NC STATE UNIVERSITY

ECE 558 Project 2

Implementation of Single view metrology: Automatic Annotation and estimation of Vanishing points

Nirmal Elamon
200201466

Aditya Rasam
200153631

Table of Contents

I. Introduction:

This project aims at building a 3D model of an object using a 2D image of the object. The procedure and the algorithm follow the Single View metrology paper by Criminisi. Further in this project we aim to detect the vanishing points automatically using Line Segment Detector and RANSAC.

Figure 1: 3-point perspective Image of a BOX

The input image is required to be 3-point perspective and accordingly the image was taken following the 3-point perspective image guide as mentioned in the project description document.

II. Computing the Vanishing points:

Parallel lines in real world intersect at the image plane. The point at which these lines meet in each X,Y and Z direction respectively can be estimated by extending the lines manually in these directions. Multiple horizontal vanishing points constitutes horizon or vanishing line. The vanishing line that results from lines in Z direction is in direction normal to the ground or reference plane. In this project, we computed the vanishing points using the method suggested by Prof. Robert. T Collins in project description document. For this we required parallel lines along X, Y and Z direction. To estimate these lines and vanishing points we made use of Line Segment Detector (LSD) and RANSAC algorithm. Using Line Segment Detector (LSD) we extracted an image that had multiple lines within it. These lines are nothing but the edges or straight-line regions in the image that saw a change in the intensities. LSD algorithm highlights such lines in an image.

However, for estimation of Vanishing points we need lines those are along X, Y and Z axis and whose best representation can be seen by the edges of the box. Hence to find the lines in the region near the edges of the box while ignoring the lines in the background we performed thresholding operation on the length of the lines detected by LSD. This removed all the unwanted small lines from the image. Now we ran a thresholding on the angles to get parallel lines on 3 different axis separately. We were left with multiple lines along the X, Y and Z direction which were not exactly parallel to each other but skewed by a certain angle. Finally, to get lines only in the desired direction we passed these line points to RANSAC algorithm, which is a kind of classifier. RANSAC consider the endpoints of multiple lines under consideration and returns a single line. In our case, it was used to estimate a single line along an edge from multiple lines. Thus, we obtained an image with lines as the edges of the box with endpoints being the corner points.

The data points obtained from RANSAC algorithm were passed to the programmed developed in Project 1 with the inputs as mentioned in sections further in this report

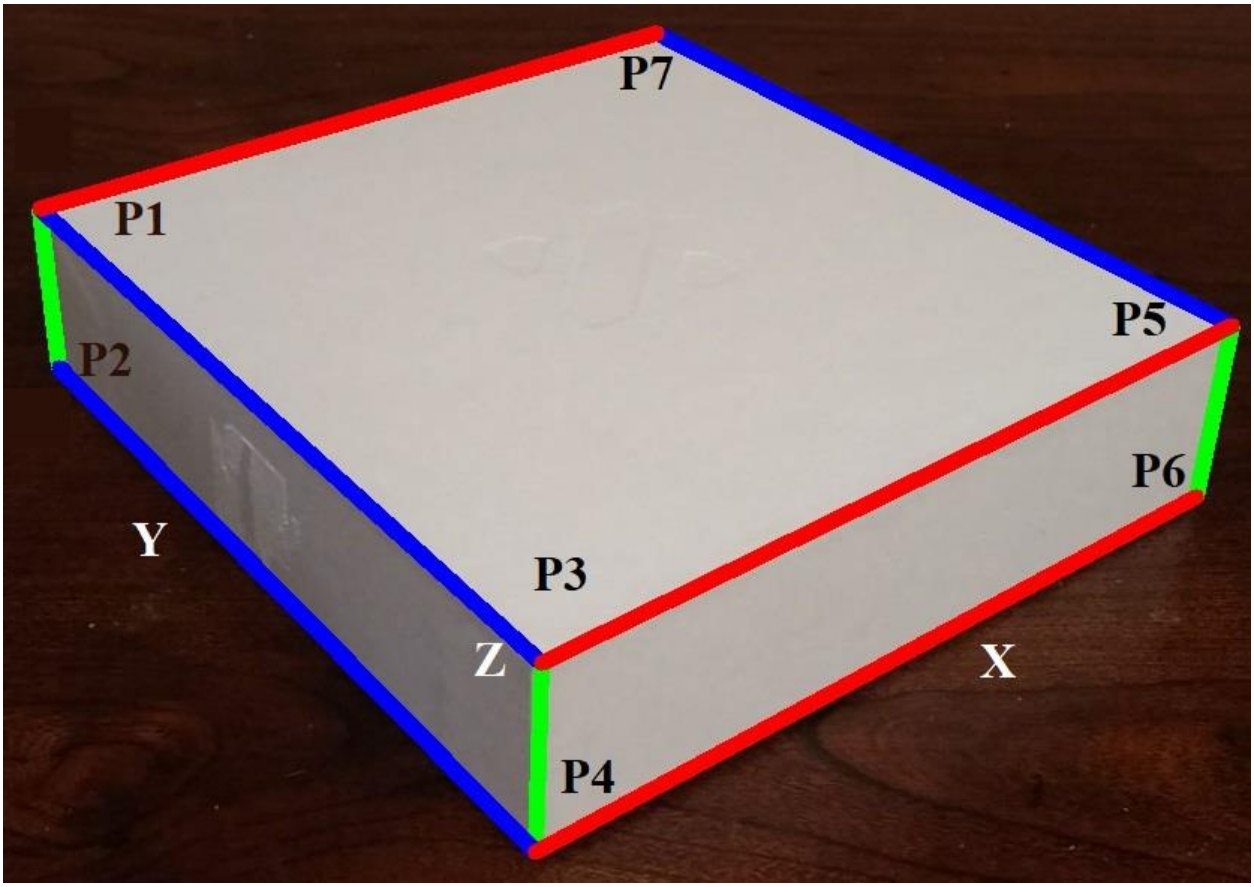


Figure 2: Vanishing Point Estimation

Procedure: In this project, we defined a function that calculates the vanishing lines co-ordinates automatically for the calculation of the vanishing point. Specifically, the endpoints for the lines on the

edges of the box along X, Y and Z direction are calculated automatically from the end points of the lines detected by LSD and RANSAC algorithm. These points are passed to the function that was developed in Project 1. So, in project 1 for the function to create a 3D model in all there are 7 points as shown in the image with X, Y and Z orientation. These points are converted to homogeneous co-ordinate within this function.

1. Each line is estimated by taking the cross product of its end-point.
    a. E.g. line 'L3' is estimated by taking cross product of vectors formed by its end-points P1 and P3
2. Now the vanishing point is estimated from the lines parallel in the required direction. This is done by taking the cross product of the two lines.
    a. E.g. vanishing point in x direction is calculated by taking cross product of line L5 and L6.

Thus vanishing points in X,Y and Z direction are computed using two lines in each direction. Following this the projection and homograph matrix are computed

III. Computing the projection matrix and homograph matrix

As computed by the code, the reference distance from the selected reference point is given in the following image. Based on the computed vanishing points, the projection matrix can be computed as follows.

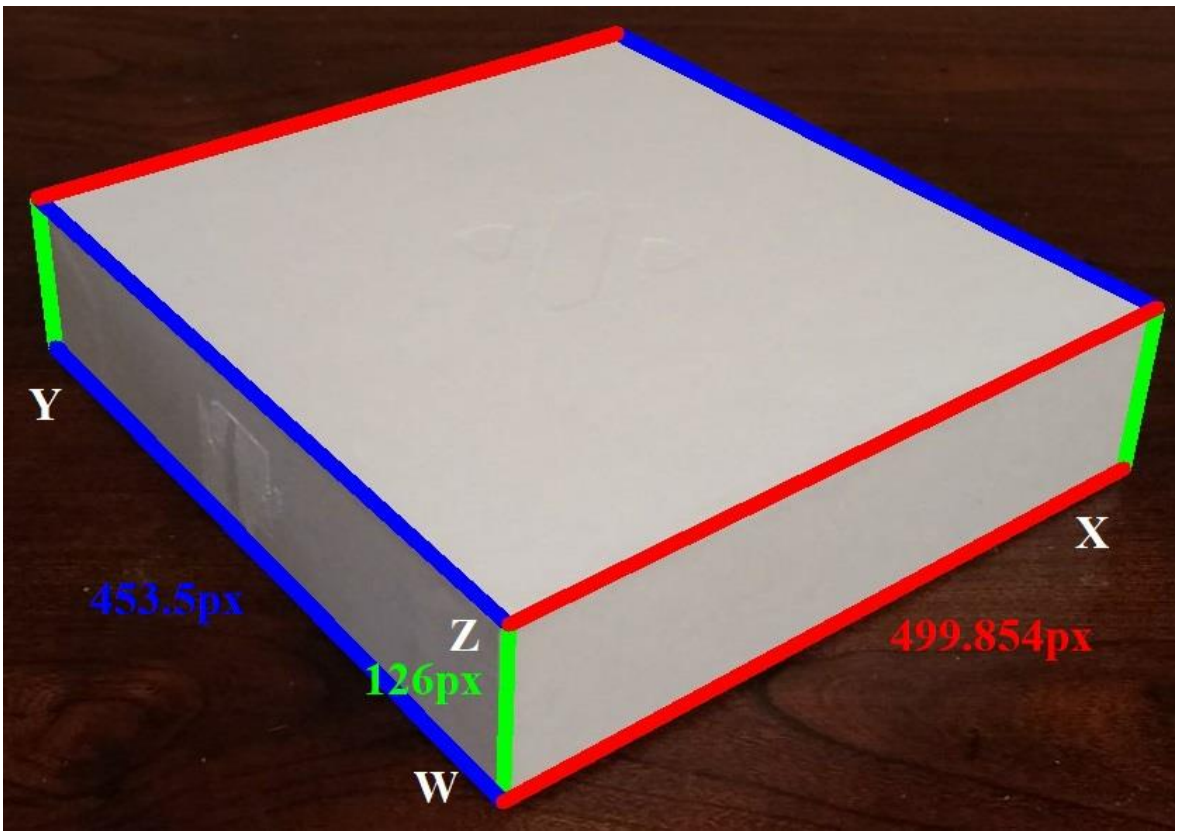


Figure 3: Reference distance

Computation of Projection Matrix:

The Projection matrix is given as

$$P = [p1\ p2\ p3\ p4]$$

Where,

Each column = Product of the vanishing point in that direction and the scaling factor α

$$p1 = \alpha_x * V_x \quad p2 = \alpha_y * V_y \quad p3 = \alpha_z * V_z \text{ and } p4 = \text{world origin}$$

Thus, the first three columns of projection matrix are known only to a certain scale and the fourth column is the world origin in the image co-ordinate. Hence, to find the projection matrix we need to know the vanishing points, scaling factors and the world origin in image co-ordinates.

Calculation of scaling factor:

To calculate the scaling factor I fixed the world co-ordinate and reference points in X,Y and Z directions. Also, their distances towards the origin was calculated. Finally, the scaling factor was calculated by solving the following equation in python

$$\alpha_x * \text{ref\_distanceX} * [Vx - X\_ref] = [X\_ref - Wo]$$

where

ref_distanceX is a scalar distance of reference point in X direction from world origin.

X_ref is the vector location of reference point in X direction

Vx is the vanishing point in X direction

Wo is the world origin in image co-ordinate

Similarly scaling factors α_y and α_z are calculated.

$$\alpha_y * \text{ref\_distanceY} * [Vy - Y\_ref] = [Y\_ref - Wo]$$

$$\alpha_z * \text{ref\_distanceZ} * [Vz - Z\_ref] = [Z\_ref - Wo]$$

This ends the calculation of projection matrix.

Computation of Homograph Matrix:

The homograph matrix is calculated from the projection matrix.

Hxy is the 1st,2nd and 4th column of Projection matrix.

Hyz is the 2nd,3rd and 4th column of Projection matrix.

Hxz is the 1st,3rd and 4th column of Projection matrix.

IV. Computing the texture maps for XY, YZ and XZ planes respectively

Using H_{xy}, H_{yz} and H_{xz} matrices, perspective transformation of the image was carried out to result in texture maps for XY, YZ and XZ as shown in the figure.

Texture maps

Figure 6: XY texture map

Figure 5: YZ texture map

Figure 4:XZ texture map

Cropped Images

Figure 8:XY Cropped

Figure 7:YZ Cropped

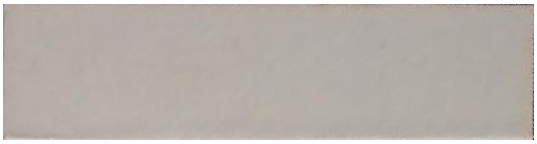


Figure 9: XZ cropped

VI. Reconstruction of 3D model:

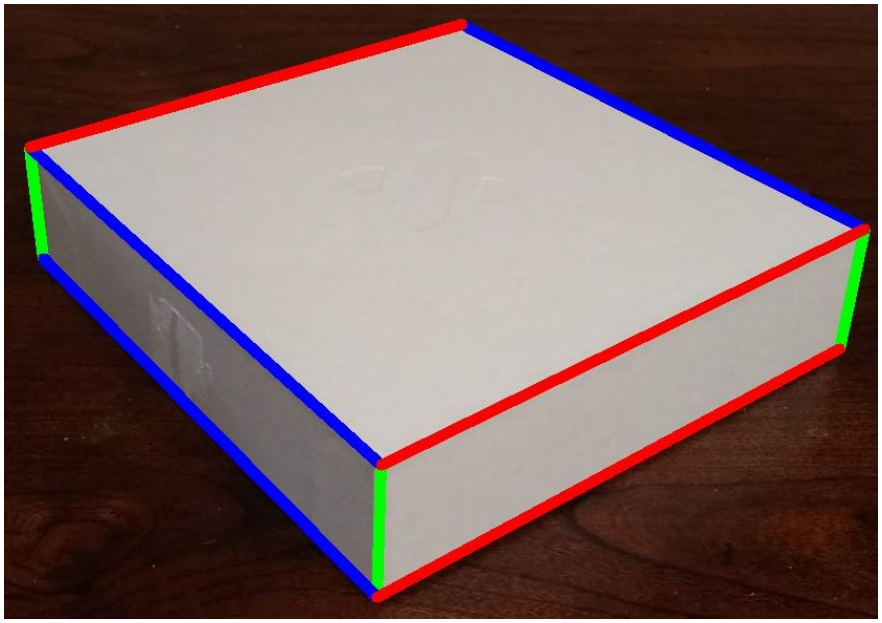
To reconstruct a 3D model, the obtained texture maps and the co-ordinates of the points in the respective plane was passed to a VRML script. This file was imported within a 3D reconstruction software like 'View 3D scene' to result in a 3D model as shown below.

Figure 11: 3D model

VII. Python Script for Single View Geometry.

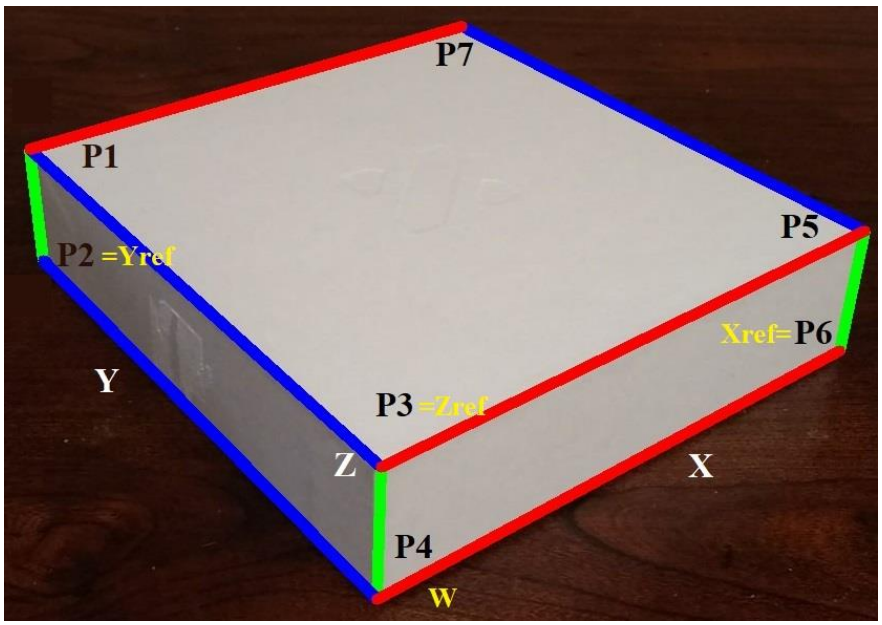
Function Description:

AutomaticAnnotation(Image): This function performs the task of annotating the edges of the box under consideration.



SingleViewGeom(P1, P2, P3, P4, P5, P6, P7, X_ref, Y_ref, Z_ref, W): The function creates the texture maps using a 3-point perspective image.

The location of the points (in the argument) on the box is given in the image.



The arguments for the coded function are automatically calculated by the function '**AutomaticAnnotation**' mentioned above0. The function '**AutomaticAnnotation**' calls the **SingleViewGeom** functions internally and passes following values to it.

Input	Data Type	Description	Example
Image	String	Name of Image file with format	'testbox0.jpg'
P1	Tuple	Pixel Co-ordinate 1	(24.32,136.55) px
P2	Tuple	Pixel Co-ordinate 2	(37.369,242.672) px
P3	Tuple	Pixel Co-ordinate 3	(359.7,439.6) px
P4	Tuple	Pixel Co-ordinate 4	(355.8,565.5) px
P5	Tuple	Pixel Co-ordinate 5	(820.9,241.68) px
P6	Tuple	Pixel Co-ordinate 6	(796,328.66) px
P7	Tuple	Pixel Co-ordinate 7	(436.5,49.19) px
X_ref	Tuple	Reference co-ordinate in X direction	P6
Y_ref	Tuple	Reference co-ordinate in Y direction	P2
Z_ref	Tuple	Reference co-ordinate in Z direction	P3
W	Tuple	World origin	P4

Instructions to run the script.

1. Then the user needs to run the script once.
2. The user is required to call the function '**AutomaticAnnotation**' and pass the image or the path to it as the argument to display the output images for XY, YZ and XZ texture maps.

Python Script

```
import cv2

import numpy as np

from sklearn import linear_model

from sklearn.cluster import KMeans

from matplotlib import pyplot as plt

from sklearn import linear_model



from scipy.spatial import distance

import math
```

```
#=============================================================================


def SingleViewGeom(Image, P1, P2, P3, P4, P5, P6, P7, X_ref, Y_ref, Z_ref, W):
    import cv2
    import numpy as np
    import numpy.linalg as lin
    from scipy.spatial import distance

    #from matplotlib import pyplot



    #p =  'C:\Users\adity\Documents\NCSU\Proj1\testbox.jpeg'#'Desktop\\testbox.jpeg'
    img = cv2.imread(Image)
    S = img.shape
    #print('s', S)




    P1= np.array(P1)
    P2= np.array(P2)
    P3= np.array(P3)
    P4= np.array(P4)
    P5= np.array(P5)
```

```
P6= np.array(P6)

P7 = np.array(P7)



w=1



#Line 1 P1 & P2

temp = [P1[0],P1[1],w]

e1_1 = np.array(temp)

temp = [P2[0],P2[1],w]

e2_1 = np.array(temp)

temp  = np.cross(e1_1, e2_1)

l1 = np.array(temp)



#Line 2 P1 & P2

temp = [P3[0],P3[1],w]

e1_2 = np.array(temp )

temp = [P4[0],P4[1],w]

e2_2 = np.array(temp)


temp  = np.cross(e1_2, e2_2)

l2 = np.array(temp)


V1 =  np.array(np.cross(l1,l2))
```

```
#Line 3 P1 & P3
temp = [P1[0],P1[1],w]
e1_3 = np.array(temp )
temp = [P3[0],P3[1],w]
e2_3 = np.array(temp)

temp  = np.cross(e1_3, e2_3)
l3 = np.array(temp)


#Line 4 P2 & P4
temp = [P2[0],P2[1],w]
e1_4 = np.array(temp )
temp = [P4[0],P4[1],w]
e2_4 = np.array(temp)


temp  = np.cross(e1_4, e2_4)
l4 = np.array(temp)

V2 =  np.array(np.cross(l3,l4))


 #Line 5 P3 & P5
temp = [P3[0],P3[1],w]
e1_5 = np.array(temp )
temp = [P5[0],P5[1],w]
e2_5 = np.array(temp)
```

```
temp  = np.cross(e1_5, e2_5)
l5 = np.array(temp)

#Line 6 P4 & P6
temp = [P4[0],P4[1],w]
e1_6 = np.array(temp )
temp = [P6[0],P6[1],w]
e2_6 = np.array(temp)



temp  = np.cross(e1_6, e2_6)
l6 = np.array(temp)
V3 = np.array(np.cross(l5,l6))



tV1 = V1[:]/V1[2]

tV2 = V2[:]/V2[2]
tV3 = V3[:]/V3[2]

#print('V1', V1)
#print('V2', V2)
```

```
#print('V3', V3)


#Vanishing points
Vy = np.array([tV2]).T
Vx = np.array([tV3]).T
Vz = np.array([tV1]).T


# print('tV1', Vx)
# print('tV2', Vy)
# print('tV3', Vz)


#World Origin in image-cords
WO = np.array(W)
WO = np.append([WO],[1])
WO = np.array([WO]).T
#Reference axis-cords in im-cords
ref_x = np.array(X_ref)#[ 197 , 317 , 1 ]
ref_x = np.append([ref_x],[1])
x_ref =  np.array([ref_x]).T


ref_y = np.array(Y_ref)#[ 442 , 317 , 1 ]
ref_y = np.append([ref_y],[1])
y_ref =  np.array([ref_y]).T



ref_z = np.array(Z_ref)#[ 319 , 227 , 1 ] #556  [ 778 , 400 , 1 ]
ref_z = np.append([ref_z],[1])
z_ref =  np.array([ref_z]).T
```

```
ref_x_dis = distance.euclidean(x_ref,WO)
ref_y_dis = distance.euclidean(y_ref,WO)
ref_z_dis = distance.euclidean(z_ref,WO)

#print('Ref_distance_X',ref_x_dis)
#print('Ref_distance_Y',ref_y_dis)
#print('Ref_distance_Z',ref_z_dis)


#%% Scaling factors of the projection matrix
temp = np.array((x_ref - WO))
tempx,resid,rank,s = np.linalg.lstsq((Vx-x_ref),temp)
a_x = (tempx )  / ref_x_dis  #%( A \ B ==> left division )


temp = np.array((y_ref - WO))
tempy,resid,rank,s = np.linalg.lstsq((Vy-y_ref),temp)
a_y = (tempy )  / ref_y_dis  #%( A \ B ==> left division )

temp = np.array((z_ref - WO))
tempz,resid,rank,s = np.linalg.lstsq((Vz-z_ref),temp)
a_z = (tempz )  / ref_z_dis  #%( A \ B ==> left division )

p1 = Vx*a_x
p2 = Vy*a_y
p3 = Vz*a_z
p4 = np.array(WO)
```

```
P = np.concatenate((p1, p2, p3, p4), axis =1)



Hxy = np.concatenate((p1, p2, p4), axis =1)
Hyz = np.concatenate((p2, p3, p4), axis =1)
Hzx = np.concatenate((p1, p3, p4), axis =1)



warp = cv2.warpPerspective(img, Hxy , (S[0]*5, S[1]*5), flags = cv2.WARP_INVERSE_MAP)
warp1 = cv2.warpPerspective(img, Hyz , (S[0]*5, S[1]*5), flags = cv2.WARP_INVERSE_MAP)
warp2 = cv2.warpPerspective(img, Hzx , (S[0], S[1]), flags = cv2.WARP_INVERSE_MAP)

img_ann=cv2.line(img ,(P1[0],P1[1]),(P2[0],P2[1]), (0,255,0), 10)
img_ann=cv2.line(img_ann ,(P3[0],P3[1]),(P4[0],P4[1]), (0,255,0), 10)
img_ann=cv2.line(img_ann ,(P5[0],P5[1]),(P6[0],P6[1]), (0,255,0), 10)

img_ann=cv2.line(img_ann ,(P1[0],P1[1]),(P3[0],P3[1]), (255,0,0), 10)
img_ann=cv2.line(img_ann ,(P2[0],P2[1]),(P4[0],P4[1]), (255,0,0), 10)
img_ann=cv2.line(img_ann ,(P7[0],P7[1]),(P5[0],P5[1]), (255,0,0), 10)

img_ann=cv2.line(img_ann ,(P6[0],P6[1]),(P4[0],P4[1]), (0,0,255), 10)
img_ann=cv2.line(img_ann ,(P3[0],P3[1]),(P5[0],P5[1]), (0,0,255), 10)
img_ann=cv2.line(img_ann ,(P1[0],P1[1]),(P7[0],P7[1]), (0,0,255), 10)
```

```
    cv2.imshow('Ann', img_ann)
    cv2.imwrite('Ann.jpg', img_ann)
    cv2.imshow('XY', warp)
    cv2.imwrite('XY.jpg', warp)
    cv2.imshow('YZ', warp1)
    cv2.imwrite('YZ.jpg', warp1)
    cv2.imshow('ZX', warp2)
    cv2.imwrite('ZX.jpg', warp2)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

    return ref_x_dis, ref_y_dis, ref_z_dis

#================================================================================
def AutomaticAnnotation(Image):



    img1 = cv2.imread(Image,0)

    img1 =np.array(img1)



    #Create default parametrization LSD
    lsd = cv2.createLineSegmentDetector(0)
```

```
#Detect lines in the image
lines = lsd.detect(img1)[0] #Position 0 of the returned tuple are the detected lines

#lines = np.array(lines)
#Draw detected lines in the image
drawn_img1 = lsd.drawSegments(img1,lines)



l = len(lines)
shape = lines.shape
p1 = [0,0]#np.zeros
p2 = [0,0]#np.zeros


d1 = np.zeros(l)
#LN = np.zeros((l,1,4))
#LN = np.array(LN)
LN = np.zeros_like(lines)
#LN = np.zeros(shape)

L = np.array(lines)
threshold = 80


p1 = np.array(p1)
p2 = np.array(p2)
d1 = np.array(d1)
```

```
i=0

k=0

for i in range (0,(l-1)):


    for j in range(0,4):
        if j == 0:
            p1[0] = L[i,0,j]
        elif j == 1:
            p1[1] = L[i,0,j]
        elif j == 2:
            p2[0] = L[i,0,j]
        elif j == 3:
            p2[1] = L[i,0,j]



    d1[i]  = distance.euclidean(p1,p2)


    if d1[i] > threshold:
        #LN[k,0] = L[i,0]
        k=k+1


LN = np.resize(LN,(k,1,4))


#LN = np.empty((k,1,4))

#LN = np.zeros()


#LN = np.array(LN)

#LN = np.asanyarray(LN)
```

```
k=0
for i in range (0,(l-1)):

    for j in range(0,4):
        if j == 0:
            p1[0] = L[i,0,j]
        elif j == 1:
            p1[1] = L[i,0,j]
        elif j == 2:
            p2[0] = L[i,0,j]
        elif j == 3:
            p2[1] = L[i,0,j]



    d1[i]  = distance.euclidean(p1,p2)

    if d1[i] > threshold:

        LN[k,0] = lines[i,0]
        k=k+1

'=========================Angle thresholding=================================='

LN = np.array(LN)

lines_new = LN
LN_x = np.zeros_like(LN)
LN_y = np.zeros_like(LN)
```

```
LN_z = np.zeros_like(LN)


a = []
x1_a = []
x2_a=[]
y1_a=[]
y2_a=[]
i=0
q=0
n_x = 0
n_y = 0
n_z = 0


l = len(LN)
for i in range (0,(l)):
    #q=q+1
    #k=k+1
    x1 = LN[i,0,0]
    x1_a.append(x1)

    y1 = LN[i,0,1]
    y1_a.append(y1)

    x2 = LN[i,0,2]
    x2_a.append(x2)

    y2 = LN[i,0,3]
    y2_a.append(y2)
```

```
angle = math.atan2((y2-y1),(x2-x1))

angle = math.degrees(angle)

angle =  angle%360

if angle > 180:
    angle = angle -180



a.append(angle)



if (angle > 75) & (angle < 105) :

    LN_z[n_z,0] = LN[i,0]
    n_z = n_z + 1

if (angle > 25) & (angle < 50):

    LN_y[n_y,0] = LN[i,0]
    n_y = n_y + 1

if (angle > 150) & (angle < 165):

    LN_x[n_x,0] = LN[i,0]
    n_x = n_x + 1
```

```
#LN_x =
#LN = np.resize(LN,(k,1,4))



LN_x = np.resize(LN_x,(n_x,1,4))
LN_y = np.resize(LN_y,(n_y,1,4))
LN_z = np.resize(LN_z,(n_z,1,4))
print(LN_x[0][0][0])

#LN_Z = np.array(LN_z)




[n1_x,n2_x,n3_x]=np.shape(LN_x)
k_means_x=[[]for i in range(2*n1_x)]
p=0
for i in range(n1_x):
    k_means_x[p].append(LN_x[i][0][0])
    k_means_x[p].append(LN_x[i][0][1])
    p=p+1
    k_means_x[p].append(LN_x[i][0][2])
    k_means_x[p].append(LN_x[i][0][3])
    p=p+1



[n1_y,n2_y,n3_y]=np.shape(LN_y)
k_means_y=[[]for i in range(2*n1_y)]
```

```
p=0
for i in range(n1_y):
    k_means_y[p].append(LN_y[i][0][0])
    k_means_y[p].append(LN_y[i][0][1])
    p=p+1
    k_means_y[p].append(LN_y[i][0][2])
    k_means_y[p].append(LN_y[i][0][3])
    p=p+1



[n1_z,n2_z,n3_z]=np.shape(LN_z)
k_means_z=[[]for i in range(2*n1_z)]
p=0
for i in range(n1_z):
    k_means_z[p].append(LN_z[i][0][0])
    k_means_z[p].append(LN_z[i][0][1])
    p=p+1
    k_means_z[p].append(LN_z[i][0][2])
    k_means_z[p].append(LN_z[i][0][3])
    p=p+1
kmeans = KMeans(n_clusters=3, random_state=0).fit(k_means_z)
labels_z=kmeans.labels_

kmeans = KMeans(n_clusters=3, random_state=0).fit(k_means_y)
labels_y=kmeans.labels_



modified_img1 = lsd.drawSegments(img1,LN)
```

```
#Show image
drawn_img1 = cv2.resize(drawn_img1, (1000,1000),1,1, cv2.INTER_AREA)

modified_img1 = cv2.resize(modified_img1, (1000,1000),1,1, cv2.INTER_AREA)

modified_img1_x = lsd.drawSegments(img1,LN_x)
modified_img1_y = lsd.drawSegments(img1,LN_y)
modified_img1_z = lsd.drawSegments(img1,LN_z)

cv2.imshow("LSD",drawn_img1)
cv2.imwrite("LSD.jpg",drawn_img1)
cv2.imshow("Edges",modified_img1)
#cv2.imshow("Mod_LSD_X",modified_img1_x)
#cv2.imshow("Mod_LSD_Y",modified_img1_y)
#cv2.imshow("Mod_LSD_Z",modified_img1_z)

Zref_X_x = LN_x[1,0,0]
Zref_X_y = LN_x[1,0,1]

Zref_Y_x = LN_y[1,0,2]
```

```
Zref_Y_y = LN_y[1,0,3]


Zref_Z_x = LN_z[2,0,0]
Zref_Z_y = LN_z[2,0,1]

Zrefx = np.array([Zref_X_x,Zref_Y_x,Zref_Z_x])
Zrefy = np.array([Zref_X_y,Zref_Y_y,Zref_Z_y])


Zrefx = np.average(Zrefx)

Zrefy = np.average(Zrefy)


Xref_X_x = LN_x[2,0,2]
Xref_X_y = LN_x[2,0,3]


#Xref_Y_x = LN_y[1,0,2]
#Xref_Y_y = LN_y[1,0,3]


Xref_Z_x = LN_z[1,0,0]
Xref_Z_y = LN_z[1,0,1]

Xrefx = np.array([Xref_X_x,Xref_Z_x])
Xrefy = np.array([Xref_X_y,Xref_Z_y])
```

```
Xrefx = np.average(Xrefx)

Xrefy = np.average(Xrefy)

#Yref_X_x = LN_x[0,0,2]
#Yref_X_y = LN_x[0,0,3]


Yref_Y_x = LN_y[2,0,0]
Yref_Y_y = LN_y[2,0,1]


Yref_Z_x = LN_z[0,0,2]
Yref_Z_y = LN_z[0,0,3]

Yrefx = np.array([Yref_Y_x,Yref_Y_x])
Yrefy = np.array([Yref_Y_y,Yref_Y_y])


Yrefx = np.average(Yrefx)

Yrefy = np.average(Yrefy)

W_X_x = LN_x[2,0,0]
W_X_y = LN_x[2,0,1]


W_Y_x = LN_y[2,0,2]
```

```
W_Y_y = LN_y[2,0,3]


W_Z_x = LN_z[2,0,2]
W_Z_y = LN_z[2,0,3]

Wx = np.array([W_X_x,W_Y_x,W_Z_x])
Wy = np.array([W_X_y,W_Y_y,W_Z_y])


Wx = np.average(Wx)

Wy = np.average(Wy)

P1_X_x = LN_x[0,0,2]
P1_X_y = LN_x[0,0,3]


P1_Y_x = LN_y[1,0,0]
P1_Y_y = LN_y[1,0,1]


P1_Z_x = LN_z[0,0,0]
P1_Z_y = LN_z[0,0,1]

P1x = np.array([P1_X_x,P1_Y_x,P1_Z_x])
P1y = np.array([P1_X_y,P1_Y_y,P1_Z_y])

P1x = np.average(P1x)
```

```
P1y = np.average(P1y)

P5_X_x = LN_x[1,0,2]
P5_X_y = LN_x[1,0,3]


P5_Y_x = LN_y[0,0,0]
P5_Y_y = LN_y[0,0,1]


P5_Z_x = LN_z[1,0,2]
P5_Z_y = LN_z[1,0,3]

P5x = np.array([P5_X_x,P5_Y_x,P5_Z_x])
P5y = np.array([P5_X_y,P5_Y_y,P5_Z_y])


P5x = np.average(P5x)

P5y = np.average(P5y)

P7_X_x = LN_x[0,0,0]
P7_X_y = LN_x[0,0,1]


P7_Y_x = LN_y[0,0,2]
P7_Y_y = LN_y[0,0,3]
```

```
#P7_Z_x = LN_z[0,0,0]

#P7_Z_y = LN_z[0,0,1]


P7x = np.array([P7_X_x,P7_Y_x])

P7y = np.array([P7_X_y,P7_Y_y])



P7x = np.average(P7x)


P7y = np.average(P7y)


X_ref = [Xrefx, Xrefy]

Y_ref = [Yrefx, Yrefy]

Z_ref = [Zrefx, Zrefy]

P1 = [P1x,P1y]

P5 = [P5x,P5y]

P7 = [P7x,P7y]

W = [Wx,Wy]



#plt.imshow(drawn_img)

#plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis

#plt.show()

'''

# Robustly fit linear model with RANSAC algorithm

ransac = linear_model.RANSACRegressor()

ransac.fit(X, y)

inlier_mask = ransac.inlier_mask_
```

```
    outlier_mask = np.logical_not(inlier_mask)


    '''


    #P1
    P2 = Y_ref
    P3 = Z_ref
    P4 = W
    #P5 =
    P6 = X_ref
    #P7 =



    ref_dis_x, ref_dis_y, ref_dis_z = SingleViewGeom(Image, P1, P2, P3, P4, P5, P6, P7, X_ref, Y_ref, Z_ref,
W)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


    return


#path                                                                                                    =
'C:\\Users\\adity\\Documents\\NCSU\\1ECE558\\Proj2\\arasam_project01(1)\\arasam_project01\\aras
am_code\\testbox.jpg'
#Image = cv2.imread(path)
#Image = path


#AutomaticAnnotation('testbox.jpg')
```

VRML script

```
#VRML V2.0 utf8

Collision {
    collide FALSE
    children [
 Shape {
       appearance Appearance {
             texture ImageTexture {
                   url "XY.jpg"
             }
       }
       geometry IndexedFaceSet {
             coord Coordinate {
                   point [
                         0.000000 0.000000 125.980000,
                         499.8540000 0.000000 125.980000,
                         499.8540000 453.490000 125.980000,
                         0.000000 453.490000 125.980000,
                   ]
             }
             coordIndex [
                   0, 1, 2, 3, -1,
             ]
             texCoord TextureCoordinate {
                   point [
                         -0.000000 0.000000,
                         1.000000 0.000000,
                         1.000000 1.000000,
                         -0.000000 1.000000,
                   ]
             }
             texCoordIndex [
                   0, 1, 2, 3, -1,
             ]
             solid FALSE
       }
}
 Shape {
       appearance Appearance {
             texture ImageTexture {
                   url "ZX.jpg"
             }
       }
       geometry IndexedFaceSet {
             coord Coordinate {
                   point [
                         0.000000 0.000000 0.000000,
                         499.8540000 0.000000 0.000000,
                         499.8540000 0.000000 125.980000,
                         0.000000 0.000000 125.980000,
                   ]
             }
```

```
            coordIndex [
                  0, 1, 2, 3, -1,
            ]
            texCoord TextureCoordinate {
                  point [
                        -0.000000 0.000000,
                        1.000000 0.000000,
                        1.000000 1.000000,
                        -0.000000 1.000000,
                  ]
            }
            texCoordIndex [
                  0, 1, 2, 3, -1,
            ]
            solid FALSE
      }
}
 Shape {
      appearance Appearance {
            texture ImageTexture {
                  url "YZ.jpg"
            }
      }
      geometry IndexedFaceSet {
            coord Coordinate {
                  point [
                        0.000000 453.490000 0.000000,
                        0.000000 0.000000 0.000000,
                        0.000000 0.000000 125.980000,
                        0.000000 453.490000 125.980000,
                  ]
            }
            coordIndex [
                  0, 1, 2, 3, -1,
            ]
            texCoord TextureCoordinate {
                  point [
                        -0.000000 0.000000,
                        1.000000 0.000000,
                        1.000000 1.000000,
                        -0.000000 1.000000,
                  ]
            }
            texCoordIndex [
                  0, 1, 2, 3, -1,
            ]
            solid FALSE
      }
}
     ]
}
```

VIII. Team member Contribution

The team consisted of two members Aditya Rasam and Nirmal Elamon. We worked parallely over all part of the project. Hence there is no specific part to mention over which each one of us contributed.