy
pz = (az[0][0]/lengthz)*vz
pr = np.empty([3,4])
pr[:,0] = px
pr[:,1] = py
pr[:,2] = pz
pr[:,3] = wo

#Computing homograph matrices
#Initialize
hxy = np.zeros((3,3))
hyz = np.zeros((3,3))
hzx = np.zeros((3,3))
#Cols of P
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
```

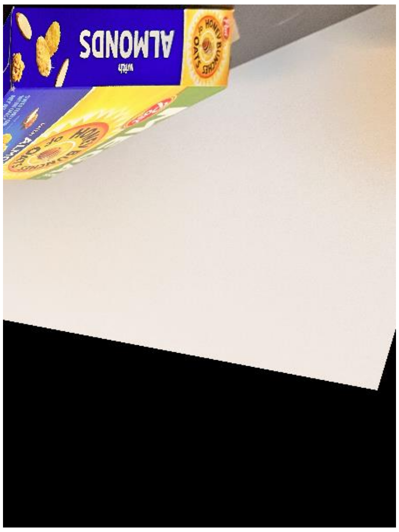
Part 4 – Computing Texture Maps –

Screenshot of the code used –

```
#Compute texture maps for XY, YZ & ZX
tmxy = cv2.warpPerspective(img,hxy,(rows,cols),flags=cv2.WARP_INVERSE_MAP)
tmyz = cv2.warpPerspective(img,hyz,(rows,cols),flags=cv2.WARP_INVERSE_MAP)
tmzx = cv2.warpPerspective(img,hzx,(rows,cols),flags=cv2.WARP_INVERSE_MAP)
cv2.imwrite("XY.png",tmxy)
cv2.imwrite("YZ.png",tmyz)
cv2.imwrite("ZX.png",tmzx)
```

Resulting texture maps –

XY



YZ

ZX

Cropped Texture Maps –

The texture maps are cropped using the snipping tool on Windows.

XY-cropped

YZ-cropped

ZX-cropped

Code used for computing projection, homographic matrices and texture maps –

```
import cv2
import numpy as np

#Read input img & convert to grayscale
img = cv2.imread('img1.png')
rows,cols,temp = img.shape
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

#Assign co-ordinates of object's vertices
P1 = [2077,2810,1]
P2 = [3164,2140,1]
P3 = [2085,2217,1]
P4 = [3209,1727,1]
P5 = [667,1559,1]
P6 = [1734,1335,1]
P7 = [675,1923,1]

#Initialize ref axes
wo = P1
refx = P2
refy = P7
refz = P3
refx = np.array([refx])
refy = np.array([refy])
refz = np.array([refz])
wo = np.array(wo)

#Calculate xyz distances
lengthx = np.sqrt(np.sum(np.square(refx - wo)))
lengthy = np.sqrt(np.sum(np.square(refy - wo)))
lengthz = np.sqrt(np.sum(np.square(refz - wo)))

#Compute vanishing pts
```

```
ax1,bx1,cx1 = np.cross(P1,P2)
ax2,bx2,cx2 = np.cross(P3,P4)
ay1,by1,cy1 = np.cross(P1,P7)
ay2,by2,cy2 = np.cross(P3,P5)
az1,bz1,cz1 = np.cross(P1,P3)
az2,bz2,cz2 = np.cross(P2,P4)

vx = np.cross([ax1,bx1,cx1],[ax2,bx2,cx2])
vy = np.cross([ay1,by1,cy1],[ay2,by2,cy2])
vz = np.cross([az1,bz1,cz1],[az2,bz2,cz2])
vx = np.array(vx/vx[2])
vy = np.array(vy/vy[2])
vz = np.array(vz/vz[2])

#Compute the projection matrices
ax,resid,rank,s = np.linalg.lstsq( (vx-refx).T , (refx - wo).T )
ay,resid,rank,s = np.linalg.lstsq( (vy-refy).T , (refy - wo).T )
az,resid,rank,s = np.linalg.lstsq( (vz-refz).T , (refz - wo).T )

px = (ax[0][0]/lengthx)*vx
py = (ay[0][0]/lengthy)*vy
pz = (az[0][0]/lengthz)*vz
pr = np.empty([3,4])
pr[:,0] = px
pr[:,1] = py
pr[:,2] = pz
pr[:,3] = wo

#Computing homograph matrices
#Initialize
hxy = np.zeros((3,3))
hyz = np.zeros((3,3))
hzx = np.zeros((3,3))
#Cols of P
```

```
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

#Compute texture maps for XY, YZ & ZX
tmxy = cv2.warpPerspective(img,hxy,(rows,cols),flags=cv2.WARP_INVERSE_MAP)
tmyz = cv2.warpPerspective(img,hyz,(rows,cols),flags=cv2.WARP_INVERSE_MAP)
tmzx = cv2.warpPerspective(img,hzx,(rows,cols),flags=cv2.WARP_INVERSE_MAP)
cv2.imwrite("XY.png",tmxy)
cv2.imwrite("YZ.png",tmyz)
cv2.imwrite("ZX.png",tmzx)
```

Part 5 - Visualizing the reconstructed 3D model

The 3D model is reconstructed and visualized using Blender.

Rendered 3D Model

References –

- Criminisi, A., Reid, I. and Zisserman, A., 2000. Single view metrology. International Journal of Computer Vision, 40(2), pp.123-148.
- https://github.com/bliuag/Single-View-Metrology/blob/master/SingleViewMetrologyFinalReport.pdf
- https://github.com/hgarud/Single_View_Metrology

****