PROJECT 1
SINGLE VIEW METEROLOGY

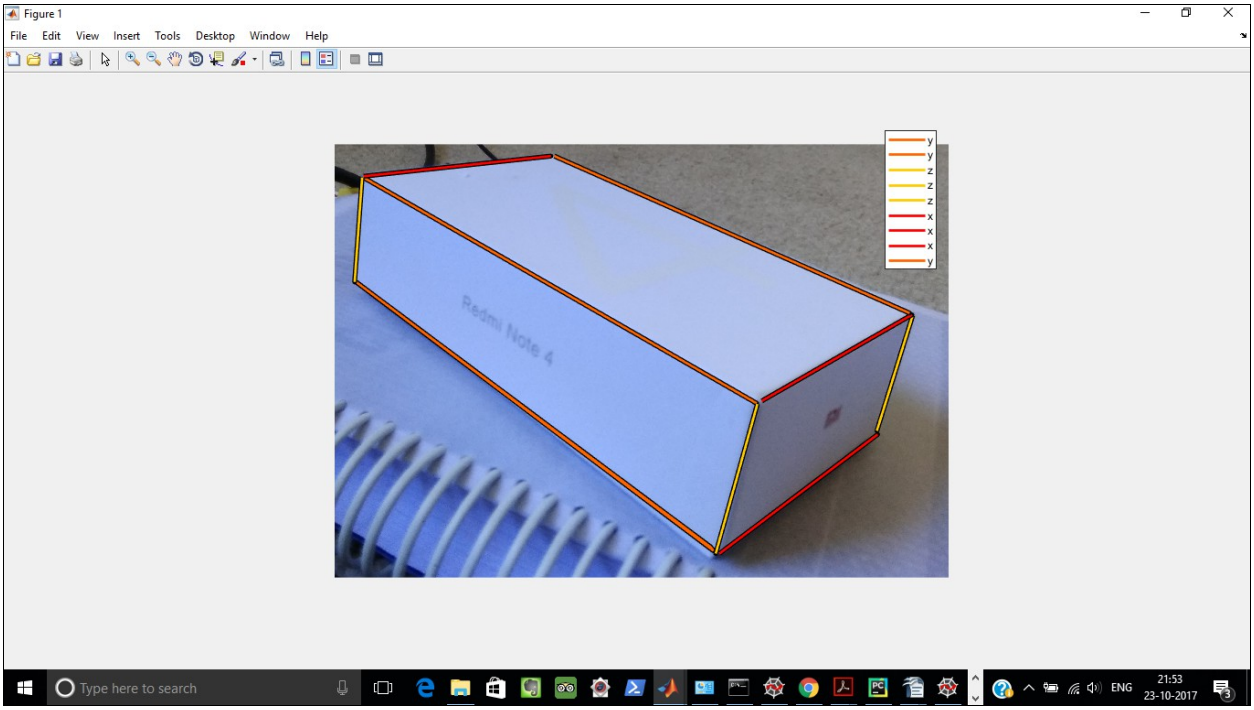
Team Members:

1. Kalyan Ghosh (UID: kghosh)
2. Akshay Kalkunte Suresh (UID: akalkun)
3. Bhargav Ram (UID: bmysore)

Steps followed to complete the project:

1. In the 1st step of the project, we took an image of a 3D box by following the **3 point perspective** imaging methodology. The picture of the box is as given below.

2. After that we used a simple annotation online tool **(LABEL ME)** to annotate the image. The annotated image is as given below.



3. The input to the below code(**SVM_Code.py**) is taken from the input file (**input.txt**) which is show below and also attached in the zipped folder.

```
y1_edge_1:124 137 1
y1_edge_2:1808 1109 1
y2_edge_1:86 588 1
y2_edge_2:1630 1747 1
y3_edge_1:940 51 1
y3_edge_2:2459 721 1
x1_edge_1:1808 1109 1
x1_edge_2:2459 721 1
x2_edge_1:1630 1747 1
x2_edge_2:2319 1223 1
x3_edge_1:124 137 1
```

```
x3_edge_2:940 51 1
z1_edge_1:1808 1109 1
z1_edge_2:1630 1747 1
z2_edge_1:2459 721 1
z2_edge_2:2319 1223 1
z3_edge_1:124 137 1
z3_edge_2:86 588 1
origin:1646 1747 1
reference_y:86 588 1
reference_x:2322 1239 1
reference_z:1808 1109 1
```

4. We then wrote the code to calculate the vanishing points, the scaling factors to generate the homography matrices and to apply the homography matrices to transform the planes. The complete code(**SVM_Code.py**) is given below and also attached in the zipped folder.

Code:

```
import numpy as np

from numpy import linalg as LA
import scipy.sparse.linalg as sla
import argparse

import cv2

def image_view(img_name):
    # Read the source 2D image
    img=cv2.imread(img_name)
    cv2.namedWindow('image',cv2.WINDOW_NORMAL)
    cv2.resizeWindow('image', 600,600)

    cv2.imshow('image',img)
    cv2.waitKey(0)
    return img

def compute_vanishing_points(line_eqs, axes):
    # Compute the 3 vanishing points.
    for axes_idx in range(0,axes_n):
        M = np.zeros((3,3))
        # Compute 2nd order moments
        for moment_idx in range(axes_idx*3, (axes_idx*3)+3):
            eqs_sh = np.reshape(np.array(line_eqs[moment_idx]), (1, 3))
            M = M + np.transpose(eqs_sh).dot(eqs_sh)

        # Perform eigen decomposition and extract the eigen vector with the
smallest eigen value. This corresponds to the vanishing point
        eigenValues, eigenVectors = sla.eigs(M, k=1, which='SM')
        eigenVectors = np.transpose(eigenVectors.real)
        # Convert coordinates into homogeneous form
        V[axes_idx]=(eigenVectors[-1]/eigenVectors[-1,-1])
```

```
    VY=np.array((V[0][:]))
    VX=np.array((V[1][:]))
    VZ=np.array((V[2][:]))

    return VX,VY,VZ

def compute_scale(V,ref_point,ref_len, world_coord):
    alpha = np.linalg.lstsq( (V-ref_point), (ref_point-world_coord))
[0]/ref_len
    return alpha

def compute_homography(P,plane):

    if plane=='xy':
        H=P[:,[0,1,3]]
    elif plane=='yz':
        H=P[:,[1,2,3]]
    elif plane=='xz':
        H=P[:,[0,2,3]]
    else:
        print("Invalid Plane")

    return H

def generate_edges(info_dict):
    # y-direction in front and top of image
    edges={}
    edges['y_f_t']={}
    edges['y_f_t'][0] = info_dict['y1_edge_1']
    edges['y_f_t'][1] = info_dict['y1_edge_2']

    # y-direction in front and bottom of image
    edges['y_f_b']={}
    edges['y_f_b'][0] = info_dict['y2_edge_1']
    edges['y_f_b'][1] = info_dict['y2_edge_2']

    # y-direction in front and bottom of image
    edges['y_b_t']={}
    edges['y_b_t'][0] = info_dict['y3_edge_1']
    edges['y_b_t'][1] = info_dict['y3_edge_2']

    # x-direction in front and top of image
    edges['x_f_t']={}
    edges['x_f_t'][0] = info_dict['x1_edge_1']
    edges['x_f_t'][1] = info_dict['x1_edge_2']

    # x-direction in front and bottom of image
    edges['x_f_b']={}
    edges['x_f_b'][0] = info_dict['x2_edge_1']
    edges['x_f_b'][1] = info_dict['x2_edge_2']

    # x-direction in back and top of image
    edges['x_b_t']={}
    edges['x_b_t'][0] = info_dict['x3_edge_1']
    edges['x_b_t'][1] = info_dict['x3_edge_2']
```

```
    # z-direction in front and right of image
    edges['z_f_r']={}
    edges['z_f_r'][0] = info_dict['z1_edge_1']
    edges['z_f_r'][1] = info_dict['z1_edge_2']

    # z-direction in back and right of image
    edges['z_b_r']={}
    edges['z_b_r'][0] = info_dict['z2_edge_1']
    edges['z_b_r'][1] = info_dict['z2_edge_2']

    # z-direction in front and left of image
    edges['z_f_l']={}
    edges['z_f_l'][0] = info_dict['z3_edge_1']
    edges['z_f_l'][1] = info_dict['z3_edge_2']

    return edges

def transform_map(src,H,plane):
    # Apply perspective transform
    img_dst = cv2.warpPerspective(src, H, (src.shape[0]*8, src.shape[1]*8),
flags=cv2.WARP_INVERSE_MAP)

    # Crop image and obtain feature maps
    gray = cv2.cvtColor(img_dst,cv2.COLOR_BGR2GRAY)
    _,thresh = cv2.threshold(gray,1,255,cv2.THRESH_BINARY)
    _,contours,_ =
cv2.findContours(thresh,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
    cnt = contours[0]
    x,y,w,h = cv2.boundingRect(cnt)
    crop = img_dst[y:y+h,x:x+w]

    cv2.namedWindow('image '+str(plane), cv2.WINDOW_NORMAL)
    cv2.imshow('image '+str(plane),crop)

def load_input(input_file):
    # Read from input file and write to a dictionary
    inDict={}
    with open(input_file) as f:
        for line in f:
            line=line.rstrip('\n')
            splitLine = line.split(':')
            if((splitLine[0]=='y1_edge_1') | (splitLine[0]=='y1_edge_2')|
(splitLine[0]=='y2_edge_1')|(splitLine[0]=='y2_edge_2')|
(splitLine[0]=='y3_edge_1')|(splitLine[0]=='y3_edge_2')| \
            (splitLine[0]=='x1_edge_1') | (splitLine[0]=='x1_edge_2') |
(splitLine[0]=='x2_edge_1')|(splitLine[0]=='x2_edge_2')|
(splitLine[0]=='x3_edge_1')|(splitLine[0]=='x3_edge_2') | \
            (splitLine[0]=='z1_edge_1') | (splitLine[0]=='z1_edge_2')|
(splitLine[0]=='z2_edge_1')|(splitLine[0]=='z2_edge_2')|
(splitLine[0]=='z3_edge_1')|(splitLine[0]=='z3_edge_2') | \
            (splitLine[0]=='origin') | (splitLine[0]=='reference_y') |
(splitLine[0]=='reference_x') | (splitLine[0]=='reference_z')):
                line = splitLine[1];
                line = line.split(' ')
                line_int = [int(item) for item in line]
                inDict[(splitLine[0])] = line_int
```

```
    return inDict

if __name__ == "__main__":

    parser = argparse.ArgumentParser(description='Process the inputs.')
    parser.add_argument('img_name', action='store', type=str,
                        help='Give the path of the image')
    parser.add_argument('input_file', action='store', type=str,
                            help='Give the path to the input file')
    args = parser.parse_args()

    # Load the information about the lines, origin and reference point into a
dictionary
    info_dict = load_input(args.input_file)

    # Total number of lines considered
    lines_n = 9

    # Number of directions to be considered
    axes_n = 3

    # Define the names corresponding to the lines

nodes=['y_f_t','y_f_b','y_b_t','x_f_t','x_f_b','x_b_t','z_f_r','z_b_r','z_f_l'
]
    line_eqs=[]
    V={}

    # Get the edges
    edges = generate_edges(info_dict)

    # Get the equations of all the lines
    for lines_idx in range(0,lines_n):
        line_eqs.append(np.cross(edges[nodes[lines_idx]][0],
edges[nodes[lines_idx]][1]))

    # Compute vanishing points
    VX,VY,VZ = compute_vanishing_points(line_eqs, axes_n)

    print ("Vanishing Point along Y=",VY)
    print ("Vanishing Point along X=",VX)
    print ("Vanishing Point along Z=",VZ)

    # Supply the origin of the image coordinate system
    wo=np.array(info_dict['origin'])

    # Setting the reference points along X,Y and Z axis
    ref_Y=np.array(info_dict['reference_y'])
    ref_X=np.array(info_dict['reference_x'])
    ref_Z=np.array(info_dict['reference_z'])

    print ("Reference point along Y=",ref_Y)
    print ("Reference point along X=",ref_X)
    print ("Reference point along Z=",ref_Z)

    # Computing reference length based on image pixel length
    ref_X_len=np.sqrt((wo[0]-ref_X[0])**2 + (wo[1]-ref_X[1])**2 )/10
```

```
ref_Y_len=np.sqrt((wo[0]-ref_Y[0])**2 + (wo[1]-ref_Y[1])**2 )/10
ref_Z_len=np.sqrt((wo[0]-ref_Z[0])**2 + (wo[1]-ref_Z[1])**2 )/10

print ("Reference length along Y=",ref_Y_len)
print ("Reference length along X=",ref_X_len)
print ("Reference length along Z=",ref_Z_len)

# Reshape all arrays to 3 x 1
VX = np.transpose(np.reshape(VX, (1, 3)))
VY = np.transpose(np.reshape(VY, (1, 3)))
VZ = np.transpose(np.reshape(VZ, (1, 3)))
ref_X = np.transpose(np.reshape(ref_X, (1, 3)))
ref_Y = np.transpose(np.reshape(ref_Y, (1, 3)))
ref_Z = np.transpose(np.reshape(ref_Z, (1, 3)))
wo = np.transpose(np.reshape(wo, (1, 3)))

# Compute the scaling factors
a_x = compute_scale(VX, ref_X, ref_X_len, wo)
a_y = compute_scale(VY, ref_Y, ref_Y_len, wo)
a_z = compute_scale(VZ, ref_Z, ref_Z_len, wo)

print ("Scaling constant a_x=",a_x)
print ("Scaling constant a_y=",a_y)
print ("Scaling constant a_z=",a_z)

#Constructing the Projection Matrix using the values calculating above
a_VX=a_x*np.array(VX)
b_VY=a_y*np.array(VY)
c_VZ=a_z*np.array(VZ)
wo=np.array(wo)

P=np.concatenate((a_VX,b_VY,c_VZ,wo),axis=1)
print('\nProjection Matrix\n')
print(P)

#Now calulating the Homography Matrices along the Hxy,Hyz,Hxz
HXY = compute_homography(P,'xy')
HYZ = compute_homography(P,'yz')
HXZ = compute_homography(P,'xz')
print('\nHXY\n')
print(HXY)
print('\nHYZ\n')
print(HYZ)
print('\nHXZ\n')
print(HXZ)

# Load the source image
orig_img = image_view(args.img_name)

# Set Translation offsets to display the image in the target image.
HXY[0,2]=HXY[0,2]-10000
HXY[1,2]=HXY[1,2]+40000

HYZ[0,2]=HYZ[0,2]-1000
HYZ[1,2]=HYZ[1,2]+200000

HXZ[0,2]=HXZ[0,2]-70000
```

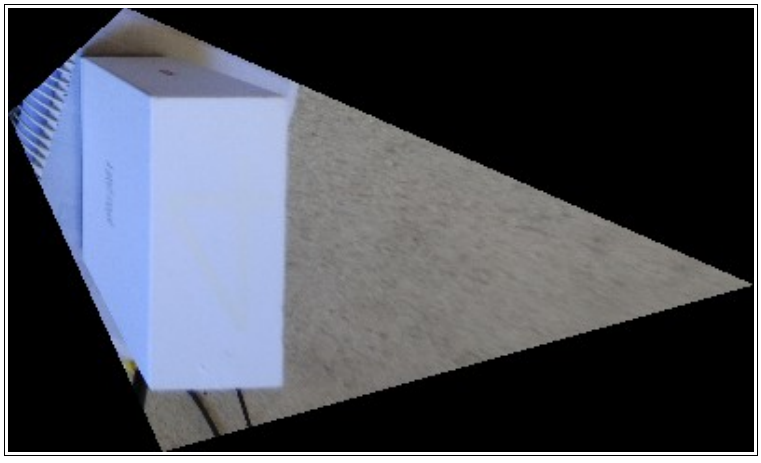
```
HXZ[1,2]=HXZ[1,2]+90000

# Compute the homography maps
transform_map(orig_img, HXY, 'xy')
transform_map(orig_img, HYZ, 'yz')
transform_map(orig_img, HXZ, 'xz')

# Pause and show the images
cv2.waitKey(0)
cv2.destroyAllWindows()
```

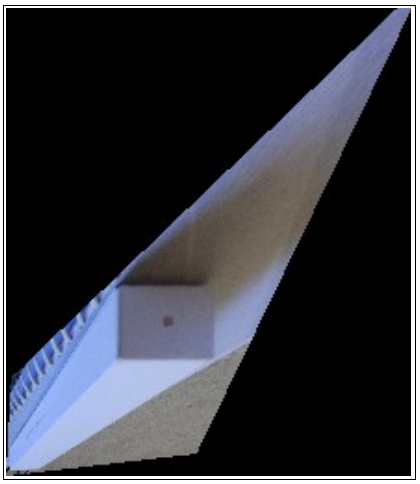
5. After getting the homography matrices from the above code,we apply the homography matrices to each of the planes of the image to generate the texture maps for each of the planes(XY,YZ,ZX). The uncropped and cropped version of the texture maps are shown as below:

Uncropped XY Texture Map:



Cropped XY Texture Map:

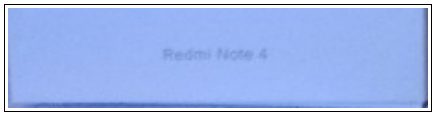
Uncropped XZ Texture Map:



Cropped XZ Texture Map:

Uncropped YZ Texture Map:

Cropped YZ Texture Map:



6. The results obtained from our code is shown below:

```
akshay@akshay-XPS-15-9550:~/ncsu/DIS/p1$ python svm_final.py Box_Image.jpg input.txt
('Vanishing Point along Y=', array([ -2.86093237e+03,  -1.61019514e+03,   1.00000000e+00]))
('Vanishing Point along X=', array([  4.28937240e+03,  -3.12382218e+02,   1.00000000e+00]))
('Vanishing Point along Z=', array([ -9.76047865e+02,   1.19476184e+04,   1.00000000e+00]))
('Reference point along Y=', array([ 86, 588,   1]))
('Reference point along X=', array([2322, 1239,    1]))
('Reference point along Z=', array([1808, 1109,    1]))
('Reference length along Y=', 194.3419923742679)
('Reference length along X=', 84.560037842943274)
('Reference length along Z=', 65.824615456529642)
('Scaling constant a_x=', array([[ 0.0039902]]))
('Scaling constant a_y=', array([[ 0.00271999]]))
('Scaling constant a_z=', array([[-0.00089361]]))

Projection Matrix

[[  1.71154466e+01  -7.78171132e+00   8.72210297e-01   1.64600000e+03]
 [ -1.24646701e+00  -4.37971686e+00  -1.06765623e+01   1.74700000e+03]
 [  3.99019834e-03   2.71999136e-03  -8.93614266e-04   1.00000000e+00]]

HXY

[[  1.71154466e+01  -7.78171132e+00   1.64600000e+03]
 [ -1.24646701e+00  -4.37971686e+00   1.74700000e+03]
 [  3.99019834e-03   2.71999136e-03   1.00000000e+00]]

HYZ

[[ -7.78171132e+00   8.72210297e-01   1.64600000e+03]
 [ -4.37971686e+00  -1.06765623e+01   1.74700000e+03]
 [  2.71999136e-03  -8.93614266e-04   1.00000000e+00]]

HXZ

[[  1.71154466e+01   8.72210297e-01   1.64600000e+03]
 [ -1.24646701e+00  -1.06765623e+01   1.74700000e+03]
 [  3.99019834e-03  -8.93614266e-04   1.00000000e+00]]
```

7. After extracting and cropping the feature maps,we generate the 3D model of the image using VRML. The VRML code(**3DModelling_Code.wrl**) is as given below and also attached in the zipped folder.

VRML Code:

```
#VRML V2.0 utf8

Collision {
    collide FALSE
    children [
 Shape {
       appearance Appearance {
             texture ImageTexture {
                   url "h_xy_cropped.png"
             }
       }
       geometry IndexedFaceSet {
             coord Coordinate {
                   point [
```

```
                        0.000000 0.000000 66.000000,
                        85.000000 0.000000 66.000000,
                        85.000000 194.000000 66.000000,
                        0.000000 194.000000 66.000000,
                  ]
            }
            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 "h_xz_cropped.png"
            }
      }
      geometry IndexedFaceSet {
            coord Coordinate {
                  point [
                        0.000000 0.000000 0.000000,
                        85.000000 0.000000 0.000000,
                        85.000000 0.000000 66.000000,
                        0.000000 0.000000 66.000000,
                  ]
            }
            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 "h_xz_cropped.png"
            }
      }
      geometry IndexedFaceSet {
            coord Coordinate {
                  point [
                        0.000000 0.000000 66.000000,
                        85.000000 0.000000 66.000000,
                        85.000000 0.000000 66.000000,
                        0.000000 0.000000 66.000000,
                  ]
            }
            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 "h_yz_cropped.png"
            }
      }
      geometry IndexedFaceSet {
            coord Coordinate {
                  point [
                        0.000000 194.000000 0.000000,
                        0.000000 0.000000 0.000000,
                        0.000000 0.000000 66.000000,
```

```
                        0.000000 194.000000 66.000000,
                ]
        }
        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
    }
}
    ]
}
```

8. The .wrl file generated from the above image is opened in **view3Dscene** to get the 3D image as shown below.

9. Contribution of each Team Member:

1. **Kalyan Ghosh:** Calculation of Projection Matrices and Homography Matrices
2. **Akshay Kalkunte Suresh:** Annotation of image and calculation of vanishing points.
3. **Bhargav Ram:** Texture Mapping and 3D Modelling