Problem1:

1.Image Collection:

a. I have used chessboard pattern and collected 50 images.

b. Link for the images: https://drive.google.com/drive/folders/1_3MWJgBhGMhOnPYzN7qCB2VqcFZBP5l4?usp=sharing

1. First mount the drive and read the 50 images using glob.

```
#Mount the google drive
from google.colab import drive
drive.mount('/content/drive')

#Reading 50 images of chessboard pattern
import cv2
#Using glob to read all images inside folder
import glob
#for imshow
from google.colab.patches import cv2_imshow
import matplotlib.pyplot as plt

#Read the folder path
img_path = '/content/drive/MyDrive/ENPM673/project3/images'

#List to store captured images
cap_imgs = []

#For loop to iterate over all captured images
for image_path in glob.glob(img_path+"/*.JPG"):
    img= cv2.imread(image_path)
    cap_imgs.append(img)

#Displaying one image out of 50 imgs
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mo
```

2.Calibration Pipeline:

a.The pattern used is Chessboard pattern.

b. Checkerboard patterns are used commonly because of the sharp changes in gradient at the corners of the Squares. Thse corners sit at the intrsection of checkerboard lines which helps in detecting corners precisely.

I used cv2.findChessboardCorners method to find the corners in the chessboard pattern. Internally it uses corner detection algorithms like Harris Corner detector.

Inputs for findChessboardCorners are Chessboard image, patternsize--> defining rows and cols of chessboard and certain flags

This method returns output as bool(True/False) and array of detected corners

After this to get only correct corners I have used cv2.cornerSubPix to filter out good corner points.

To view the corners detected I have used cv2.drawChessboardCorners.

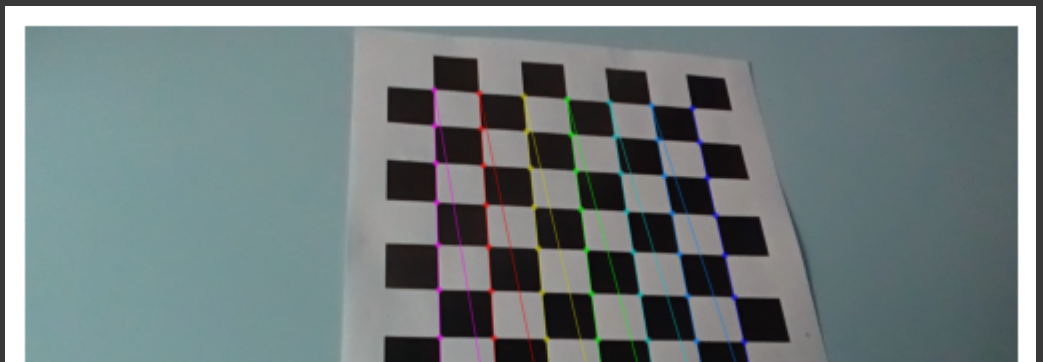
c. Opencv methods used cv2.findChessboardCorners,cv2.cornerSubPix,cv2.drawChessboardCorners,cv2.calibrateCamera.

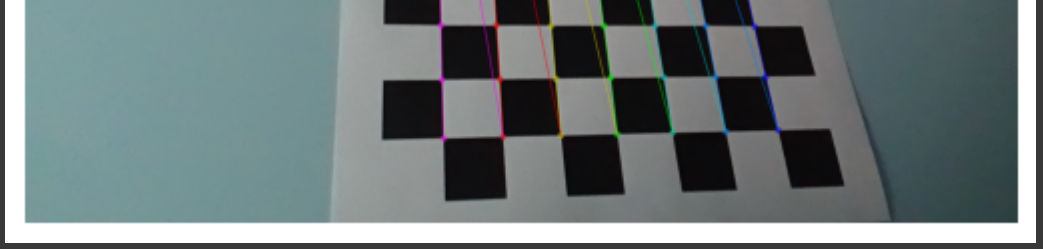
Pipeline:

1. Take 50 images of chessboard pattern.
2. Use the world coords of Chessboard pattern(rows,cols, width of square)
3. Do preprocessing of images(Grayscale, resizing) and find corners using cv2.findChessboardCorners.
4. Filter out the corners using cv2.cornerSubPix
5. Display the corners detected using cv2.drawChessboardCorners.
6. Find the intrinsic camera parmeters using cv2.calibrateCamera.
7. use cv2.undistort to undistort the image.
8. use cv2.projectPoints() to project 3-D Pts into 2-D Img PLane

```
import numpy as np
#graycale list
gray_imgs = []

#Iterating over all the images
for img in cap_imgs:
  img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  gray_imgs.append(img)

#Defining known chessboard parameters
rows = 10
cols = 7
size = 23 #mm

# Form 3-d coords of pattern
#Form a empty matrix of 0's
pts_3d = np.zeros((rows * cols, 3), np.float32)
pts_3d[:, :2] = np.mgrid[0:rows, 0:cols].T.reshape(-1, 2)

#Lists
#world_points_contains_corners
```

```
#world points contains corners
world_pts = []
#image_pts w.r.t pts in image plane
image_pts = []

#plot list
crner_imgs = []

#Iterate over all the gray scale images
index = 0
for img in gray_imgs:
  #disp img
  tmp = np.copy(cap_imgs[index])
  plot_img = np.copy(cap_imgs[index])

  index += 1
  #Use the method to find the corners
  val, corner_pts = cv2.findChessboardCorners(img, (rows,cols), None)

  #Checking all 50 imgs corners are detected successfully
  #print(val)

  # If val is true then we need to filter the corner pts

  if val == True:
    world_pts.append(pts_3d)

    #Filtering corners using cv2 method
    iter,epsilon = 30, 0.001
    win_size = (11,11)
    filtered_corners = cv2.cornerSubPix(img, corner_pts, win_size, (-1,-1), (cv2.TERM_CRIT

    #Appending to image plane pts
    image_pts.append(filtered_corners)

    # Displaying the corners only val is true
    cv2.drawChessboardCorners(tmp, (rows,cols), filtered_corners, val)

    #used for plotting
    for i in range(len(filtered_corners)):
      x, y = filtered_corners[i].ravel()
      cv2.circle(plot_img, (int(x), int(y)), 3, (0, 255, 0), -1)  # plot orig before calib

    #appending to lst for plotting
    crner_imgs.append(plot_img)

#Display one image
plt.imshow(tmp)
plt.axis('off')
plt.show()
```





Calibrate the camera using cv2.calibrateCamera.

This method returns Camera matrix,Rotational and Translational Coeff and Distortion coeffs.

```
#Calling camera calibratiopn method
shape = img.shape[::-1]
val, camera_mat, dist_coeff, rot_coeff, trans_coeff = cv2.calibrateCamera(world_pts, image_p

#Printing the o/ps
print("Camera Intrinsic Matrix:")
print(camera_mat)
print("Lens Distortion Coefficients:")
print(dist_coeff)
print("Rotational vectors:")
print(rot_coeff)
print("Translational Vector:")
print(trans_coeff)
```

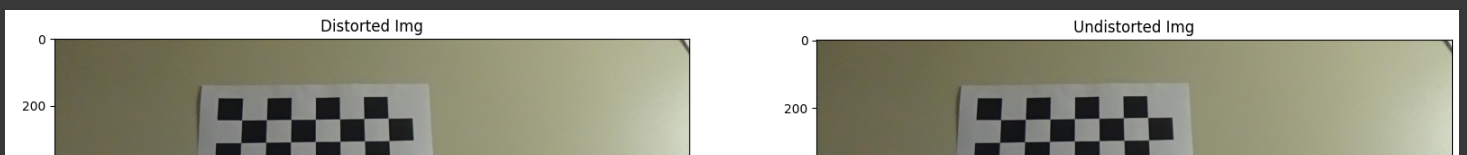
```
Camera Intrinsic Matrix:
[[1.51111303e+03 0.00000000e+00 9.41787220e+02]
 [0.00000000e+00 1.50927488e+03 5.24395989e+02]
 [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
Lens Distortion Coefficients:
[[-0.00197175  0.18925725 -0.00301686 -0.00412019 -0.32569795]]
Rotational vectors:
(array([[-0.2148225 ],
       [-0.24341224],
       [-1.5809714 ]]), array([[-0.34049234],
       [-0.3731376 ],
       [-1.55506446]]), array([[-0.79652125],
       [-0.563597  ],
       [-1.22794869]]), array([[-0.71152557],
       [-0.50163161],
       [-1.20668552]]), array([[-0.68175787],
       [-0.47850524],
       [-1.20146961]]), array([[-0.55814318],
       [-0.39819372],
       [-1.17745368]]), array([[-0.80479858],
       [-0.36872445],
       [-0.79239494]]), array([[-0.78198233],
       [-0.36582337],
       [-0.81579922]]), array([[-0.69324771],
       [-0.35917131],
       [-0.77362636]]), array([[-0.60208049],
       [-0.36948761],
       [-0.71597486]]), array([[-0.43687798],
       [-0.33887367],
       [-0.6809306 ]]), array([[-0.48857653],
       [-0.20885085],
       [-0.28084934]]), array([[-0.60991917],
       [-0.20662102],
       [-0.35591478]]), array([[-0.64209022],
       [-0.2196916 ],
```

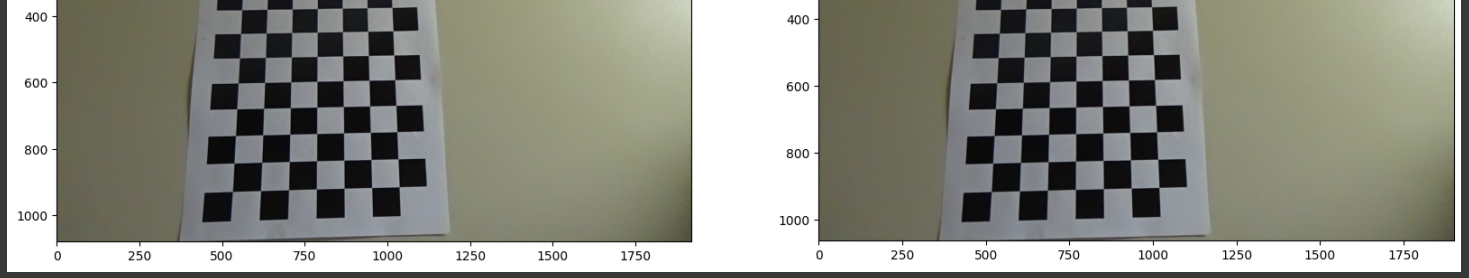
```
[-0.39979503]]), array([[-0.90633365],
[-0.26705798],
[-0.47608576]]), array([[-0.32184682],
[-0.20646766],
[-0.04342845]]), array([[-0.3394594 ],
[-0.11511767],
[-0.07321635]]), array([[-0.40974904],
[-0.12899931],
[-0.11778377]]), array([[-0.49867906],
[-0.13514084],
[-0.16236537]]), array([[-0.53409475],
[-0.06454082],
[-0.09718094]]), array([[-0.96356453],
[-0.07923606],
[-0.12638204]]), array([[0.07002694],
[0.84687585],
[3.00562541]]), array([[-0.1436168 ],
[-0.48562277],
[-3.04030337]]), array([[-0.36184313],
[ 0.85922998],
[ 2.87626655]]), array([[-0.30933567],
[ 0.72507829],
[ 1.64115896]]), array([[-0.44281389],
```

Image undistortion:

1. Use cv2.getOptimalNewCameraMatrix() to minimize the black pixels around the edges. It is used to get optimal camera mat to be used in undistortion.
2. After getting new camera matrix use undistort to perform Image distortion.

```
import matplotlib.pyplot as plt

#Open sample distorted image
img1 = cap_imgs[0]
#Do getOptimalNewCameraMatrix() to get new camera matrix
height, width = img1.shape[:2]
New_Cam_Mat, roi = cv2.getOptimalNewCameraMatrix(camera_mat, dist_coeff, (width,height), 1

#undistort and crop roi
undistort_img = cv2.undistort(img1, camera_mat, dist_coeff, None, New_Cam_Mat)
#crop roi to remove black are after undistortion
undistort_img = undistort_img[roi[1]:roi[1]+roi[3], roi[0]:roi[0]+roi[2]]

# Plotting using matplot
plt.figure(figsize=(20, 10))

# Plot distortedImg
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img1, cv2.COLOR_BGR2RGB))
plt.title('Distorted Img')
# Plot undistortedImg
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(undistort_img, cv2.COLOR_BGR2RGB))
plt.title('Undistorted Img')
plt.show()
```





Ways to improve the accuracy of Camera Calibration Matrix:

1. Increase the no of Calibr Images
2. Take photos from all orientations.
3. Make sure the calibration board is flat.
4. Try using larger calib Board

3. Reprojection Error Analysis:

a. Plot a graph showing the reprojection error for each image used in the calibration process.

b. The x-axis should represent the image number, and the y-axis should represent the reprojection error.

use cv2.projectPoints() to project 3D Points into 2D image plane

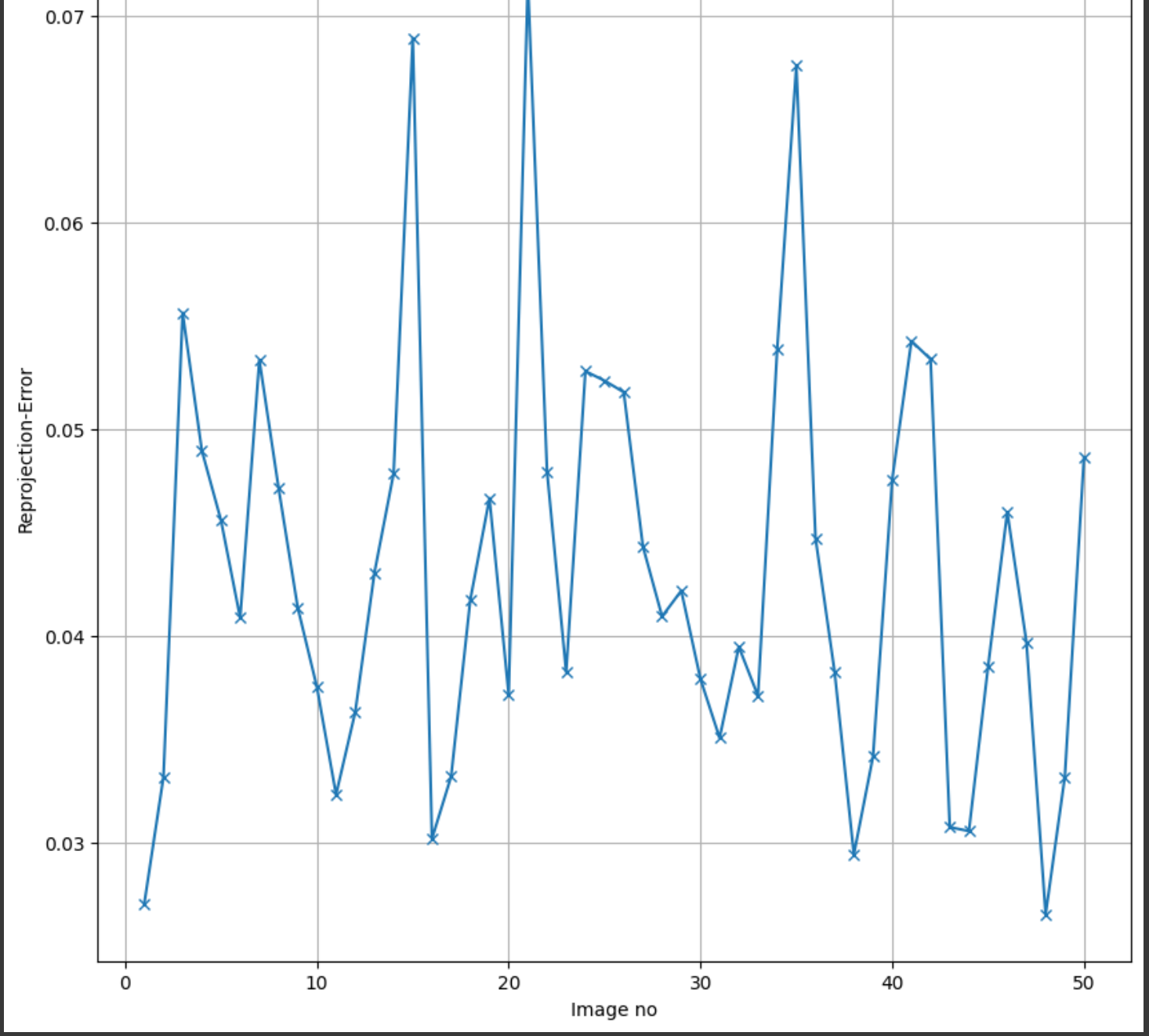
print reprojection error

after getting reprojection error plot the reprojection error to visualize

```
import matplotlib.pyplot as plt

#list to store reprojection err
reproj_lst = []

for i in range(len(world_pts)):
  #using cv2.project points
  Img_Pts, _ = cv2.projectPoints(world_pts[i], rot_coeff[i], trans_coeff[i], camera_mat, dis
  #calc error
  Err = cv2.norm(image_pts[i], Img_Pts, cv2.NORM_L2)/len(Img_Pts)
  print("Image ",i+1,"--->Reprojection Error :",Err)
  #appending to list
  reproj_lst.append(Err)
```

```
#plotting
plt.figure(figsize=(10, 10))
#continuois values
plt.plot(np.arange(1, len(reproj_lst)+1), reproj_lst, marker='x', linestyle='-')
plt.title('Reprojection Error Plot')
plt.xlabel('Image no')
plt.ylabel('Reprojection-Error')
plt.grid(True)
plt.show()
```

```
Image  1 --->Reprojection Error : 0.027019799649235794
Image  2 --->Reprojection Error : 0.03316267584122581
Image  3 --->Reprojection Error : 0.05563421277921326
Image  4 --->Reprojection Error : 0.048961417456061716
Image  5 --->Reprojection Error : 0.04557919205573224
Image  6 --->Reprojection Error : 0.040853977663401486
Image  7 --->Reprojection Error : 0.05334991644144941
Image  8 --->Reprojection Error : 0.04715369514450029
Image  9 --->Reprojection Error : 0.04136216854219647
Image  10 --->Reprojection Error : 0.037509111087557674
Image  11 --->Reprojection Error : 0.03230107678869652
Image  12 --->Reprojection Error : 0.03630148105928074
Image  13 --->Reprojection Error : 0.043037928208960054
Image  14 --->Reprojection Error : 0.04785231218799351
Image  15 --->Reprojection Error : 0.06893797570688577
Image  16 --->Reprojection Error : 0.0301635485855765
Image  17 --->Reprojection Error : 0.03318003084026608
Image  18 --->Reprojection Error : 0.04173835849920571
Image  19 --->Reprojection Error : 0.04665642033137087
Image  20 --->Reprojection Error : 0.037145447909416346
Image  21 --->Reprojection Error : 0.071706944725014
Image  22 --->Reprojection Error : 0.04795339603088213
Image  23 --->Reprojection Error : 0.038227791423389675
Image  24 --->Reprojection Error : 0.052804287022332
Image  25 --->Reprojection Error : 0.05234274863329719
Image  26 --->Reprojection Error : 0.0517885704365713
Image  27 --->Reprojection Error : 0.04431558427622138
Image  28 --->Reprojection Error : 0.04092758570474348
Image  29 --->Reprojection Error : 0.04217337798701796
Image  30 --->Reprojection Error : 0.03789893555755034
Image  31 --->Reprojection Error : 0.03504278757143183
Image  32 --->Reprojection Error : 0.03946043208402014
Image  33 --->Reprojection Error : 0.037078611623459205
Image  34 --->Reprojection Error : 0.053840733805242236
Image  35 --->Reprojection Error : 0.06759728787284146
Image  36 --->Reprojection Error : 0.04472602411625202
Image  37 --->Reprojection Error : 0.038211482097698174
Image  38 --->Reprojection Error : 0.0294020575387899
Image  39 --->Reprojection Error : 0.03414690872662069
Image  40 --->Reprojection Error : 0.04756430829941694
Image  41 --->Reprojection Error : 0.05426408856170814
Image  42 --->Reprojection Error : 0.05338441226416529
Image  43 --->Reprojection Error : 0.030723159638107126
Image  44 --->Reprojection Error : 0.030534249211503103
Image  45 --->Reprojection Error : 0.03851796694113959
Image  46 --->Reprojection Error : 0.04598604493419736
Image  47 --->Reprojection Error : 0.03966770688509269
Image  48 --->Reprojection Error : 0.026500809528470285
Image  49 --->Reprojection Error : 0.03316343014498374
Image  50 --->Reprojection Error : 0.048641788807226215
```



c. Discuss the significance of the reprojection error and its implications for the accuracy of the calibration.

1. The reprojectionError is crucial in assessing camera calibration accuracy.Lower the error indicates better the calibration is ,and how well the calibrated model can predict the positions of Image Pts in 3-D space.
2. BY reducing the Reprojection error, the camera intrinsic parameters are correctly estimated and used in the applications like 3D recontruction and Depth estimation.
3. High reprojection errors create significant diferences between projected and observed pts, reducing the calculated distance and depth.So, reducing reprojection error is crucial for precise measuremnts.
4. By calculating the reprojection error we can estimate how well the camera calibration system is.
5. For application like Augmented Reality it is crucial to have proper calibration parameters. This can be verified by checking the Reprojecton error

verified by checking the Reprojecton error.

4.Visualization of Calibration Results:

a. After calibration, redraw the detected corners or centroids on the original images.

b. Show the detected corners/centroids before and after calibration.

c. Differentiate between the original corner/centroid detection and the reprojection of these points based on the calculated camera parameters (R, T, and K).

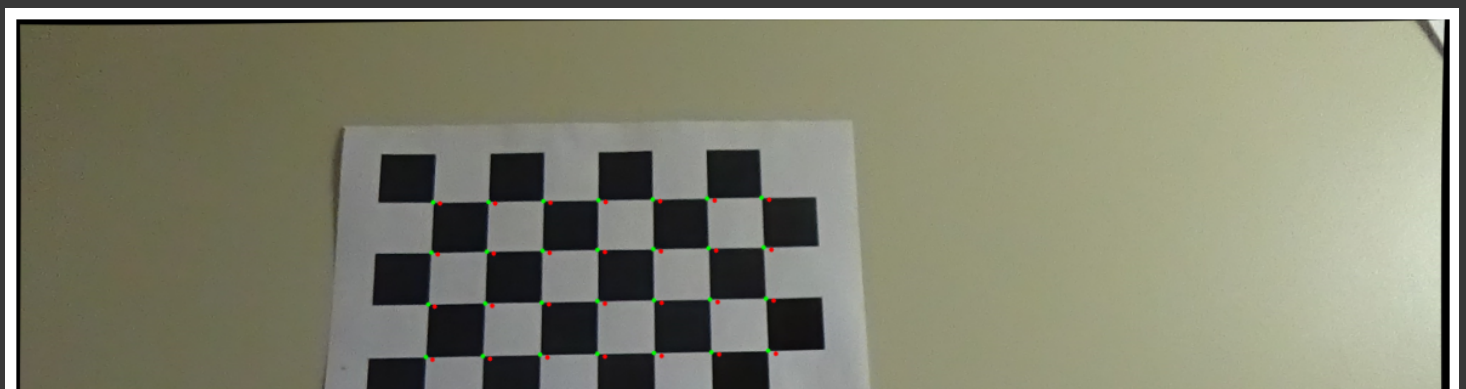
d. Use different colors to represent the original and reprojected points for clarity.

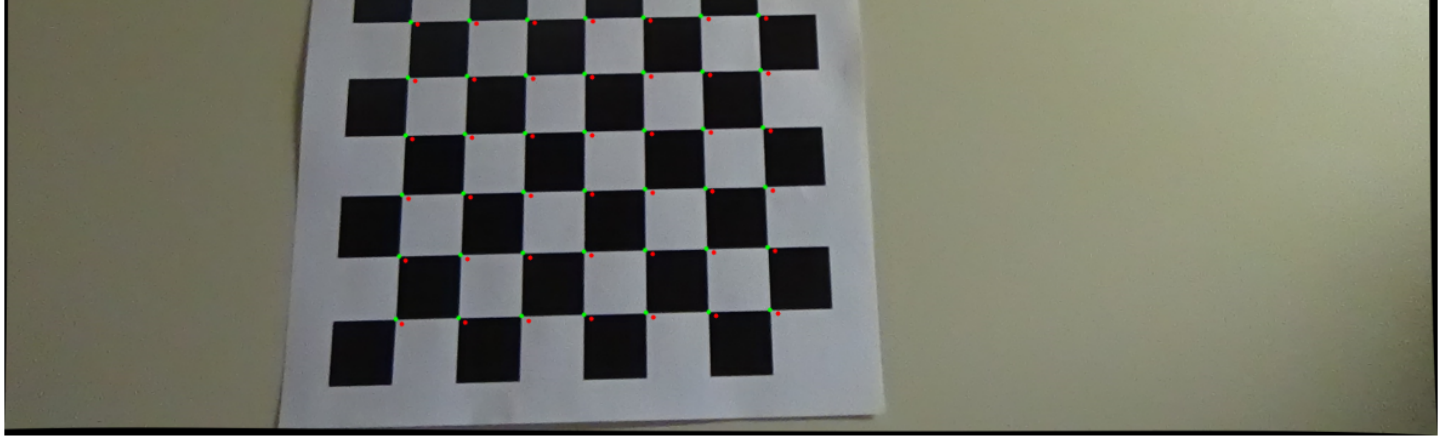
Here the corners before calibration are plotted in green color.

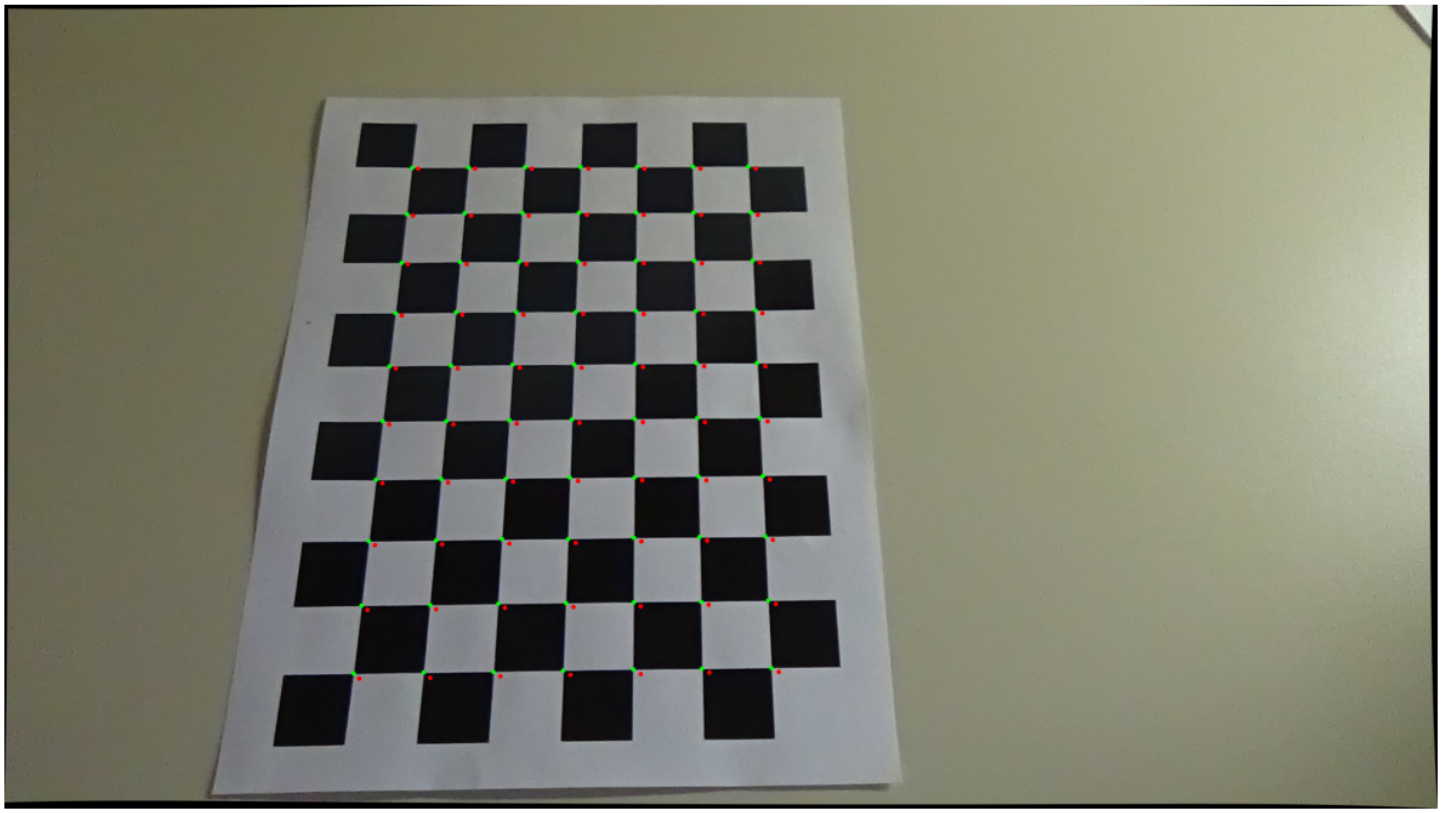
The image has to be undistorted using camera parameters to perform camera calibration

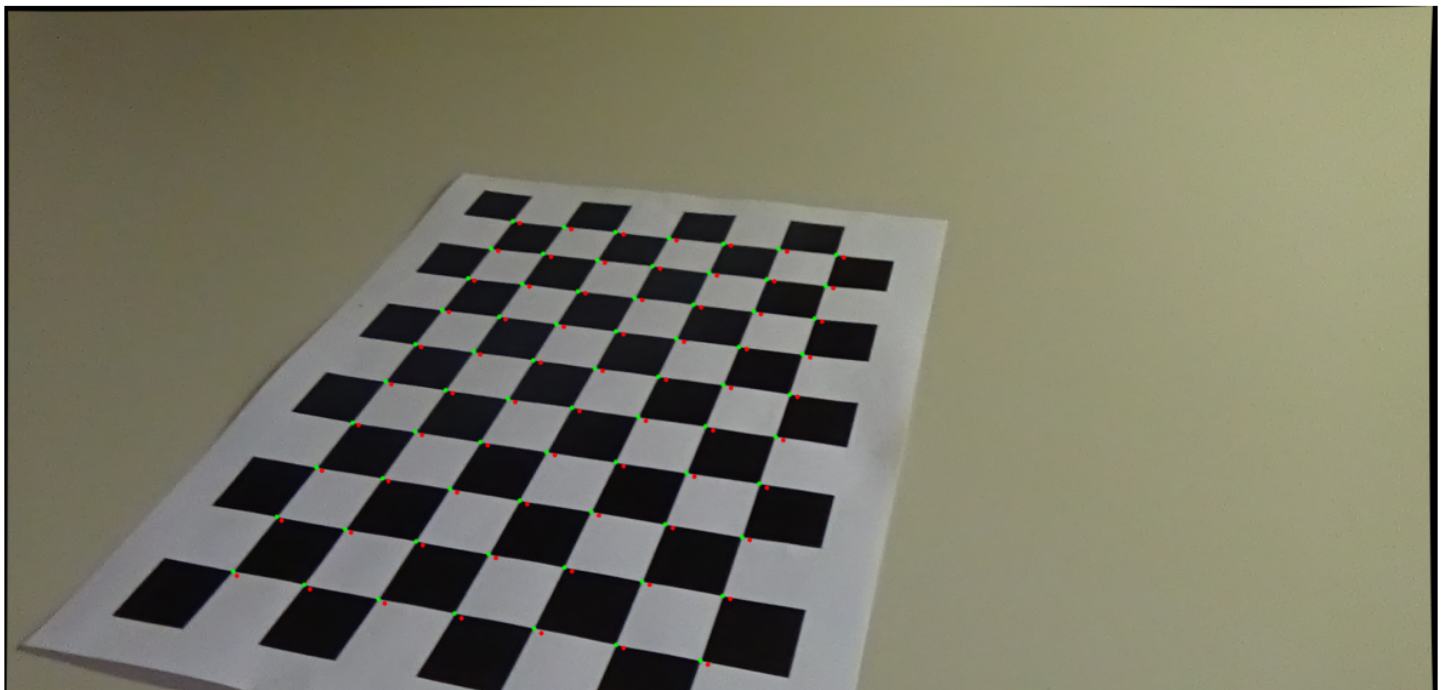
The corners after calibration are plotted in red color.

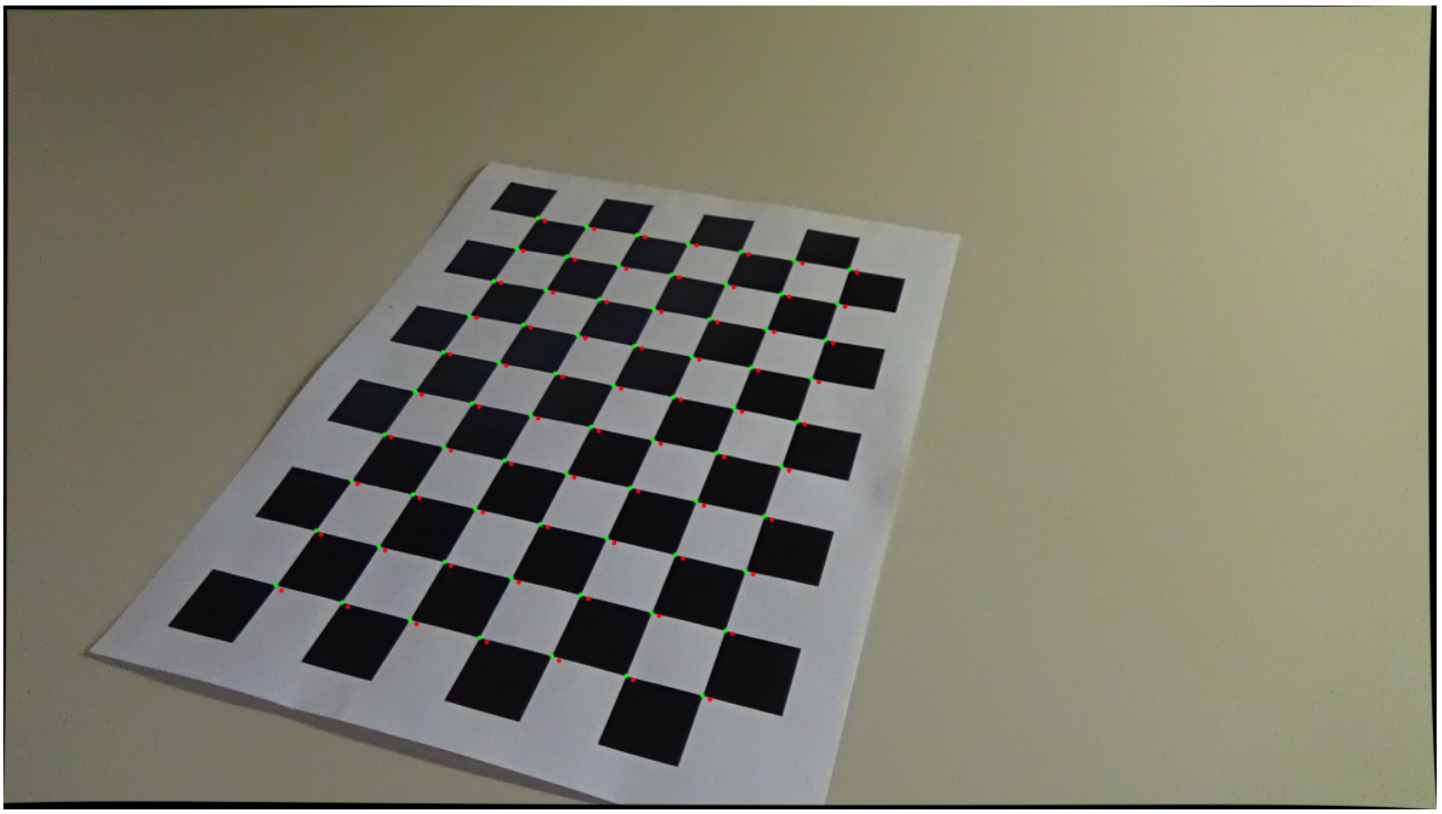
```
#perform undistortion for all images
index = 0
#lst to store all imgs
final_Imgs = []

for img in crner_imgs:
  h, w = img.shape[:2]
  #undistrotion
  newCamMat, ROI = cv2.getOptimalNewCameraMatrix(camera_mat, dist_coeff, (w,h), 1, (w,h))
  img = cv2.undistort(img, camera_mat, dist_coeff, None, newCamMat)
  #projecting pts
  imgpt, _= cv2.projectPoints(world_pts[index], rot_coeff[index], trans_coeff[index], came
  index+=1
  #plotting red
  for i in range(len(imgpt)):
    x, y = imgpt[i].ravel()
    cv2.circle(img, (int(x), int(y)), 3, (0, 0, 255), -1)  # plot orig after calib in red
  final_Imgs.append(img)

fig, axes = plt.subplots(5, 1, figsize=(50, 50))

#Plotting all images before calibration in green color
index = 0
for img in final_Imgs:
  if(index==5):  #plot 5 imgs
    break
  axes[index].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
  axes[index].axis('off')
  index+=1
```

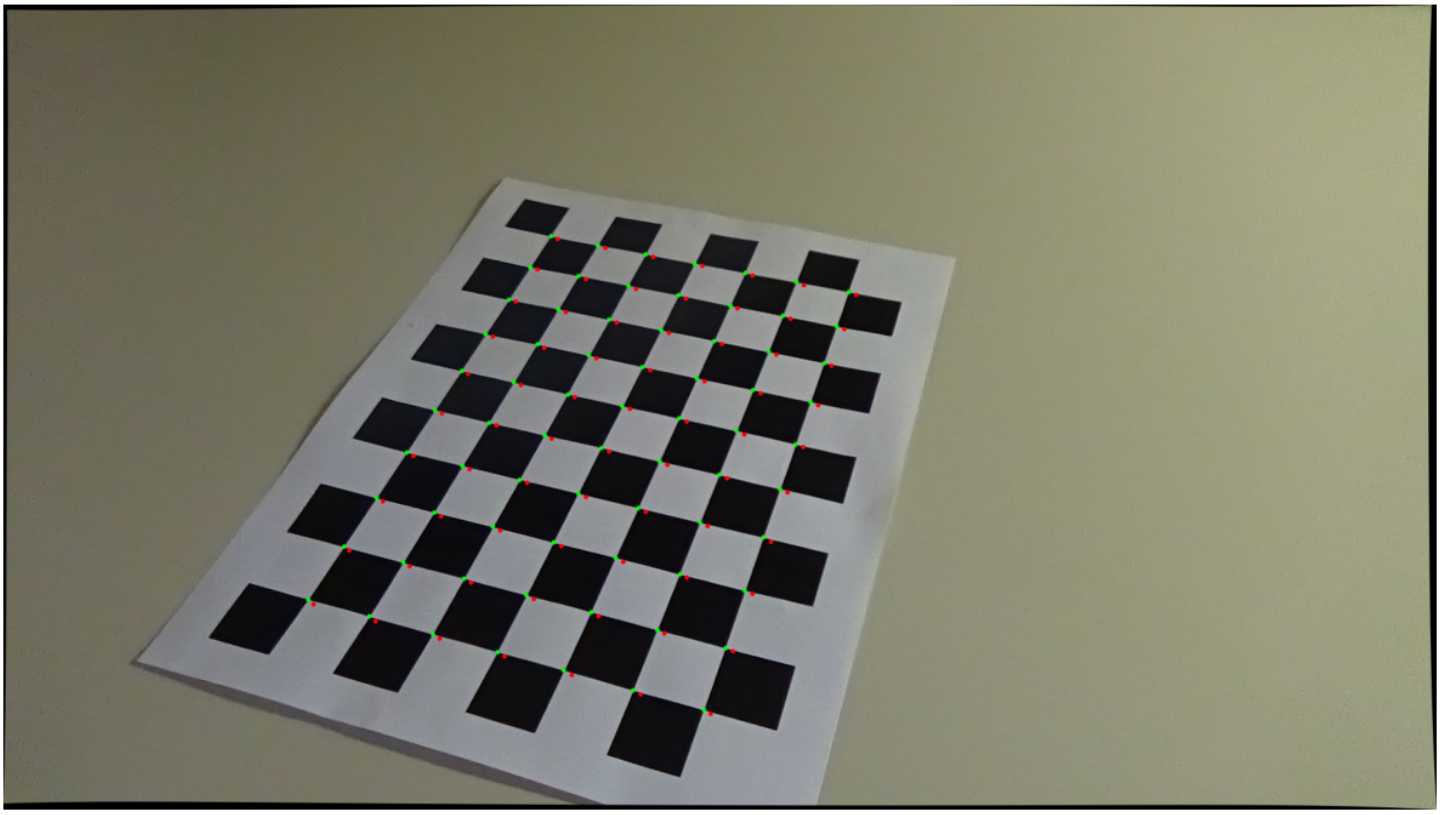












Problem2:

Pipeline for Creating a Stereo Vision System:

1. Calibration:

a.Identify matching features between the two images in each dataset using any feature matching algorithms

Here the given images are copied to my drive.

SIFT feature detection is uded to match feature between two images.

```
#Store the images from all the dataset in lists
import cv2
from google.colab.patches import cv2_imshow

#Read the image datases all rooms
classroom_path   = '/content/drive/MyDrive/ENPM673/project3/classroom'
storageroom_path = '/content/drive/MyDrive/ENPM673/project3/storageroom'
traproom_path    = '/content/drive/MyDrive/ENPM673/project3/traproom'

#List for storing the images
classroom_lst = []
storageroom_lst = []
traproom_lst = []

#Reading the images and saving the images to the list
classroom_lst.append(cv2.imread(classroom_path+"/im0.png"))
classroom_lst.append(cv2.imread(classroom_path+"/im1.png"))

storageroom_lst.append(cv2.imread(storageroom_path+"/im0.png"))
storageroom_lst.append(cv2.imread(storageroom_path+"/im1.png"))

traproom_lst.append(cv2.imread(traproom_path+"/im0.png"))
traproom_lst.append(cv2.imread(traproom_path+"/im1.png"))

#Function to perform SIFT
def SIFT_detection(img1, img2):
  #Feature Detection using SIFT detector
```

```
  sift_detector = cv2.SIFT_create()

  # Finding keypts and descriptors using SIFT
  keypts1, desc1 = sift_detector.detectAndCompute(img1, None)
  keypts2, desc2 = sift_detector.detectAndCompute(img2, None)

  #Matcher creation
  bf_matcher = cv2.BFMatcher()

  #KNN Matcher
  matches = bf_matcher.knnMatch(desc1, desc2, 2)

  #Ratio test to remove bad matches with dist .75
  final_matches = []
  for i, j in matches:
      if i.distance < 0.75 * j.distance:
          final_matches.append(i)

  #Display the matches between two images
  disp_img = cv2.drawMatches(img1, keypts1, img2, keypts2, final_matches, None, flags=cv2.

  #imshow for display
  cv2_imshow(disp_img)

  #returning good matches
  return keypts1, keypts2,final_matches

#Call for dataset1
key_pts11, key_pts12, matches1 = SIFT_detection(classroom_lst[0], classroom_lst[1])

#Call for dataset2
key_pts21, key_pts22, matches2 = SIFT_detection(storageroom_lst[0], storageroom_lst[1])

#Call for dataset3
key_pts31, key_pts32, matches3 = SIFT_detection(traproom_lst[0], traproom_lst[1])
```

b. Estimate the Fundamental matrix using RANSAC method based on the matched features.

Using opencv function cv2.findFundamentalMat to find the fundamental matrix and RANSAC is used

```
import numpy as np

# Function to get fundamental matrix
def find_fund_mat(key_pts1, key_pts2, matches):
  #convert to numpy
  Pt_1 = np.float32([key_pts1[i.queryIdx].pt for i in matches]).reshape(-1,1,2)
  Pt_2 = np.float32([key_pts2[i.trainIdx].pt for i in matches]).reshape(-1,1,2)

  # Using opencv fun to get fundamental matrix using ransac
  fund_mat, mask = cv2.findFundamentalMat(Pt_1, Pt_2, cv2.FM_RANSAC,0.1,0.99)

  return Pt_1, Pt_2, fund_mat

#Dataset1
Pt_11, Pt_12, fund_mat1 = find_fund_mat(key_pts11, key_pts12, matches1)
print("Fundamental matrix for class room is:\n", fund_mat1)

#Dataset2
Pt_21, Pt_22, fund_mat2 = find_fund_mat(key_pts21, key_pts22, matches2)
print("Fundamental matrix for storage room is:\n", fund_mat2)

#Dataset3
Pt_31, Pt_32, fund_mat3 = find_fund_mat(key_pts31, key_pts32, matches3)
print("Fundamental matrix for trap room is:\n", fund_mat3)
```

```
Fundamental matrix for class room is:
 [[ 6.96279678e-09  1.91248773e-06  1.59210853e-03]
 [-1.08296127e-06  5.09749009e-07  2.51035404e-01]
 [-1.82773816e-03 -2.51704398e-01  1.00000000e+00]]
Fundamental matrix for storage room is:
 [[ 3.02348122e-09 -4.45848227e-07  8.05256229e-03]
 [ 5.30268104e-07  1.34387280e-07  2.04999483e-01]
 [-8.11151133e-03 -2.05576414e-01  1.00000000e+00]]
Fundamental matrix for trap room is:
 [[ 6.38327261e-09  1.64159225e-05 -6.21271033e-03]
 [-2.05938282e-05  3.48075243e-07 -3.06783135e+00]
 [ 9.00701735e-03  3.07231601e+00  1.00000000e+00]]
```

c. Compute the Essential matrix from the Fundamental matrix considering calibration parameters.

Essential Matrix = (Calibration Matrix-2nd cam)^Transpose . Fundamantal Matrix . Calibration -1st cam

we get calibration matrix as

fx 0 cx

0 fy cy

0 0 1

we already know principal points and fx,fy from the text file given.

```
# Function to read from text file
import re

def read_txt_file(filePath):
  with open(filePath, 'r') as file:
      # Read the content
      line = file.readlines()
  for i in range(len(line)):
```

```
  for i in range(len(line)):
    if i==3:
      #print(line[i])
      baseline = float(line[i].split('=')[1].strip())
    #reading baseline and vmin and vmax
    if i==6:
      ndisp = int(line[i].split('=')[1].strip())
    if i==7:
      vmin = int(line[i].split('=')[1].strip())
    if i==8:
      vmax = int(line[i].split('=')[1].strip())

  # using re to find all numbers
  number_lst1 = re.findall(r'\d+\.\d+|\d+', line[0])
  number_lst2 = re.findall(r'\d+\.\d+|\d+', line[1])

  # Convert str to float
  matrix1 = [float(val) for val in number_lst1]
  #remove 1st number since its cam number
  matrix1  = np.array(matrix1[1:])
  # Convert str to float
  matrix2 = [float(val) for val in number_lst2]
  #remove 1st number since its cam number
  matrix2  = np.array(matrix2[1:])

  #Reshaping to 3x3 matrix
  cal_mat1 = matrix1.reshape(3, 3)
  cal_mat2 = matrix2.reshape(3, 3)

  return cal_mat1, cal_mat2,baseline,ndisp,vmin,vmax

#Reading path of calib.txt dataset1
path1 = '/content/drive/MyDrive/ENPM673/project3/classroom/calib.txt'
path2 = '/content/drive/MyDrive/ENPM673/project3/storageroom/calib.txt'
path3 = '/content/drive/MyDrive/ENPM673/project3/traproom/calib.txt'

#Calling read txt file functions for all the datasets
cal_mat11, cal_mat12,baseline1, ndisp1,vmin1, vmax1 = read_txt_file(path1)
cal_mat21, cal_mat22,baseline2, ndisp2,vmin2, vmax2 = read_txt_file(path2)
cal_mat31, cal_mat32,baseline3, ndisp3,vmin3, vmax3 = read_txt_file(path3)

#define a function to find the Essential matrix
def essential_matrix(Fund_Mat, cal_mat1, cal_mat2):
  #perfoming cam2_mat^transpose . Fund_mat . cam1_mat
  Ess_mat = np.matmul(np.matmul(cal_mat2.T, Fund_Mat), cal_mat1)
  return Ess_mat

#Calling for all datasets
Ess_mat1 = essential_matrix(fund_mat1, cal_mat11, cal_mat12)
Ess_mat2 = essential_matrix(fund_mat2, cal_mat21, cal_mat22)
Ess_mat3 = essential_matrix(fund_mat3, cal_mat31, cal_mat32)

#dataset 1
print("Essential matrix for class room is:\n", Ess_mat1)
#dataset 2
print("Essential matrix for storage room is:\n", Ess_mat2)
#dataset 3
print("Essential matrix for trap room is:\n", Ess_mat3)
```

```
Essential matrix for class room is:
 [[ 2.12320332e-02  5.83185236e+00  4.56413170e+00]
 [-3.30233242e+00  1.55440525e+00  4.38815359e+02]
 [-4.20154948e+00 -4.39011159e+02  7.91189376e-01]]
Essential matrix for storage room is:
 [[ 9.17610603e-03 -1.35312585e+00  1.36123143e+01]
 [ 1.60933573e+00  4.07858306e-01  3.58001912e+02]
 [-1.36269343e+01 -3.58635197e+02  7.18403928e-01]]
Essential matrix for trap room is:
 [[ 1.99760160e-02  5.13725093e+01  4.33302055e+00]
 [-6.44469799e+01  1.08927772e+00 -5.47306650e+03]
 [-3.25731816e+00  5.47224889e+03  4.22399590e+00]]
```

d. Decompose the Essential matrix into rotation and translation matrices.

Decomposing is done uisng recoverpose opencv function

R is rotational matrix

T is translational matrix

```
#Function to decompose Essential matrix and set Rotaional and transation matrix
def decompose_ess_matrix(Ess_mat,pt1,pt2,cam_mat):
    #using opencv recoverpose to get rot and trans matrix
    _, R, T, _ = cv2.recoverPose(Ess_mat, pt1, pt2,cam_mat)
    return R, T

#Calling all datasets
R1, T1 = decompose_ess_matrix(Ess_mat1,Pt_11, Pt_12,cal_mat11)
R2, T2 = decompose_ess_matrix(Ess_mat2,Pt_21, Pt_22,cal_mat21)
R3, T3 = decompose_ess_matrix(Ess_mat3,Pt_31, Pt_32,cal_mat31)

#print
print("Rotational matrix of class room:")
print("R:")
print(R1)
print("Translational matrix of class room:\n")
print("T:")
print(T1)
print("************************************************\n")

print("Rotational matrix of storage room:")
print("R:")
print(R2)
print("Translational matrix of storage room:\n")
print("T:")
print(T2)
print("************************************************\n")

print("Rotational matrix of trap room:")
print("R:")
print(R3)
print("Translational matrix of trap room:\n")
print("T:")
print(T3)
print("************************************************\n")
```

```
Rotational matrix of class room:
R:
[[ 9.99983326e-01  8.32256891e-04 -5.71454047e-03]
```

```
 [-8.47385480e-04  9.99996142e-01 -2.64547438e-03]
 [ 5.71231670e-03  2.65027269e-03  9.99980173e-01]]
Translational matrix of class room:

T:
[[-0.99985795]
 [ 0.01042343]
 [-0.01324527]]
************************************************

Rotational matrix of storage room:
R:
[[ 9.99999758e-01  1.57848756e-05 -6.95028920e-04]
 [-1.68676495e-05  9.99998786e-01 -1.55790495e-03]
 [ 6.95003485e-04  1.55791630e-03  9.99998545e-01]]
Translational matrix of storage room:

T:
[[-0.99927093]
 [ 0.03798765]
 [ 0.00381344]]
************************************************

Rotational matrix of trap room:
R:
[[ 9.99997133e-01 -1.84678795e-04 -2.38730709e-03]
 [ 1.85833385e-04  9.99999866e-01  4.83424424e-04]
 [ 2.38721749e-03 -4.83866679e-04  9.99997034e-01]]
Translational matrix of trap room:

T:
[[-9.99955628e-01]
 [-7.84418617e-04]
 [ 9.38756357e-03]]
************************************************
```

2.Rectification:

a. Apply perspective transformation to rectify images and ensure horizontal epipolar lines.

using cv2.stereoRectifyUncalibrated to find homography matrices and perform wrap perspective transform

b. Print the homography matrices (H1 and H2) for rectification.

H1 and H2 matrices are printed

```
#Define a function for rectification
def rectifiction(Img1, Img2, Fund_mat, pt1, pt2):
  #getting Homography matrices
    _, H1, H2 = cv2.stereoRectifyUncalibrated(pt1, pt2, Fund_mat, imgSize=Img1.shape[:2])

    # Do transformation to rectify the imgs
    rect_Img1 = cv2.warpPerspective(Img1, H1, (Img1.shape[1], Img1.shape[0]))
    rect_Img2 = cv2.warpPerspective(Img2, H2, (Img2.shape[1], Img2.shape[0]))

    return rect_Img1, rect_Img2,H1, H2

#call all datasets
rect_Img11,rect_Img12,H11, H12 = rectifiction(classroom_lst[0], classroom_lst[1],fund_mat1
fig, axes = plt.subplots(1, 2, figsize=(25, 25))
axes[0].imshow(cv2.cvtColor(rect_Img11, cv2.COLOR_BGR2RGB))
```

```
axes[0].imshow(cv2.cvtColor(rect_Img11, cv2.COLOR_BGR2RGB))
axes[0].set_title('ClassRoom Rectified1')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(rect_Img12, cv2.COLOR_BGR2RGB))
axes[1].set_title('ClassRoom Rectified2')
axes[1].axis('off')
plt.show

print("Homography Matrix of class room is:")
print("H1:")
print(H11)
print("H2:")
print(H12)

rect_Img21,rect_Img22,H21, H22 = rectifiction(storageroom_lst[0], storageroom_lst[1],fund_
fig, axes = plt.subplots(1, 2, figsize=(25, 25))
axes[0].imshow(cv2.cvtColor(rect_Img21, cv2.COLOR_BGR2RGB))
axes[0].set_title('StorageRoom Rectified1')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(rect_Img22, cv2.COLOR_BGR2RGB))
axes[1].set_title('StorageRoom Rectified2')
axes[1].axis('off')
plt.show

print("Homography Matrix of storage room is:")
print("H1:")
print(H21)
print("H2:")
print(H22)

rect_Img31,rect_Img32,H31, H32 = rectifiction(traproom_lst[0], traproom_lst[1],fund_mat3,
fig, axes = plt.subplots(1, 2, figsize=(25, 25))
axes[0].imshow(cv2.cvtColor(rect_Img31, cv2.COLOR_BGR2RGB))
axes[0].set_title('TrapRoom Rectified1')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(rect_Img32, cv2.COLOR_BGR2RGB))
axes[1].set_title('TrapRoom Rectified2')
axes[1].axis('off')
plt.show

print("Homography Matrix of trap room is:")
print("H1:")
print(H31)
print("H2:")
print(H32)
```

```
Homography Matrix of class room is:
H1:
[[ 2.50151761e-01 -3.95120132e-02 -8.11647928e+00]
 [ 1.83153522e-03  2.51709318e-01 -1.86895061e+00]
 [-1.08713722e-06  5.38084095e-07  2.52047540e-01]]
H2:
[[ 9.95793920e-01 -1.36527091e-02  1.53778837e+01]
 [ 6.39867774e-03  1.00000625e+00 -3.46129031e+00]
 [-7.61501030e-06  1.04404655e-07  1.00401188e+00]]
Homography Matrix of storage room is:
H1:
[[-2.05497926e-01  9.17836758e-03 -5.73895971e-01]
 [-8.10070186e-03 -2.05574483e-01  5.19134573e+00]
 [-5.29431417e-07 -1.17626775e-07 -2.04931085e-01]]
H2:
[[ 1.00049375e+00 -3.71378624e-02  3.53857234e+01]
```

```
 [ 3.91952645e-02  9.99233783e-01 -2.04298743e+01]
 [ 2.18882936e-06 -8.12483273e-08  9.98896031e-01]]
Homography Matrix of trap room is:
H1:
[[ 2.98188563e+00 -5.70974915e-02 -7.50183073e+01]
 [ 8.98444225e-03  3.07235662e+00 -2.38053004e+00]
 [ 2.05344474e-05 -2.96435961e-07  3.05896338e+00]]
H2:
[[ 1.00287208e+00  3.10233083e-03 -4.52915971e+00]
 [ 2.02099111e-03  1.00001104e+00 -1.10193026e+00]
 [ 5.32752345e-06  1.64804072e-08  9.97107316e-01]]
```

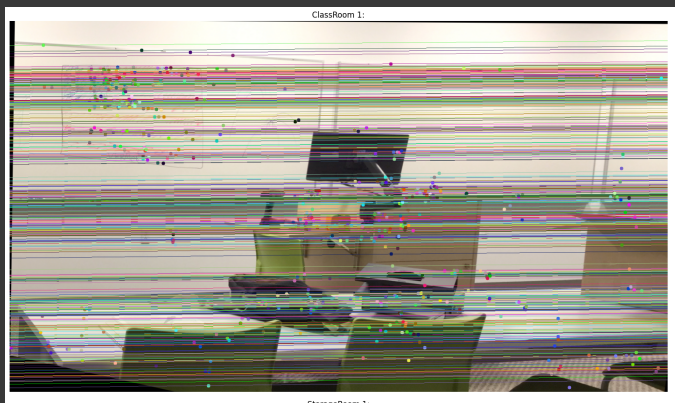
c. Visualize epipolar lines and feature points on both rectified images.

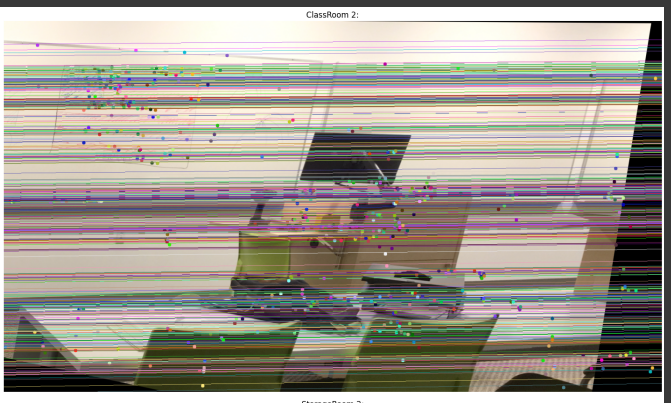
using cv2.computeCorrespondEpilines to find the epilines and use a plotting function to plot it

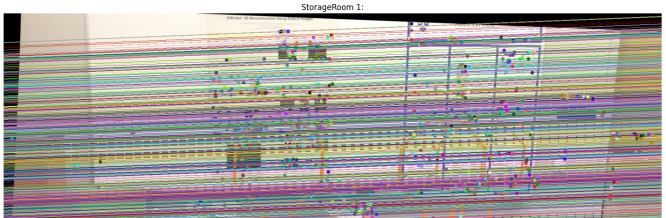
Both epilines and keypoints are plotted in same images

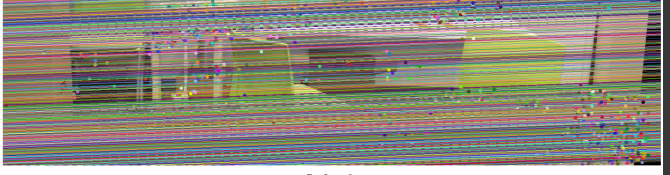
```
import matplotlib.pyplot as plt

#function to plot epilines and keypoints
def plotting(epi_lines, Pt_1, Pt_2,Img1, Img2):
  #plotting epilines
  temp1 = np.copy(Img1)
  temp2 = np.copy(Img2)
  for r, pt1,pt2 in zip(epi_lines, Pt_1, Pt_2):
    #random color plot
    color = tuple(np.random.randint(0, 255, 3).tolist())
    x_0, y_0 = map(int, [0, -r[2] / r[1]])
    x_1, y_1 = map(int, [Img1.shape[1], -(r[2] + r[0] * Img1.shape[1]) / r[1]])
    #plot cv2.line - epilines
    temp1 = cv2.line(temp1, (x_0, y_0), (x_1, y_1), color, 1)
    #plotting keypts
    pt1 = np.int32(pt1)
    pt2 = np.int32(pt2)
    temp1 = cv2.circle(temp1,(pt1[0][0], pt1[0][1]),5,color,-1)
    temp2 = cv2.circle(temp2,(pt2[0][0], pt2[0][1]),5,color,-1)
  return temp1, temp2

#function to compute epilines
def draw_epilines_key_pts(Img1, Img2, Pt_1, Pt_2,Fund_mat,key_pt1,key_pt2):
  #epilines
```

```
  #for img1
  epi_lines_1 = cv2.computeCorrespondEpilines(Pt_2, 2, Fund_mat).reshape(-1,3)
  final_Img1, _ = plotting(epi_lines_1, Pt_1, Pt_2,Img2, Img1)
  #for img2
  epi_lines_2 = cv2.computeCorrespondEpilines(Pt_1, 1, Fund_mat).reshape(-1,3)
  final_Img2, _ = plotting(epi_lines_2, Pt_1, Pt_2,Img1, Img2)

  return final_Img1, final_Img2

#calling plotting function
Img11, Img12 = draw_epilines_key_pts(rect_Img11, rect_Img12, Pt_11, Pt_12,fund_mat1,key_pt

# Matplot
fig, axes = plt.subplots(1, 2, figsize=(40, 40))
axes[0].imshow(cv2.cvtColor(Img11, cv2.COLOR_BGR2RGB))
axes[0].set_title('ClassRoom 1:')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(Img12, cv2.COLOR_BGR2RGB))
axes[1].set_title('ClassRoom 2:')
axes[1].axis('off')
plt.show()

Img21, Img22 = draw_epilines_key_pts(rect_Img21, rect_Img22, Pt_21, Pt_22,fund_mat2,key_pt
# Matplot
fig, axes = plt.subplots(1, 2, figsize=(40, 40))
axes[0].imshow(cv2.cvtColor(Img21, cv2.COLOR_BGR2RGB))
axes[0].set_title('StorageRoom 1:')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(Img22, cv2.COLOR_BGR2RGB))
axes[1].set_title('StorageRoom 2:')
axes[1].axis('off')
plt.show()

Img31, Img32 = draw_epilines_key_pts(rect_Img31, rect_Img32, Pt_31, Pt_32,fund_mat3,key_pt
# Matplot
fig, axes = plt.subplots(1, 2, figsize=(40, 40))
axes[0].imshow(cv2.cvtColor(Img31, cv2.COLOR_BGR2RGB))
axes[0].set_title('TrapRoom 1:')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(Img32, cv2.COLOR_BGR2RGB))
axes[1].set_title('TrapRoom 2:')
axes[1].axis('off')
plt.show()
```



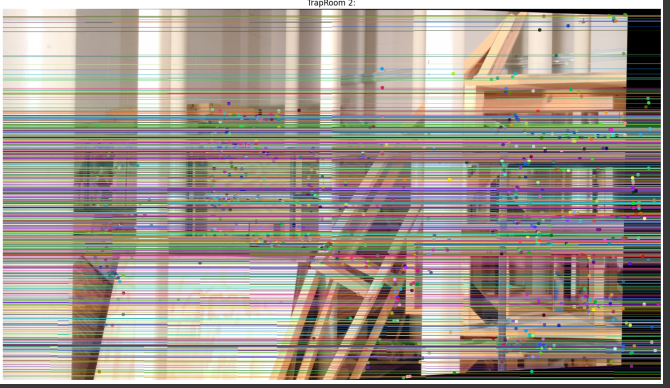






TrapRoom 1:

TrapRoom 2:



3.Compute Depth Image:

a. Calculate the disparity map representing the pixel-wise differences between the two images

b.Rescale the disparity map and save it as grayscale and color images using heat map conversion.

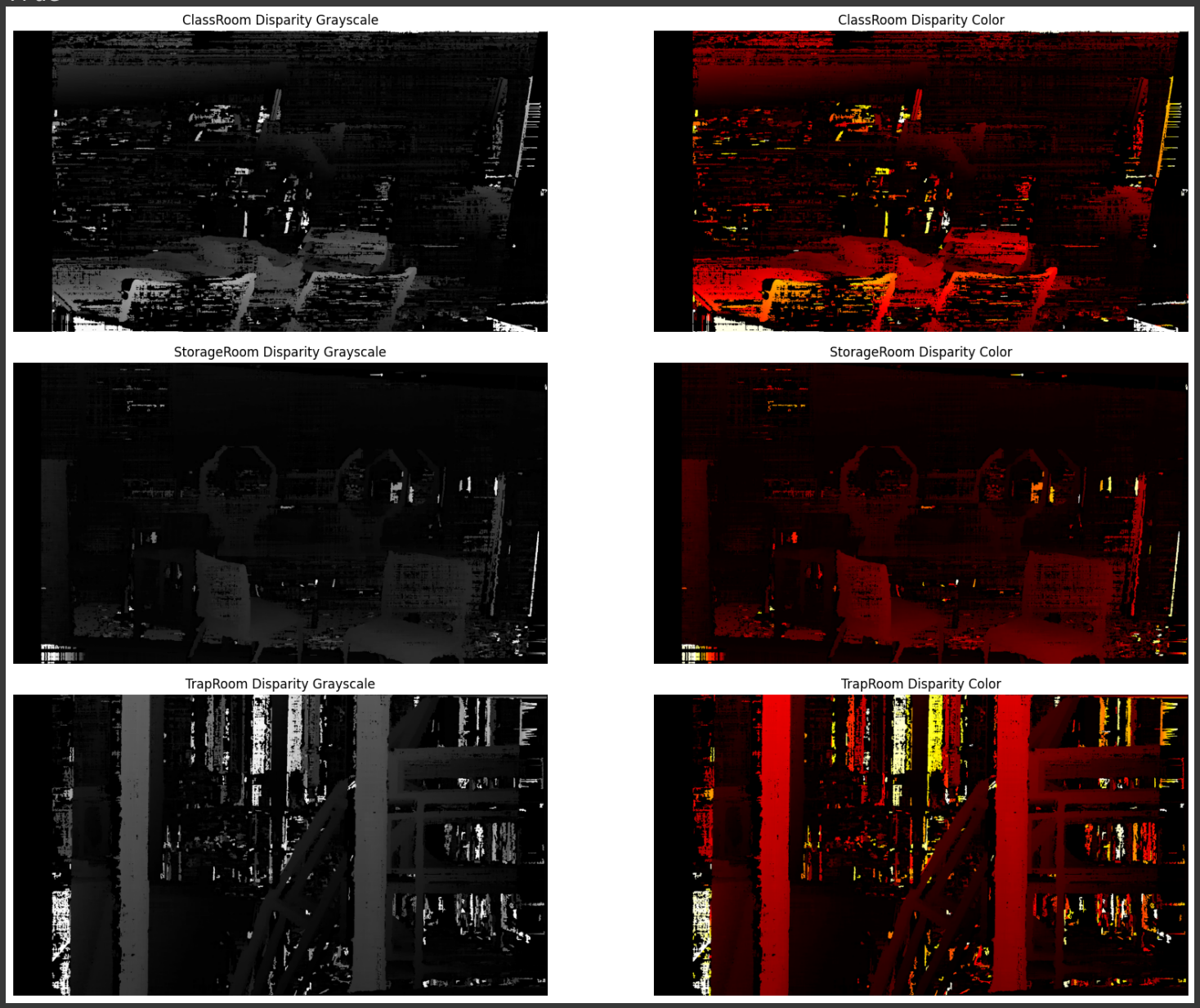
Here i have used cv2.StereoSGBM_create for creating SGBM object and then used compute to compute disparity map for rectified images.

Then Rescaled using normalizing the values and then appliedcolor map to visualize in color HOT

```
# Creating stereo SGBM object abd using disparity levels read from txt file
streo1 = cv2.StereoSGBM_create(minDisparity = 0,numDisparities = ndisp1, blockSize = 3,P1=
streo2 = cv2.StereoSGBM_create(minDisparity = 0,numDisparities = ndisp2, blockSize = 3,P1=
streo3 = cv2.StereoSGBM_create(minDisparity = 0,numDisparities = ndisp3, blockSize = 3,P1=
# Computing the disparity_Map for all datasets
disp_Map1 = streo3.compute(rect_Img11, rect_Img12)
disp_Map2 = streo2.compute(rect_Img21, rect_Img22)
disp_Map3 = streo3.compute(rect_Img31, rect_Img32)

# Rescaing-Normalizing the map and applying colormap
Disp_gray1 = cv2.normalize(disp_Map1, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX,
Disp_color1 = cv2.applyColorMap(Disp_gray1, cv2.COLORMAP_HOT)

# Display the disparity map
# Matplot
fig, axes = plt.subplots(1, 2, figsize=(20, 20))
axes[0].imshow(cv2.cvtColor(Disp_gray1, cv2.COLOR_BGR2RGB))
axes[0].set_title('ClassRoom Disparity Grayscale')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(Disp_color1, cv2.COLOR_BGR2RGB))
axes[1].set_title('ClassRoom Disparity Color')
axes[1].axis('off')
plt.show

Disp_gray2 = cv2.normalize(disp_Map2, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX,
Disp_color2 = cv2.applyColorMap(Disp_gray2, cv2.COLORMAP_HOT)

fig, axes = plt.subplots(1, 2, figsize=(20, 20))
axes[0].imshow(cv2.cvtColor(Disp_gray2, cv2.COLOR_BGR2RGB))
axes[0].set_title('StorageRoom Disparity Grayscale')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(Disp_color2, cv2.COLOR_BGR2RGB))
axes[1].set_title('StorageRoom Disparity Color')
axes[1].axis('off')
plt.show

Disp_gray3 = cv2.normalize(disp_Map3, None, alpha=0, beta=255, norm_type=cv2.NORM_MINMAX,
Disp_color3 = cv2.applyColorMap(Disp_gray3, cv2.COLORMAP_HOT)

fig, axes = plt.subplots(1, 2, figsize=(20, 20))
axes[0].imshow(cv2.cvtColor(Disp_gray3, cv2.COLOR_BGR2RGB))
axes[0].set_title('TrapRoom Disparity Grayscale')
axes[0].axis('off')
axes[1].imshow(cv2.cvtColor(Disp_color3, cv2.COLOR_BGR2RGB))
axes[1].set_title('TrapRoom Disparity Color')
axes[1].axis('off')
plt.show
```

```
#saving images to drive
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/classroom/disp
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/classroom/disp

cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/storageroom/di
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/storageroom/di

cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/traproom/dispa
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/traproom/dispa
```

```
True
```



c.Utilize the disparity information to compute depth values for each pixel.

Depth values = (baseline * focal_length) / disparity

Also add small values to avoid div by zero error

```
#Function to get depth from disparity map
def compute_Depth(baseline, focal_len, disparity):
    #adding e-3 to avoid div by zero error
    DepthMap = (baseline * focal_len) / (disparity + 1e-3)
    return DepthMap

#calling for al functions
DepthMap1 = compute_Depth(baseline1, cal_mat11[0][0], disp_Map1)
print("Depth classroom:")
print(DepthMap1)
DepthMap2 = compute_Depth(baseline2, cal_mat21[0][0], disp_Map2)
print("Depth storageroom:")
print(DepthMap2)
DepthMap3 = compute_Depth(baseline3, cal_mat31[0][0], disp_Map3)
print("Depth traproom:")
print(DepthMap3)
```

```
Depth classroom:
[[-7.40419294e+04 -7.40419294e+04 -7.40419294e+04 ...  5.43642168e+02
   5.43642168e+02  5.43642168e+02]
 [-7.40419294e+04 -7.40419294e+04 -7.40419294e+04 ...  5.43642168e+02
   5.43642168e+02  5.43642168e+02]
 [-7.40419294e+04 -7.40419294e+04 -7.40419294e+04 ...  5.42646031e+02
   5.42646031e+02  5.42646031e+02]
 ...
 [-7.40419294e+04 -7.40419294e+04 -7.40419294e+04 ...  1.18459683e+09
   1.18459683e+09  1.18459683e+09]
 [-7.40419294e+04 -7.40419294e+04 -7.40419294e+04 ...  1.18459683e+09
   1.18459683e+09  1.18459683e+09]
 [-7.40419294e+04 -7.40419294e+04 -7.40419294e+04 ...  1.18459683e+09
   1.18459683e+09  1.18459683e+09]]
Depth storageroom:
[[-2.41471538e+04 -2.41471538e+04 -2.41471538e+04 ...  3.86330314e+08
   3.86330314e+08  3.86330314e+08]
 [-2.41471538e+04 -2.41471538e+04 -2.41471538e+04 ...  3.86330314e+08
   3.86330314e+08  3.86330314e+08]
 [-2.41471538e+04 -2.41471538e+04 -2.41471538e+04 ...  3.86330314e+08
   3.86330314e+08  3.86330314e+08]
 ...
 [-2.41471538e+04 -2.41471538e+04 -2.41471538e+04 ...  3.86330314e+08
   3.86330314e+08  3.86330314e+08]
 [-2.41471538e+04 -2.41471538e+04 -2.41471538e+04 ...  3.86330314e+08
   3.86330314e+08  3.86330314e+08]
 [-2.41471538e+04 -2.41471538e+04 -2.41471538e+04 ...  3.86330314e+08
   3.86330314e+08  3.86330314e+08]]
Depth traproom:
[[-3.26669960e+04 -3.26669960e+04 -3.26669960e+04 ...  5.22639269e+08
   5.22639269e+08  5.22639269e+08]
 [-3.26669960e+04 -3.26669960e+04 -3.26669960e+04 ...  5.22639269e+08
   5.22639269e+08  5.22639269e+08]
 [-3.26669960e+04 -3.26669960e+04 -3.26669960e+04 ...  5.22639269e+08
```

```
  5.22639269e+08  5.22639269e+08]
 ...
 [-3.26669960e+04 -3.26669960e+04 -3.26669960e+04 ...  5.22639269e+08
  5.22639269e+08  5.22639269e+08]
 [-3.26669960e+04 -3.26669960e+04 -3.26669960e+04 ...  5.22639269e+08
  5.22639269e+08  5.22639269e+08]
 [-3.26669960e+04 -3.26669960e+04 -3.26669960e+04 ...  5.22639269e+08
  5.22639269e+08  5.22639269e+08]]
```

d.Generate a depth image representing the spatial dimensions of the scene.

e. Save the depth image as grayscale and color using heat map conversion for visualization.

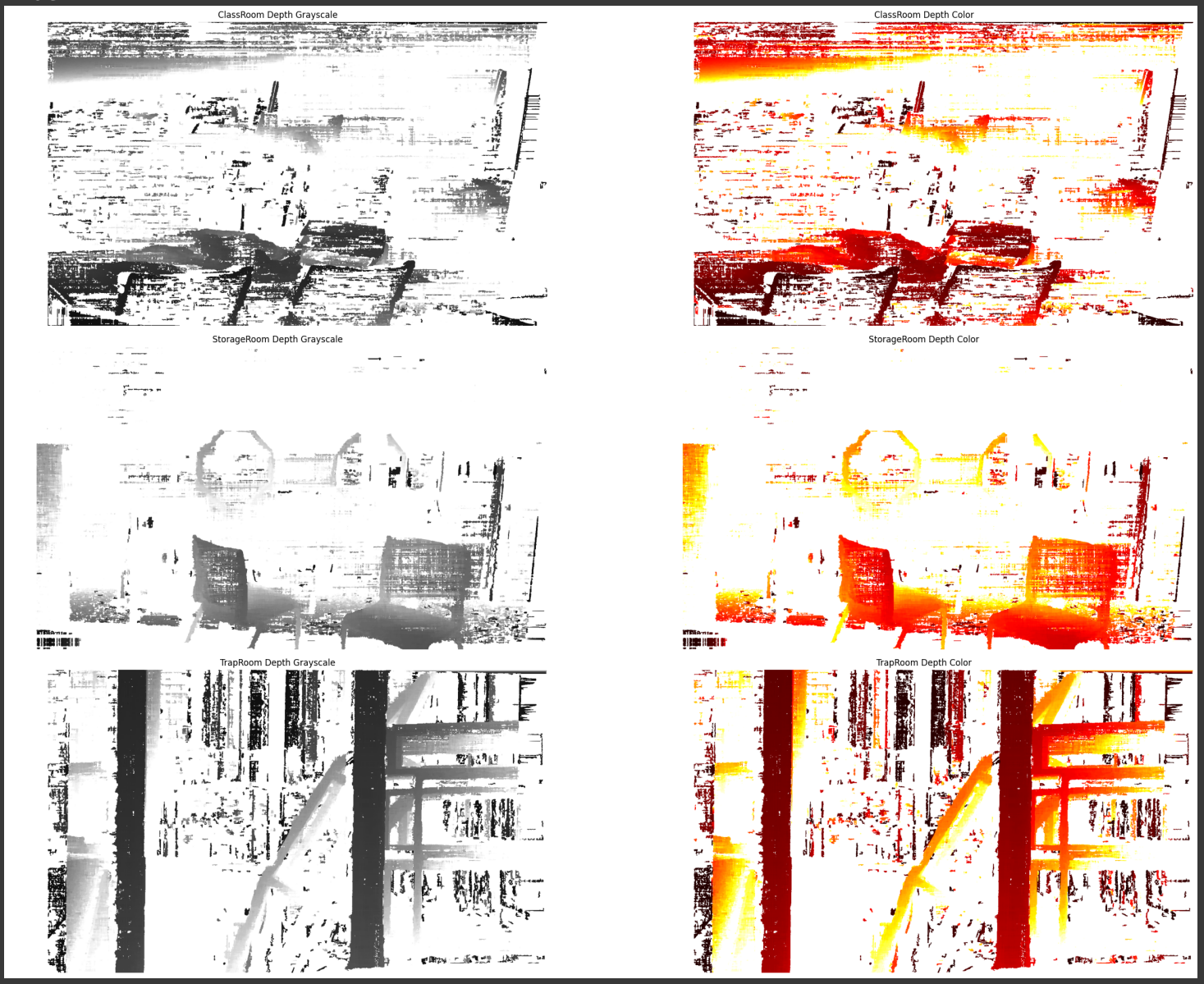
here the depth is normalized and color map is applied for displaying

Finally the images are stored in drive

```
#Normalizing and displaying color and graysacle images
DepthMap_gray1 = cv2.convertScaleAbs(DepthMap1, alpha=0.03)
depth_MapColor1 = cv2.applyColorMap(DepthMap_gray1, cv2.COLORMAP_HOT)
DepthMap_gray2 = cv2.convertScaleAbs(DepthMap2, alpha=0.06)
depth_MapColor2 = cv2.applyColorMap(DepthMap_gray2, cv2.COLORMAP_HOT)
DepthMap_gray3 = cv2.convertScaleAbs(DepthMap3, alpha=0.06)
depth_MapColor3 = cv2.applyColorMap(DepthMap_gray3, cv2.COLORMAP_HOT)

# Display the Depth Map
# Matplot lib
figure, Axs = plt.subplots(1, 2, figsize=(30, 30))
Axs[0].imshow(cv2.cvtColor(DepthMap_gray1, cv2.COLOR_BGR2RGB))
Axs[0].set_title('ClassRoom Depth Grayscale')
Axs[0].axis('off')
Axs[1].imshow(cv2.cvtColor(depth_MapColor1, cv2.COLOR_BGR2RGB))
Axs[1].set_title('ClassRoom Depth Color')
Axs[1].axis('off')
plt.show
figure, Axs = plt.subplots(1, 2, figsize=(30, 30))
Axs[0].imshow(cv2.cvtColor(DepthMap_gray2, cv2.COLOR_BGR2RGB))
Axs[0].set_title('StorageRoom Depth Grayscale')
Axs[0].axis('off')
Axs[1].imshow(cv2.cvtColor(depth_MapColor2, cv2.COLOR_BGR2RGB))
Axs[1].set_title('StorageRoom Depth Color')
Axs[1].axis('off')
plt.show
figure, Axs = plt.subplots(1, 2, figsize=(30, 30))
Axs[0].imshow(cv2.cvtColor(DepthMap_gray3, cv2.COLOR_BGR2RGB))
Axs[0].set_title('TrapRoom Depth Grayscale')
Axs[0].axis('off')
Axs[1].imshow(cv2.cvtColor(depth_MapColor3, cv2.COLOR_BGR2RGB))
Axs[1].set_title('TrapRoom Depth Color')
Axs[1].axis('off')
plt.show

#saving imgs to drive
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/classroom/dept
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/classroom/dept

cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/storageroom/de
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/storageroom/de
```

```
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/traproom/depth
cv2.imwrite('/content/drive/MyDrive/ENPM673/project3/problem2_output_images/traproom/depth
```

```
True
```



Output images are stored in drive: Link:https://drive.google.com/drive/folders/1M4VgkLukmZEdi1ztF1CUyQ45804hODwD?usp=drive_link